



ETCD v2 参考手册中文版 v0.1

作者：陆世亮，陈永刚，赖东林
联系方式：hzlushiliang@corp.netease.com

序言	1
第一章 ETCD 概述	2
第二章 ETCD 相关概念	3
2.1 RAFT 一致性算法	3
2.1.1 领导选举	3
2.1.2 日志备份	4
2.1.3 安全机制	5
2.1.4 跟随者或者候选者崩溃	5
2.1.5 集群成员变更	5
2.1.6 日志压缩	6
2.2 KEY/VALUE 键值对存储	6
2.3 ETCD 代理	7
2.4 ETCD 发现服务	8
2.4.1 协议流程	9
2.4.2 公共发现服务	11
2.5 本章小结	12
第三章 搭建 ETCD 集群	13
3.1 ETCD 源码获取	13
3.2 ETCD 启动方式	13
3.2.1 静态启动方式	14
3.2.2 ETCD 发现服务方式	17
3.2.3 DNS 发现服务方式	22
3.3 运行时健康检查	25
3.3.1 curl 命令	26
3.3.2 etcdctl 命令	26
3.4 使用 ETCD 代理	27
3.4.1 启动代理	27
3.4.2 代理转化为成员	28

3.5 本章小结.....	30
第四章 ETCD 高级配置.....	31
4.1 成员参数选项.....	31
4.2 集群参数选项.....	34
4.3 代理参数选项.....	36
4.4 安全参数选项.....	37
4.5 其它参数选项.....	39
4.6 本章小结.....	41
第五章 操作 ETCD 集群.....	42
5.1 常用键值对操作.....	42
5.1.1 设置键值对	42
5.1.2 获取键值	43
5.1.3 删除键值对	43
5.1.4 设置键值对 TTL.....	44
5.1.5 创建随机键名	44
5.1.6 创建目录	44
5.1.7 删除目录	44
5.1.8 查看目录	45
5.1.9 设置目录 TTL.....	45
5.1.10 比较并交换	45
5.1.11 比较并删除	45
5.1.12 响应错误代码	46
5.2 获取分析数据.....	47
5.2.1 领导者数据	47
5.2.2 成员自身数据	49
5.2.3 存储数据	49
5.3 监控键值对.....	50
5.4 ETCD 集群调优	53

5.4.1 时间参数调优	53
5.4.2 快照参数调优	54
5.5 安全策略	55
5.5.1 安全传输	55
5.5.2 访问控制	57
5.5 本章小结	70
第六章 运行时配置重载	71
6.1 调整集群大小	71
6.2 替换失败成员	74
6.3 故障恢复	76
6.4 更新成员	78
6.5 删除成员	79
6.6 添加成员	79
6.7 本章小结	82
第七章 ETCD 集群监控	83
7.1 监控指标解读	83
7.1.1 服务器指标	83
7.1.2 wal 指标	84
7.1.3 http 请求指标	84
7.1.4 快照指标	85
7.1.5 raft 指标	86
7.1.6 代理指标	86
7.2 PROMETHEUS 简介	87
7.2.1 安装	88
7.2.2 配置	88
7.2.3 使用	89
7.3 GRAFANA 简介	89
7.3.1 安装	90

7.3.2 配置	90
7.3.3 使用	91
7.4 本章小结	91
第八章 ETCD 性能测试与分析	92
8.1 测试工具与方案	92
8.1.1 测试工具	92
8.1.2 测试方案	93
8.2 测试结果	93
8.3 结果分析	94
8.4 本章小结	95
第九章 常见问题与解答	96
9.1 一般性问题	96
9.2 ETCD 如何处理成员关系以及健康检查	97
9.3 ETCD 代理的用途是什么	98
9.4 ETCD 与 ZOOKEEPER 的比较	98
9.5 本章小结	98
致谢	99



序言

随着云计算的普及，以及微服务架构的兴起，对于分布式一致性存储的需求逐渐成为刚需，作为当下最流行的分布式键值对存储系统，etcd 得到了广泛的关注和应用，但是当前 etcd 的资料仅仅局限于官网的英文文档，国内尚无系统的中文版资料，针对官网英文文档的顺序混乱，以及可读性差等缺陷，本文在英文文档的基础上进行整理，细化，给出了更为系统和详尽的 etcd 中文白皮书。

本文主要面向初识 etcd 的用户，以及 etcd 相关的开发研究人员，希望借助于白皮书，能够让更多的人快速学习，理解和应用 etcd，并且在 etcd 的实际应用中，能够快速的问题和解决问题，构建出更加稳定的系统。

本文一共分为九章，其中：

第一章对 ETCD 进行简单的介绍；

第二章对 ETCD 中涉及到的重要概念进行讲解；

第三章描述如何从头搭建一个 ETCD 集群；

第四章介绍 ETCD 的高级配置；

第五章讲述如何与 ETCD 集群进行交互；

第六章对 ETCD 的运行时配置重载特性进行深入解读；


第七章论述 ETCD 的监控方案；

第八章给出 ETCD 的性能测试方案，并对测试数据进行分析；

第九章给出常见的问题以及解答。

第一章 ETCD 概述

etcd 是一个分布式键值对存储系统，工作于集群的多台节点上，节点之间相互协作，以提供可靠存储的能力，集群内部有领导者选举等机制，即使出现了网络隔离的情况，也不会导致脑裂，与此同时，即使出现了个别节点的故障失败，集群也能从容应对，正常对外提供服务。集群 etcd 是一个开源项目，代码托管在 github 上，位于 <https://github.com/coreos/etcd>。



在使用 etcd 时，用户完全可以将其看成一个黑盒子，就像读写 redis 数据库一样去读写 etcd。举例来说，假设用户有多个 mysql 数据库，分布在不同的节点上，出于业务的需求，需要保证每个 mysql 服务的配置都是相同的，那么用户可以选择将 mysql 的配置信息存储在 etcd 当中，并且在每个 mysql 节点上对这些配置信息进行监控，一旦发现配置信息的变更，立即根据最新的配置信息，重启 mysql 服务，这样一来，用户只需要修改 etcd 中的配置信息，即可对所有的 mysql 服务进行修改和同步。

在 etcd 的实现层面，其使用 golang 语言开发，具有很好的跨平台特性，用户可以直接下载编译好的二进制文件，即可在本地运行 etcd 服务，省去了配置编译安装等繁琐的步骤，第三章会通过具体的案例来进行描述。

etcd 的核心机制是 raft 一致性算法，raft 算法保证了集群的一致性，使得用户可以将 etcd 集群看成是一个单机的键值对存储系统，第二章的第一小节会对 raft 算法进行介绍。

etcd 对外提供了设计良好的 restful API，客户端与 etcd 集群的交互方式基于 http 协议，通过 JSON 格式来传输数据，后续的章节中均会涉及到 API 的使用。

最后，etcd 还提供很多安全措施来保障通信数据的安全，比如说代理的使用，TLS 加解密以及访问控制等。

第二章 ETCD 相关概念

本章将介绍若干 etcd 相关的概念，这些概念基础而关键，只有在很好的理解这些概念的前提下，用户才能更好的阅读后续的章节，同时也才能更好的使用 etcd。

这些概念包括 raft 一致性算法，键值对存储，etcd 代理以及发现服务。

2.1 RAFT 一致性算法

raft 算法由斯坦福大学的两位学生 Diego Ongaro 和 John Ousterhout 提出，该算法要解决的问题是，给定一个分布式系统，系统包含若干节点，每个节点都要存储相同的数据，它们互为镜像，那么如何能够使得这些节点就数据的增删改查达成一致，也就是一致性问题。分布式使得系统具有容错的能力，因为分布式系统中不仅仅包含一个节点，而是包含了多个节点，这些节点均互为镜像，存储着相同的数据，即使其中的若干节点出现了故障，那么也能保障数据不丢失，服务不中断，这样的能力在对于现时代的 IDC 或者其它应用服务提供商而言，是最基本的需求，但是这也带来一个问题，就是一致性问题使得分布式系统的实现变得复杂，过去的很长一段时间内，一致性算法就等同于 paxos 算法，不过后者过于抽象，常人难以理解，工程实现更是难上加难，针对上述问题以及需求，raft 算法的出现，以其简单易懂的特性，立即引起了广泛的关注。

Raft 算法包含六个相互独立的部分：领导选举，日志备份，安全机制，跟随者与候选者崩溃，集群成员变更，日志压缩。

2.1.1 领导选举

集群中每个节点都有三种状态，分别是跟随者，候选者以及领导者，但是任何时刻，一个节点只能处于这三种状态中的一种，举例而言，不能同时是跟随者与候选者，也不能同时是跟随者与领导者。

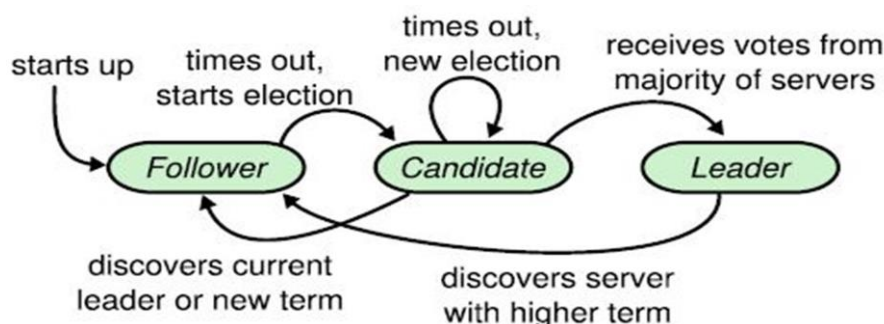
这三种状态的职能是各不相同的，对于跟随者而言，如果其收到客户端的数据库读请求，那么它可以直接处理该请求并且将响应返回给客户端，但是如果其收到的是客户端的数据库写请求，那么它必须将请求转发给领导者，由领导者来向整个集群下达日志备份的操作命令，

最后领导者再将日志备份的结果（失败或者成功）返回给跟随者，跟随者再将响应返回给客户端；

对于候选者而言，它会发起投票请求，请求集群的其它节点选举自己为领导者，请求发出之后，会等待其它节点的响应，响应结果要么投票给你，要么不投票给你，统计过后，可能存在如下的三种情况：1. 如果一个候选者发现获得了大多数节点的投票，那么它将成为领导者，随机向集群的其它成员发送心跳信息，声明自己的领导者身份，其它成员会回退成跟随者，选举结束，集群进入稳定期；2. 若是发现别的节点成为了领导者，自己就回退成跟随者，选举结束；3. 最差的情况是没人成为领导者，选票被瓜分了，没有任何一个节点获取到多数的选票，那么最终会导致超时，节点仍然会停留在候选者的状态上，并且回退一段时间，接着发起选举；

对于领导者而言，它最核心的职能就是完成日志备份，将日志同步到集群的其它成员中去，同时还需要维护集群的状态信息，进行集群管理。

这三种状态之间的切换可以通过如下的有限状态机模型来表达：



2.1.2 日志备份

领导选举完成之后，集群中唯一的领导者就确定下来了，其它节点均作为跟随者，听从领导者的指令，此时可以进行常规的日志备份操作。

客户端会将写数据库的操作请求发送给集群的成员，接收到客户端请求的成员可能是跟随者，也可能是领导者自身，如上一小节所述，如果一个跟随者接收到了客户端的数据库写请求，那么它会第一时间将请求转发给领导者，由领导者来完成日志备份的工作，如果是领导者接收到了客户端的数据库写请求，那么请求无须转发，领导者可以直接处理，领导者会将这个日志发给集群的其它所有成员，告诉其他成员，现在有一条日志需要你们写入，集群成员在收到这个命令后，会回复领导者可以写入，领导者统计各个成员的回复，如果超过一

半的成员回复是肯定的，那么领导者认为这条日志可以被写入磁盘了，此时会将最终的决定发给所有成员，告知各个成员，这条日志可以在集群范围内写入，最后日志被写入各个成员的磁盘中，最终领导者将执行成功的响应恢复给客户端，或者转发给接收到客户端请求的成员，由该成员将响应回复给客户端。

以上情况是最理想的，如果有成员出现了网络问题，或者宕机停服了，那么领导者会不断给其发送日志备份的命令，直到集群的多数成员达成一致。

2.1.3 安全机制

为了保证被选举出来的领导者拥有全部的历史日志，raft 向选举过程加入了安全机制，当一个节点收到别的节点的投票请求时，需要比较该成员的选举期数 term 以及日志序列号 ID，只有这两个都不小于自己的选举期数和日志序列号的时候，才有可能投票给该成员，这样子能够保证，被选举出来的领导者拥有全部的历史日志，能简化领导选举后的日志同步的过程，这样一来，一个领导者被选举出来后，只需要其它跟随者向自己同步日志即可，不需要领导者从其它成员处同步日志。

2.1.4 跟随者或者候选者崩溃

相比于领导者崩溃的情况，raft 对这种情况的处理相对简单，如果一个跟随者或者候选者没有响应，那么领导者会不停地向他们发送同样的请求，如果过段时间崩溃的成员恢复过来了，那么就可以继续收到同样的请求，根据这个请求就可以很快加入到集群当中去。

2.1.5 集群成员变更

在实际的使用中，用户可能希望集群成员变更不要导致停服，针对这种情况，raft 对集群成员的变更也给出了方案，该方案要求成员的变更必须一个一个成员进行的，不能同时上线或者下线多个成员，而且变更单个成员的具体做法是进行两阶段提交。

首先领导者会收到来自管理员的成员变更指令，领导者将这条指令日志发送给集群的其它成员，该日志中包含了旧集群的成员信息和新集群的成员信息，一旦收到这条指令，成员

对后续的日志备份操作，都会要求同时被旧集群的大多数成员接受，同时也要求被新集群的大多数成员接受。这个过程会一致持续下去，直到这条配置变更的日志被集群大多数成员接受，这样一来，领导者就可以将只包含新集群成员的配置变更日志发给集群的其他成员，如此一来，老集群的成员即可安全下线。这样的算法能够保证在集群配置变更的过程中，不会出现脑裂问题。

2.1.6 日志压缩

在实际的应用中，不能让日志无限制的增长下去，一来会占用大量的磁盘空间，二来会导致服务重启时需要耗费大量的时间来回放数据。

Raft 通过快照的方式来进行日志压缩，集群的成员都能够对自己的日志进行快照，快照中包含一些元数据信息，比如快照的最后一个日志序列号，以及该日志所处着的选举期。一旦成员完成了快照，就可以将这条日志以及前面的所有日志删除掉，从而释放磁盘空间。

2.2 KEY/VALUE 键值对存储

etcd 是一个 key/value 键值对存储系统，对于信息的存储，不同于 mysql 等关系型数据库，在关系型数据库中，数据被组织成表格，表格中包含多条记录，而在键值对存储系统中，信息被存储成【键名：键值】的二元信息组。

常用的键值对数据库有 redis 以及 memcached，不同的是，它们属于内存数据库，并且是单机的，而 etcd 将数据持久化到磁盘中，并且原生支持分布式，在 etcd 中，还有一个重要的概念，就是数据目录，一个键名就像是文件系统下的文件名，而文件是存放在目录中的，在 etcd 中，键名也隶属一个目录，这样一来，目录和键名就形成了一个目录树，类似于文件系统的目录树，如下图所示：



在上图中，有两个目录，分别是 config 目录和 feature-flags 目录，在 config 目录下有一个键，名称为 database，而在 feature-flags 目录下则有两个键名。

2.3 ETCD 代理

etcd 可以用作一个透明的代理。由于代理可以作为本地服务运行在每台节点上，这使得框架内的 etcd 发现变得很容易。在代理模式下，etcd 作为一个反向代理，将客户端的请求转发给一个存活的 etcd 集群。Etcd 代理并不参与 etcd 集群的一致性日志备份操作，所以它既不增加集群的弹性，也不会降低集群的写入性能。

etcd 当前支持两种代理模式：读写和只读。默认的模式是读写，这种模式下会将客户端的读请求和写请求均转发给 etcd 集群。一个只读模式的 etcd 代理只会将读请求转发给 etcd 集群，对于客户端的写请求一律返回 HTTP 501 响应。

etcd 代理会定期轮询集群中的所有节点，避免将所有连接都转发给同一个集群成员。

etcd 代理所轮询的节点名单由集群中通告的所有客户端 URL 所组成。这些客户端 URL 在每一个 etcd 集群节点的 advertise-client-urls 配置选项中指定。

etcd 代理通过验证命令行选项来发现对端的 URL。按照优先级排序，这些命令行选项分别是 discovery, discovery-srv 以及 initial-cluster。其中 initial-cluster 选项被设置成一系列逗号分割的 url，这些 url 会在启动阶段被使用，直到发现最终的 etcd 集群。

通过上述方式获取到一系列的对端 url 之后，代理会从第一个可达的对端哪里获取到一系列的客户端 url。这些客户端 url 在对端节点的 advertise-client-urls 配置选项中指定。接下来代理会每隔 30 秒与该可达对端节点建立连接，以定期刷新客户端 url 列表。

由于 etcd 代理能够从集群获取到客户端 url 的配置信息，代理的 advertise-client-urls 配置选项就不需要指定了，这也意味着，每个代理的 initial-cluster 配置选项必须要正确配置，同时要为每一个非代理兼一级对端节点正确配置 advertise-client-urls 选项。否则，发给 etcd 代理的请求会被错误的转发。千万注意不要将 advertise-client-urls 选项配置成 etcd 代理本身，否则代理会进入死循环的状态，不断将请求转发给它自己，直到代理的资源被耗尽为止。出现前面两种情况的时候，杀死 etcd 服务，指定正确的配置，并且重启 etcd。

etcd 代理的启动和对端发现过程总结如下：

1. Etcd 代理顺序执行以下的动作，直到获知集群的对端 url：
 - 1) 如果配置了代理的 discovery 选项,那么调用相应的发现服务来获取对端 url。
该对端 url 是所有一级,非代理集群成员的 initial-advertise-peer-urls 配置项的组合。
 - 2) 如果配置了代理的 discovery-srv 选项,那么对端 url 将通过 DNS 发现的方式获取到。
 - 3) 如果配置了代理的 initial-cluster 选项,那么这个选项的值就是对端 url。
 - 4) 否则将对端 url 的值设置成默认的 http://localhost:2380, 以及
http://localhost:7001。
2. 使用这些对端 url 来与集群中的非代理节点建立连接,以进一步获取到它们的客户端 url。这样一来,客户端 url 就是集群中所有非代理节点的 advertise-client-urls 选项值的组合。
3. 发往代理的客户端请求会被转发到这些获取到的客户端 url 去。

总是先启动集群的一级成员节点,然后再启动代理。一个代理必须要能够与集群成员建立连接,以获取成员上面的配置信息,当连接通道不存在的时候,代理会以激进的方式去尝试建立连接。先启动集群的非代理节点能保证代理在启动的时候获取到客户端 url。

2.4 ETCD 发现服务

etcd 发现服务被用于 etcd 集群的启动,在集群的启动阶段,通过一个共享的发现 url,发现服务协议有助于新的成员发现其它的集群成员,并且最终加入到集群中去。与 etcd 发现服务功能类似的还有 dns 发现服务,本小节仅仅介绍 etcd 发现服务,dns 发现服务会在后续的章节中进行讲解。

etcd 发现服务协议仅仅只在集群的启动阶段使用,并且不能用于运行时配置重载或者集群监控。

该协议使用一个全新的发现令牌来启动一个单独的集群。需要明确的是,一个发现令牌只能代表一个 etcd 集群。只要基于该令牌的发现服务协议启动了,即使中途失效,也不能将该令牌用于其它集群的启动。

本章节剩余部分将通过例子来演示发现的流程,这些例子使用的是自己搭建的 etcd 发

现集群。作为一项公共的 etcd 发现服务，discovery.etcd.io 的流程也是一样的，但是多了一层过滤，以清除非法的 url，自动生成 UUID，并且提供请求的过载保护。其核心机制正如白皮书中描述的一样，也是使用 etcd 集群来进行数据存储。

2.4.1 协议流程

etcd 发现服务的理念是使用一个内部的 etcd 集群来协助新集群的启动。首先，所有的新成员与发现服务进行交互，以获取到期望的成员列表，接着每个成员使用该列表来启动 etcd 服务，这个流程和使用 -initial-cluster 标识的内部流程是一致的。

在下面的流程案例中，为了便于理解，本文以 curl 命令来演示协议的每个步骤。

按照约定，etcd 的发现服务使用键名前缀 _etcd/registry。如果位于 http://example.com 的主机运行着用于服务发现的 etcd 集群，对应于发现密钥空间的完整 url 是 http://example.com/v2/keys/_etcd/registry。在这个例子中，本文将使用这个 url 前缀。

1 生成一个新的发现令牌

生成一个独特的令牌，用于标识新的集群。在接下来的步骤中，该令牌在发现密钥空间将作为一个独特的前缀。一个简单的方法是使用 uuidgen 环境变量作为令牌：

```
UUID=$(uuidgen)
```

2 指定预期的集群大小

用户需要为该令牌指定期望的集群大小。发现服务使用该大小值来判断集群的初始化成员是否都已经到位。

```
curl -X PUT http://example.com/v2/keys/_etcd/registry/${UUID}/_config/size -d
value=${cluster_size}
```

通常集群的大小是 3，5 或者 7。

3 启动 etcd 进程

在已知发现 url 的基础上，用户可以将其用于指定 -discovery 标识，并且启动 etcd 进程。如果指定了 -discovery 标识，每一个 etcd 进程都将遵循下面的几步操作。

4 自我注册

Etcd 进程要做的第一件事情就是以成员的身份将自己注册到发现 url 中去。这是通过在发现 url 中生成成员 ID 来实现的。

```
curl -X PUT
http://example.com/v2/keys/_etcd/registry/${UUID}/${member_id}?prevExist=false
-d
value="${member_name}=${member_peer_url_1}&${member_name}=${member_peer_url_2}"
```

5 检查状态

发现服务会检查发现 url 中预期的集群大小以及注册状态，以决定下一步的动作

```
curl -X GET http://example.com/v2/keys/_etcd/registry/${UUID}/_config/size
curl -X GET http://example.com/v2/keys/_etcd/registry/${UUID}
```

如果注册的成员还没全部到位，发现服务会等待其它成员的出现。

如果注册的成员数量大于预期的大小 N，发现服务会将最先出现的 N 个成员视为集群的成员。如果成员本身就在成员列表中，发现流程会成功执行，并且从成员列表中获取所有的对端 url。如果成员不在成员列表中，发现服务会返回失败，告知集群已经满员。

在 etcd 的实现中，成员在注册自己之前可能会检查集群的状态。所以如果集群满员了，那么该成员的注册会很快失败。

6 等待所有成员的出现

```
curl -X GET
http://example.com/v2/keys/_etcd/registry/${UUID}?wait=true&waitIndex=${current_etcd_index}
```

该命令会一直等待，直到发现所有成员。

2.4.2 公共发现服务

CoreOS Inc. 在 <https://discovery.etcd.io/> 提供了一项公共的发现服务，该服务提供了一些易于使用的新特性。

1 掩盖键名前缀

公共服务会将请求 `https://discovery.etcd.io/${UUID}` 导向背后真正的集群，该集群的实际 url 位于 `/v2/keys/_etcd/registry`。公共服务掩盖了注册的键名前缀，暴露给用户的是简洁的发现 url。

2 获取新的令牌

```
GET /new
```

http 的 query 请求部分如下：

```
size=${cluster_size}
```

可能的响应如下：

```
200 OK
```

```
400 Bad Request
```

200 状态码的实体数据部分包含：

```
最新生成的发现 url
```

发现服务中令牌生成的流程可以参考文章 [Creating a New Discovery Token](#) 到 [Specifying the Expected Cluster Size。](#)

3 检查发现状态

```
GET /${UUID}
```

通过请求该 UUID 的值，用户可以检查该令牌的状态，包括哪些机器已经注册到该令牌。

4 开源仓库

该仓库位于 <https://github.com/coreos/discovery.etcd.io>。用户可以使用它来部署自己的 etcd 发现服务。

2.5 本章小结

本章对 etcd 相关的概念进行介绍，涉及到 raft 一致性算法，键值对存储，etcd 代理，etcd 发现服务等四个方面，为后面章节的论述做好准备和铺垫。

第三章 搭建 ETCD 集群

本章将从头开始搭建一个 etcd 集群，对启动集群的几种方案进行了分述和比较，并给出 etcd 代理的启动方法，通过本章的学习，用户应该能够搭建属于自己的 etcd 集群。

本章使用的操作系统为 linux 操作系统。

3.1 ETCD 源码获取

etcd 源码有两种，一种是编译好的二进制可执行文件，第二种是未编译的源码文件，这里推荐用户直接下载可执行文件，省去源码配置编译安装的复杂流程。

下载二进制源码到本地

```
curl -L
https://github.com/coreos/etcd/releases/download/v2.3.7/etcd-v2.3.7-linux-amd64
.tar.gz -o etcd-v2.3.7-linux-amd64.tar.gz
```

解压源码

```
tar xzvf etcd-v2.3.7-linux-amd64.tar.gz
```

进入解压后的目录

```
cd etcd-v2.3.7-linux-amd64
```

3.2 ETCD 启动方式

etcd 包含三种启动方式，分别是静态配置方式，etcd 发现服务方式以及 dns 发现服务方式。以静态方式启动一个 etcd 集群，需要每个成员都知道集群的其它成员信息。在一些情况下，用户无法事先知道其它集群成员的 IP 地址。此时，用户可以通过发现服务的方式启动集群。

一旦 etcd 集群运行起来，可以通过运行时配置重载来添加或者删除成员。为了更好的

理解运行时配置重载的设计原理，建议用户阅读[运行时配置重载的设计文档](#)。

本章节将阐述以下三种启动 etcd 集群的机制：

- . 静态方式
- . etcd 发现
- . DNS 发现

本文将分别使用这三种机制来创建一个 3 节点的 etcd 集群，这 3 个节点的信息如下表所示：

名称	IP 地址	主机名
infra0	10.0.1.10	infra0.example.com
infra1	10.0.1.11	infra1.example.com
infra2	10.0.1.12	infra2.example.com

3.2.1 静态启动方式

由于事先知道了集群的成员，成员 IP 地址和成员数量，用户可以设置 initial-cluster 标识，以离线配置的方式启动集群。在每台节点上执行如下的命令或者设置相应的环境变量：

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380"
ETCD_INITIAL_CLUSTER_STATE=new
```

```
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380 \
--initial-cluster-state new
```

需要注意的是，initial-cluster 配置项中指定的 url 都是通告对端 url，也就是说，

这些 url 应该和每个节点上的 initial-advertise-peer-urls 标志位的值对应起来。

如果用户处于测试的目的，正在使用相同的配置来部署多个集群（或者创建和部署一个单独的集群），那么强烈建议为每一个集群指定一个独特的 initial-cluster-token 值。这样一来，即使集群之间的配置是相同的，etcd 也能为他们生成不同的集群 ID 以及成员 ID。这样子能够避免发生跨集群的通信，后者可能会导致集群的崩溃。

Etcd 监听在 listen-client-urls，以接收客户端的请求。Etcd 成员将 advertise-client-urls 中的地址通告给其它成员，代理以及客户端。用户需要保证 advertise-client-urls 中的地址对于客户端是可达的。一个常见的错误是将 advertise-client-urls 设置成 localhost，或者留白成默认值，这会导致远程的客户端无法连接 etcd。

在每个节点上使用如下的标识来启动 etcd：

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
--listen-peer-urls http://10.0.1.10:2380 \
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.
12:2380 \
--initial-cluster-state new
```

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
--listen-peer-urls http://10.0.1.11:2380 \
--listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.11:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.
12:2380 \
```

```
--initial-cluster-state new
```

```
$ etcd --name infra2 --initial-advertise-peer-urls http://10.0.1.12:2380 \
--listen-peer-urls http://10.0.1.12:2380 \
--listen-client-urls http://10.0.1.12:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.12:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.
12:2380 \
--initial-cluster-state new
```

在后续 etcd 的运行中，以 `--initial-cluster` 为前缀的命令行参数将被忽略。用户在 etcd 集群启动之后，可以随意删除环境变量或者命令行参数。如果用户稍后需要对配置进行更改（比如从集群中添加或者移除成员），请参考运行时配置重载手册。

【错误案例】

在下面的错误案例中，本文并未将当前新节点添加到初始节点列表中。如果这是一个新的集群，该节点必须添加到集群的初始节点列表中。

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
--listen-peer-urls https://10.0.1.11:2380 \
--listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.11:2379 \
--initial-cluster infra0=http://10.0.1.10:2380 \
--initial-cluster-state new
etcd: infra1 not listed in the initial cluster config
exit 1
```

在接下来的例子中，本文试图将节点 `infra0` 映射到一个新的地址 (`127.0.0.1:2380`)，该地址并不在集群的初始节点列表中 (`10.0.1.10:2380`)。如果一个节点需要监听多个地址，那么这些地址都需要配置到 “`initial-cluster`” 标识中去。

```

$ etcd --name infra0 --initial-advertise-peer-urls http://127.0.0.1:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.
12:2380 \
  --initial-cluster-state=new
etcd: error setting up initial cluster: infra0 has different advertised URLs in the
cluster and advertised peer URLs list
exit 1
  
```

如果用户使用新的配置来启动一个节点，并且试图加入集群，那么会得到一个冲突的集群 ID，而且 etcd 也会失败退出。

```

$ etcd --name infra3 --initial-advertise-peer-urls http://10.0.1.13:2380 \
  --listen-peer-urls http://10.0.1.13:2380 \
  --listen-client-urls http://10.0.1.13:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.13:2379 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra3=http://10.0.1.
13:2380 \
  --initial-cluster-state=new
etcd: conflicting cluster ID to the target cluster (c6ab534d07e8fcc4 !=
bc25ea2a74fb18b0). Exiting.
exit 1
  
```

3.2.2 ETCD 发现服务方式

在一些情况下，用户无法事先知道集群节点的 IP 地址。如果用户使用的是云提供商的

服务，或者网络启用了 DHCP，那么这种情况会很常见。在这些情况下，与其进行静态配置，用户可以使用一个已经存在的 etcd 集群来帮助启动新的集群。本文将该流程命名为“发现”。

一共有两种发现方法：

- . etcd 发现服务

- . DNS SRV 记录

本小节先描述 etcd 发现服务，下一小节描述 DNS 发现服务。

1 发现 url 的生命周期

一个发现 url 标示了一个唯一的 etcd 集群。相比于复用一個发现 url，用户应该为每一个新的集群重新创建发现 url。

此外，发现 url 仅能用于集群的初始化启动阶段。想要更改一个已经运行的集群中的成员，请参考运行时配置重载文档。

2 自定义的 etcd 发现服务

发现服务使用一个已经运行的 etcd 集群来启动新的集群。如果用户使用自己的 etcd 集群来提供发现服务，那么可以运行如下的命令来创建 url：

```
$ curl -X PUT
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83/_config/size -d value=3
```

通过设置 url 中的 size 键，用户可以创建一个发现 url，其预期的集群大小为 3。

如果用户通过发现服务启动的集群大小超过了预期值，多余的 etcd 进程将默认回退成代理。

这种情况下，用户可用的 url 是 https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83，而且 etcd 成员在启动的时候将使用该 url 进行注册。

每个成员都必须指定不同的 name 标识。Hostname 或者 machine-id 都是理想的选项。否则发现服务会因为名称冲突而失败。

现在用户可以通过相关的标识来启动每一个 etcd 节点。

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
```

```
--listen-peer-urls http://10.0.1.10:2380 \
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
--listen-peer-urls http://10.0.1.11:2380 \
--listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.11:2379 \
--discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83
```

```
$ etcd --name infra2 --initial-advertise-peer-urls http://10.0.1.12:2380 \
--listen-peer-urls http://10.0.1.12:2380 \
--listen-client-urls http://10.0.1.12:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.12:2379 \
--discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83
```

这样一来每个成员都会使用自定义的发现服务来自我注册，一旦所有成员均注册完毕，集群即可启动完毕。

3 公用的 etcd 发现服务

如果没有可用的 etcd 集群，用户可以使用位于 `discovery.etcd.io` 的公共发现服务。用户可以使用 `new` 关键字来创建一个私有的发现 url，操作如下：

```
$ curl https://discovery.etcd.io/new?size=3
```

返回的响应如下：

```
https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

这样会创建一个初始预期大小为 3 的集群。如果没有指定大小，默认的大小也是 3。

如果用户通过发现服务启动的集群大小超过了预期值，多余的 etcd 进程将默认回退成

代理。

接下来可以通过指定环境变量或者指定配置项来使用该 url。

```
ETCD_DISCOVERY=https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
-discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

每个成员都必须指定不同的 name 标识。Hostname 或者 machine-id 都是理想的选项。否则发现服务会因为名称冲突而失败。

现在用户可以通过相关的标识来启动每一个 etcd 节点。

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
--listen-peer-urls http://10.0.1.10:2380 \
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
--listen-peer-urls http://10.0.1.11:2380 \
--listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.11:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

```
$ etcd --name infra2 --initial-advertise-peer-urls http://10.0.1.12:2380 \
--listen-peer-urls http://10.0.1.12:2380 \
--listen-client-urls http://10.0.1.12:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.12:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

这样一来每个成员都会使用自定义的发现服务来自我注册，一旦所有成员均注册完毕，集群即可启动完毕。

用户也可以通过环境变量 ETCD_DISCOVERY_PROXY 来指定 http 代理，通过 http 代理来连接到发现服务。

4 错误和警告的案例

发现服务器错误

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
--listen-peer-urls http://10.0.1.10:2380 \
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

错误响应如下：

```
etcd: error: the cluster doesn't have a size configuration value in
https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de/_config
exit 1
```

用户错误

如果发现服务已经注册了预期数量的节点，并且用户明确禁止了 `discovery-fallback` 标识，那么这种错误就会发生。

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
--listen-peer-urls http://10.0.1.10:2380 \
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de \
--discovery-fallback exit
```

错误响应如下：

```
etcd: discovery: cluster is full
exit 1
```

警告

这个警告是无害的，只是通知用户该发现 url 在当前节点上会被忽略。

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
--listen-peer-urls http://10.0.1.10:2380 \
```

```
--listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
--advertise-client-urls http://10.0.1.10:2379 \
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de
```

错误响应如下：

```
etcdserver: discovery token ignored since a cluster has already been initialized. Valid log found at
/var/lib/etcd
```

3.2.3 DNS 发现服务方式

DNS 的 SRV 记录可以作为一种发现机制。`-discovery-srv` 标识可以指定包含发现 SRV 记录的 DNS 域名。下面的 DNS SRV 记录将会顺序被查找：

```
. _etcd-server-ssl._tcp.example.com
. _etcd-server._tcp.example.com
```

如果找到了 `_etcd-server-ssl._tcp.example.com`，那么 etcd 会尝试基于 ssl 的启动流程。

为了帮助客户端发现 etcd 集群，下面的 DNS SRV 记录将被顺序查找：

```
. _etcd-client._tcp.example.com
. _etcd-client-ssl._tcp.example.com
```

如果找到了 `_etcd-client-ssl._tcp.example.com`，客户端会尝试通过 ssl 与 etcd 集群进行通信。

1 创建 DNS SRV 记录

```
$ dig +noall +answer SRV _etcd-server._tcp.example.com
_etcd-server._tcp.example.com. 300 IN SRV 0 0 2380 infra0.example.com.
_etcd-server._tcp.example.com. 300 IN SRV 0 0 2380 infra1.example.com.
_etcd-server._tcp.example.com. 300 IN SRV 0 0 2380 infra2.example.com.
```

```
$ dig +noall +answer SRV _etcd-client._tcp.example.com
_etcd-client._tcp.example.com. 300 IN SRV 0 0 2379 infra0.example.com.
```

```
_etcd-client._tcp.example.com. 300 IN SRV 0 0 2379 infra1.example.com.
_etcd-client._tcp.example.com. 300 IN SRV 0 0 2379 infra2.example.com.
```

```
$ dig +noall +answer infra0.example.com infra1.example.com infra2.example.com

infra0.example.com. 300 IN A 10.0.1.10
infra1.example.com. 300 IN A 10.0.1.11
infra2.example.com. 300 IN A 10.0.1.12
```

####使用 DNS 启动 etcd 集群

Etcd 集群成员可以监听在域名或者 IP 地址，启动过程会解析对应的 DNS A 类型的记录。

位于--initial-advertise-peer-urls 的被解析地址必须包含在被解析出来的 SRV 目标中。Etcd 成员通过读取解析后的地址来判断自己是否属于 SRV 记录中定义的集群。

```
$ etcd --name infra0 \
--discovery-srv example.com \
--initial-advertise-peer-urls http://infra0.example.com:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster-state new \
--advertise-client-urls http://infra0.example.com:2379 \
--listen-client-urls http://infra0.example.com:2379 \
--listen-peer-urls http://infra0.example.com:2380
```

```
$ etcd --name infra1 \
--discovery-srv example.com \
--initial-advertise-peer-urls http://infra1.example.com:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster-state new \
--advertise-client-urls http://infra1.example.com:2379 \
--listen-client-urls http://infra1.example.com:2379 \
--listen-peer-urls http://infra1.example.com:2380
```

```
$ etcd --name infra2 \  
--discovery-srv example.com \  
--initial-advertise-peer-urls http://infra2.example.com:2380 \  
--initial-cluster-token etcd-cluster-1 \  
--initial-cluster-state new \  
--advertise-client-urls http://infra2.example.com:2379 \  
--listen-client-urls http://infra2.example.com:2379 \  
--listen-peer-urls http://infra2.example.com:2380
```

除了使用域名，用户也可以使用 IP 地址来启动集群：

```
$ etcd --name infra0 \  
--discovery-srv example.com \  
--initial-advertise-peer-urls http://10.0.1.10:2380 \  
--initial-cluster-token etcd-cluster-1 \  
--initial-cluster-state new \  
--advertise-client-urls http://10.0.1.10:2379 \  
--listen-client-urls http://10.0.1.10:2379 \  
--listen-peer-urls http://10.0.1.10:2380
```

```
$ etcd --name infra1 \  
--discovery-srv example.com \  
--initial-advertise-peer-urls http://10.0.1.11:2380 \  
--initial-cluster-token etcd-cluster-1 \  
--initial-cluster-state new \  
--advertise-client-urls http://10.0.1.11:2379 \  
--listen-client-urls http://10.0.1.11:2379 \  
--listen-peer-urls http://10.0.1.11:2380
```

```

$ etcd --name infra2 \
--discovery-srv example.com \
--initial-advertise-peer-urls http://10.0.1.12:2380 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster-state new \
--advertise-client-urls http://10.0.1.12:2379 \
--listen-client-urls http://10.0.1.12:2379 \
--listen-peer-urls http://10.0.1.12:2380

```

1) Etcd 代理配置

DNS SRV 记录同样可以用于配置运行在代理模式的 etcd 节点：

```
$ etcd --proxy on --discovery-srv example.com
```

2) Etcd 客户端配置

DNS SRV 记录同样可以用于帮助客户端发现 etcd 集群。

官方的 etcd/client 支持 DNS 发现。

通过指定 --discovery-srv 选项，Etdctl 同样支持 DNS 发现。

```
$ etcdctl --discovery-srv example.com set foo bar
```

3) 错误案例

用户可能会遇到类似 cannot find local etcd \$name from SRV records. 这样的错误。这说明 etcd 成员没有在 SRV 记录定义的集群中发现自己。标识 --initial-advertise-peer-urls 解析出来的地址必须匹配从 SRV 目标中解析出来的地址。

3.3 运行时健康检查

在上一小节 3.2 中，系统介绍了 etcd 的三种启动方式，当 etcd 集群启动之后，用户首先要做的就是确认集群的健康状态，也就是进行健康检查。

etcd 对外提供了特定的 API，以供用户进行集群的健康检查，用户可以通过 curl 命令或者 etcd 自带的 etcdctl 命令来查看集群的状态。

3.3.1 curl 命令

在最底层的实现上，etcd 通过 http 的 /health 路径将健康信息暴露给用户，这些信息是以 JSON 格式传输的。如果 http 响应返回的是 {"health": "true"}，那么就说明集群是健康的。值得注意的是，/health 路径在版本 2.2 的 etcd 中还只是实验性质的功能。

```
$ curl -L http://10.0.1.12:2379/health
```

如果集群健康，返回的响应如下：

```
{"health": "true"}
```

上面的例子显示，集群的状态是健康的

3.3.2 etcdctl 命令

用户也可以使用 etcdctl 来获取集群层面的健康信息。该工具会和集群的所有成员建立连接，获取它们的健康信息，并且汇总给用户。

```
$/etcdctl cluster-health
```

返回的响应如下：

```
member 8211f1d0f64f3269 is healthy: got healthy result from http://10.0.1.11:12379
member 91bc3c398fb3c146 is healthy: got healthy result from http://10.0.1.12:22379
member fd422379fda50e48 is healthy: got healthy result from http://10.0.1.13:32379
cluster is healthy
```

上面的例子显示，集群的状态是健康的

3.4 使用 ETCD 代理

本章的前面 3 个小节描述了如何搭建一个 etcd 集群，在实际的生产环境中，用户可能需要在集群与客户端之间加入代理，本小节将描述如何启动一个 etcd 代理，在启动代理之后，有些情况下，用户需要将代理的角色进行转换，转换成集群成员，以提升集群的弹性，本小节也会描述将一个代理转化为成员的步骤。

3.4.1 启动代理

为了以代理模式启动 etcd，用户需要提供三个选项：proxy，listen-client-urls，以及 initial-cluster(用于发现)。

为了启动一个读写代理，设置 -proxy on; 为了启动一个只读代理，设置 -proxy readonly。

代理会监听在 listen-client-urls，并且将请求转发给从 initial-cluster 或者 discovery 中获取到的 etcd 集群。

---静态配置方式

要让启动的代理去连接静态配置的 etcd 集群，指定 initial-cluster 选项，举例如下：

```
etcd --proxy on \
--listen-client-urls http://127.0.0.1:2379 \
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380
```

---发现服务方式

如果使用发现服务去启动一个 etcd 集群，用户也可以用同样的发现服务去启动一个代理。

为了使用发现服务去启动代理，需要指定 discovery 选项。代理会一直等到 discovery url 指定的 etcd 集群启动完毕，才开始转发请求。

```
etcd --proxy on \
--listen-client-urls http://127.0.0.1:2379 \
```



```
--discovery https://discovery.etcd.io/3e86b59982e49066c5d813af1c2e2579cbf573de \
```

---使用发现服务回退到代理模式

如果用户使用发现服务启动的 etcd 节点数量超过了事先指定的大小，那么多余的 etcd 进程会默认回退成读写模式的代理。它们会按照上述的方法将请求转发到 etcd 集群中去。举个例子，如果用户创建了一个大小为 5 的发现 url，并且使用该发现 url 启动了 10 个 etcd 进程，那么最终形成的集群中将包含 5 个 etcd 成员以及 5 个代理。值得注意的是这种行为可以通过设置 discovery-fallback='exit' 来禁止。

3.4.2 代理转化为成员

代理虽然不参与一致性操作，但仍然是 etcd 集群的成员之一。任何情况下，代理都不会自动转换为参与一致性操作的集群成员。

如果用户需要将代理转化成集群成员，需要执行以下的四步操作：

- . 使用 etcdctl 将当前代理作为一个集群成员添加到当前集群中去
- . 停止当前代理进程或者服务
- . 删除当前已经存在的代理数据目录
- . 以新的成员配置信息重启 etcd 进程

下面通过一个例子来描述如何将代理转化为集群的成员。

假设当前有一个 etcd 集群，包含一个集群成员和一个代理，集群的信息如下：

节点名称	IP 地址
Infra0	10.0.1.10
Proxy0	10.0.1.11

该例子将演示如何将一个集群的代理提升为集群的成员。在完成以下 4 步操作之后，这个集群最终会包含两个集群成员。

---将一个新的成员添加到当前集群

首先，使用 etcdctl 将该代理添加到集群中，生成的环境变量可用于正确配置新的成员：

```
$ etcdctl -endpoint http://10.0.1.10:2379 member add infra1 http://10.0.1.11:2380
```

返回的响应如下：

```
added member 9bf1b35fc7761a23 to cluster
```

```
ETCD_NAME="infra1"
```

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380"
```

```
ETCD_INITIAL_CLUSTER_STATE=existing
```

---杀死代理进程

杀死当前代理以清除其位于磁盘的状态，并且使用新的配置来重启它

```
px aux | grep etcd
```

```
kill %etcd_proxy_pid%
```

或者(如果用户使用是以 systemd 服务的形式启动的 etcd 代理)

```
sudo systemctl stop etcd
```

---删除存在的代理数据目录

```
rm -rf %data_dir%/proxy
```

---以新成员的身份重启 etcd 服务

最终，启动这个重新配置后的成员，并且保证它正确地加入了集群。

```
$ export ETCD_NAME="infra1"
```

```
$ export ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380"
```

```
$ export ETCD_INITIAL_CLUSTER_STATE=existing
```

```
$ etcd --listen-client-urls http://10.0.1.11:2379 \
```

```
--advertise-client-urls http://10.0.1.11:2379 \
```

```
--listen-peer-urls http://10.0.1.11:2380 \
```

```
--initial-advertise-peer-urls http://10.0.1.11:2380 \
```

```
--data-dir %data_dir%
```

如果基于 systemd 启动 etcd，需要修改 service 文件，加入正确的配置信息，并且重启服务：

```
sudo systemctl restart etcd
```

3.5 本章小结

本章首先描述了如何从头搭建一个 etcd 集群，重点介绍三种 etcd 集群的启动方式，接着介绍了如何启动一个 etcd 代理，并且给出了将代理转化为集群成员的具体步骤，通过本章的内容，用户可以掌握最基本的 etcd 集群搭建方案。

-wal-dir

. 到指定 wal 目录的路径。如果设置了该标识，etcd 会将 WAL 文件写入该目录，默认情况下是写入数据目录的。这种用法可以指定所要使用的硬盘，进而避免日志和其它 IO 操作之间的竞争。

. 默认值: ""

. 环境变量名称: ETCD_WAL_DIR

-snapshot-conut

. 当提交的事务达到多少时，将快照写入磁盘

. 默认值: " 10000 "

. 环境变量名称: ETCD_SNAPSHOT_COUNT

-heartbeat-interval

. 心跳间隔的时长(单位毫秒)

. 默认: " 100 "

. 环境变量: ETCD_HEARTBEAT_INTERVAL

-election-timeout

. 选举超时的时长(单位毫秒)。具体的细节参见后续的 etcd 调优章节。

. 默认值: " 1000 "

. 环境变量名称: ETCD_ELECTION_TIMEOUT

-listen-peer-urls

. 监听对端节点流量所在的 url 列表。该标识告诉 etcd 要在指定的 scheme://IP:port 地址上接受来自对端节点的连接请求。方案可以是 http 或者 https。如果 IP 被设置成 0.0.0.0，那么 etcd 会在所有网口的指定端口进行监听。如果同时给出了 IP 和端口，etcd 会在指定的网口和端口进行监听。可以使用多个 url 来指定用于监听的多个地址和端口。Etcd 会响应针对列表地址中的请求。

. 默认值: http://localhost:2380, http://localhost:7001

- . 环境变量名称: ETCD_LISTEN_PEER_URLS
- . 例子: `http://10.0.0.1:2380`
- . 非法例子: “`http://example.com:2380`” (域名不能用于绑定地址)

-listen-client-urls

. 用于监听客户端流量的 url 列表。该标识告诉 etcd 要在指定的 `scheme://IP:port` 地址上接受来自对客户端的连接请求。方案可以是 `http` 或者 `https`。如果 IP 被设置成 `0.0.0.0`, 那么 etcd 会在所有网口的指定端口进行监听。如果同时给出了 IP 和端口, etcd 会在指定的网口和端口进行监听。可以使用多个 url 来指定用于监听的多个地址和端口。Etcd 会响应针对列表地址中的请求。

- . 默认值: `http://localhost:2379, http://localhost:4001`
- . 环境变量名称: ETCD_LISTEN_CLIENT_URLS
- . 例子: `http://10.0.0.1:2379`
- . 非法例子: “`http://example.com:2379`” (域名不能用于绑定地址)

-max-snapshots

- . 保留的最大快照文件数量(0 表示没有上限)
- . 默认值: 5
- . 环境变量名称: ETCD_MAX_SNAPSHOTS
- . windows 用户的默认值是无上限, 但是推荐使用的大小是 5(或者根据用户的偏好进行设置)

-max-wals

- . 保留的 wal 文件的最大数量(0 表示无上限)
- . 默认值: 5
- . 环境变量名称: ETCD_MAX_WALS
- . windows 用户的默认值是无上限, 但是推荐使用的大小是 5(或者根据用户的偏好进行设置)

-cors

- . 用于 CORS (跨区域资源共享) 的白名单, 逗号分割
- . 默认值: none
- . 环境变量名称: ETCD_CORS

4.2 集群参数选项

--initial 前缀的标识用于启动新成员 (静态启动, 发现服务启动或者运行时配置重载), 在重启成员时被忽略

--discovery 标识在使用发现服务时需要指定。

-initial-advertise-peer-urls

. 通告给集群其它节点的, 本地成员的对端 url 列表。这些地址用于在集群范围内传输 etcd 数据。至少有一个地址要能够路由至所有的集群成员。这些 url 可以包含域名。

- . 默认值: http://localhost:2380, http://localhost:7001
- . 环境变量名称: ETCD_INITIAL_ADVERTISE_PEER_URLS
- . 例子: http://example.com:2380, http://10.0.0.1:2380

-initial-cluster

- . 启动时的集群初始化配置
- . 默认值: “default=http://localhost:2380, default=http://localhost:7001”
- . 环境变量名称: ETCD_INITIAL_CLUSTER
- . 键的名称是每个节点所提供的--name 标识的值。默认使用 default, 因为--name 标识的默认值就是 default。

-initial-cluster-state

. 初始化的集群状态 (“new” 或者 “existing”)。启动阶段使用静态方式或者 DNS 方式时, 将所有成员的该值设置成 “new”。如果这个选项被设置成 “existing”, etcd 尝试加入一个已经存在的集群。如果设置了错误的值, etcd 会尝试去启动, 但是最终是安全

的退出。

- . 默认值: “new”
- . 环境变量: ETCD_INITIAL_CLUSTER_STATE

-initial-cluster-token

- . 初始化的集群令牌, 用于 etcd 集群的启动。
- . 默认值: “etcd-cluster”
- . 环境变量名称: ETCD_INITIAL_CLUSTER_TOKEN

-advertise-client-urls

- . 当前成员的客户端 url 列表, 将会被通告给集群其它成员。这些 url 可以包含域名。
- . 默认值: http://localhost:2379, http://localhost:4001
- . 环境变量名称: ETCD_ADVERTISE_CLIENT_URLS
- . 例子: http://example.com:2379, http://10.0.0.1:2379
- . 当作为一个集群成员, 而且使用代理特性的时候, 用户需要非常小心的使用诸如 http://localhost:2379 这样的 url。这样会导致环路, 因为代理会将请求转发给自己, 直到它的资源(内存, 文件句柄)最终被耗尽

-discovery

- . 用于启动集群的发现 url
- . 默认值: none
- . 环境变量名称: ETCD_DISCOVERY

-discovery-srv

- . 用于启动集群的 DNS srv 域
- . 默认值: none
- . 环境变量名称: ETCD_DISCOVERY_SRV

-discovery-fallback

- . 当发现服务失败时的预期行为(“exit” 或者 “proxy”)
- . 默认值: “proxy”
- . 环境变量名称: ETCD_DISCOVERY_FALLBACK

-discovery-proxy

- . 用于导向发现服务的 http 代理
- . 默认值: none
- . 环境变量名称: ETCD_STRICT_RECONFIG_CHECK

-strict-reconfig-check

- . 拒绝会导致仲裁失败的配置重载请求
- . 默认值: “false”
- . 环境变量名称: ETCD_STRICT_RECONFIG_CHECK

4.3 代理参数选项

-proxy

- . 设置代理模式(“off”, “readonly” 或者 “on”)
- . 默认值: “off”
- . 环境变量名称: ETCD_PROXY

-proxy-failure-wait

- . 端节点在重新被转发请求前, 保持在失败状态的时间长度, 单位是毫秒
- . 默认值: 5000
- . 环境变量名称: ETCD_PROXY_FAILURE_WAIT

-proxy-refresh-interval

- . 端节点刷新的时长, 单位是毫秒
- . 默认值: 30000

. 环境变量名称: ETCD_PROXY_REFRESH_INTERVAL

-proxy-dial-timeout

. 呼叫超时的时间, 或者使用 0 来关闭超时

. 默认值: 1000

. 环境变量名称: ETCD_PROXY_DIAL_TIMEOUT

-proxy-write-timeout

. 写入的超时时长, 单位是毫秒, 设置成 0 时关闭超时

. 默认值: 5000

. 环境变量名称: ETCD_PROXY_WRITE_TIMEOUT

-proxy-read-timeout

. 读数据的超时时长, 单位是毫秒, 设置成 0 时关闭超时

. 在使用监视功能的时候请勿改动这个配置, 因为监视功能是基于长轮训请求实现的

. 默认值: 0

. 环境变量名称: ETCD_PROXY_READ_TIMEOUT

4.4 安全参数选项

-ca-file[该标识已作废]

. 客户端服务器的 TLS CA 文件路径。--ca-file ca.crt 的配置方法可以使用

--trusted-ca-file ca.crt - client-cert-auth 来代替, 这两个配置方法对于 etcd 而言都是一样的

. 默认值: none

. 环境变量名称: ETCD_CA_FILE

-cert-file

. 客户端服务器的 TLS cert 文件路径

. 默认值: none

. 环境变量名称: ETCD_CERT_FILE

-key-file

. 客户端服务器的 TLS key 文件路径

. 默认值: none

. 环境变量名称: ETCD_KEY_FILE

-client-cert-auth

. 要求客户端进行身份认证

. 默认值: false

. 环境变量名称: ETCD_CLIENT_CERT_AUTH

-trusted-ca-file

. 客户端服务器的 TLS 可信赖 CA key 文件路径

. 默认值: none

. 环境变量名称: ETCD_TRUSTED_CA_FILE

-peer-ca-file[该标识已作废]

. 对端 etcd 服务器的 TLS CA 文件路径。--peer-ca-file ca.crt 的配置方法可以使用 --peer-trusted-ca-file ca.crt --peer-client-cert-auth 来代替，这两个配置方法对于 etcd 而言都是一样的

. 默认值: none

. 环境变量名称: ETCD_PEER_CA_FILE

-peer-cert-file

- . 对端 etcd 服务器的 TLS cert 文件路径
- . 默认值: none
- . 环境变量名称: ETCD_PEER_CERT_FILE

-peer-key-file

- . 对端 etcd 服务器的 TLS key 文件路径
- . 默认值: none
- . 环境变量名称: ETCD_PEER_KEY_FILE

-peer-client-cert-auth

- . 要求对端 etcd 服务器进行身份认证
- . 默认值: false
- . 环境变量名称: ETCD_PEER_CLIENT_CERT_AUTH

-peer-trusted-ca-file

- . 对端 etcd 服务器的 TLS 可信赖 CA 文件路径
- . 默认值: none
- . 环境变量名称: ETCD_PEER_TRUSTED_CA_FILE

4.5 其它参数选项

【日志标识】

-debug

- . 将所有 go 语言子模块的默认日志模式调整成 DEBUG
- . 默认值: false (默认情况下所有子模块的日志模式是 INFO)
- . 环境变量名称: ETCD_DEBUG

-log-package-levels

- . 单独设置某个 etcd 子模块的日志模式。比如 etcdserver=WARNING, security=DEBUG

- . 默认值: false(默认情况下所有子模块的日志模式是 INFO)
- . 环境变量名称: ETCD_LOG_PACKAGE_LEVELS

【不安全的标识】

在使用如下的不安全标识时要格外小心,因为它可能导致一致性协议无法正常工作。举例子说,如果集群的其它成员仍然存活,那么这个配置项可能会导致灾难性的后果。请遵照指示使用该标识。

-force-new-cluster

. 强制创建一个新的集群,该集群只包含一个成员。这会强制更改配置,将其它存活的成员剔除出集群,并且只添加它自己到集群中去。这可以用于恢复备份。

- . 默认值: false
- . 环境变量名称: ETCD_FORCE_NEW_CLUSTER

【实验性的标识】

-experimental-v3demo

- . 使能实验性的版本 3 的 API。
- . 默认值: false
- . 环境变量名称: ETCD_EXPERIMENTAL_V3DEMO

【其它标识】

-version

- . 打印版本号并且退出
- . 默认值: false

【性能分析标识】

-enable-pprof

- . 通过 http 服务器,对外提供运行时的性能数据分析。http 服务器的地址是客户端 url+ “/debug/pprof”
- . 默认值: false

4.6 本章小结

本章主要对 etcd 的高级配置选项进行了逐个的说明，在实际的使用中，用于可以根据需求进行适当的配置。

第五章 操作 ETCD 集群

通过前面章节的叙述，用户已经能够搭建正常运行的 etcd 集群，并且可以根据实际场景的需要，来针对性的配置 etcd 集群，而 etcd 集群的客户端交互，集群管理以及安全策略等方面，则是本章的主要内容。

5.1 常用键值对操作

Etcd 是键值对数据库，日常的使用也多围绕键值对进行，包括键值对的增删改查，设置 TTL 生存周期，由于 etcd 中同时有键值对目录的概念，所有也包括一些目录的操作，操作的同时也可以增加预判条件，在本节的末尾，还会给出各种操作的错误代码，方便用户排查问题。

5.1.1 设置键值对

假设用户要创建的键名是 “message”，键值是 “hello etcd”，etcd 集群中有一个成员运行在本地 127.0.0.1:2379，则可以通过 curl 或者 etcdctl 命令来进行操作：

```
curl http://127.0.0.1:2379/v2/keys/message -XPUT -d value="hello etcd"
```

成功的响应如下：

```
{
  "action": "set",
  "node": {
    "createdIndex": 2,
    "key": "/message",
    "modifiedIndex": 2,
    "value": "hello etcd"
  }
}
```

```
}
```

上述的响应中包含若干属性，这些属性的含义是：

.action: 用户指定的操作类型，通过 PUT 方法来设置键值对时，对应的 action 就是”set”

.node.key: 用户指定设置的键名，由于 etcd 中包含键目录的概念，所以这里是键的绝对路径

.node.value: 用户指定设置的键值

.node.createdIndex: 这是一个单调增的整数，对同一个键名而言，每次修改，这个值都会增加 1

.node.modifiedIndex: 使用上类似 node.createdIndex，不过这个值代表的是所有已经持久化到磁盘的操作，所以这个值总是小于等于 node.createdIndex 的大小。

对于上述的响应，后续不再继续分析，只给出操作的命令

另一方面，用户也可以选择使用 etcdctl 来设置键值，命令如下：

```
etcdctl --endpoints=http://127.0.0.1:2379 set message "Hello etcd"
```

如果要设置的键值对已经存在，那么上述的命令将覆盖原来的键值，也就是键值的更新。

5.1.2 获取键值

假设用户要获取的键名是 “message”，etcd 集群中有一个成员运行在本地 127.0.0.1:2379，则可以通过 curl 或者 etcdctl 命令来进行操作：

```
curl http://127.0.0.1:2379/v2/keys/message
```

另一方面，用户也可以选择使用 etcdctl 来获取键值，命令如下：

```
etcdctl --endpoints=http://127.0.0.1:2379 get message
```

5.1.3 删除键值对

假设用户要删除的键名是 “message”，etcd 集群中有一个成员运行在本地 127.0.0.1:2379，则可以通过 curl 或者 etcdctl 命令来进行操作：

```
curl http://127.0.0.1:2379/v2/keys/message -XDELETE
```


另一方面，用户也可以选择使用 `etcdctl` 来获取键值，命令如下：

```
etcdctl --endpoints=http://127.0.0.1:2379 rm message
```

5.1.4 设置键值对 TTL

用户可以指定键值对在多少秒后自动过期，假设用户要指定键值对 5 秒后过期，那么可以使用如下的命令：

```
curl http://127.0.0.1:2379/v2/keys/message -XPUT -d ttl=5
```

另一方面，用户也可以选择使用 `etcdctl` 来获取键值，命令如下：

```
etcdctl --endpoints=http://127.0.0.1:2379 set message "Hello etcd" --ttl "5"
```

5.1.5 创建随机键名

某些情况下，用户在一个目录中可能要创建很多键值对，而用户并不关心这些键值对的名称，那么用户可以让 `etcd` 创建随机的键名，这些键名是按序增长的，用户只需要关心键值即可，通过 `POST` 方法来实现，命令如下：

```
curl http://127.0.0.1:2379/v2/keys/message -XPOST -d value="hello etcd"
```

5.1.6 创建目录

除了创建键值对，用户也可以创建目录，假设要创建的目录是 `dir`，那么命令如下：

```
curl http://127.0.0.1:2379/v2/keys/dir -XPUT -d dir=true
```

5.1.7 删除目录

用户将一个目录完整删除掉，这样一来，目录中的内容都会清空，假设要删除的目录是 `dir`，则命令如下：

```
curl 'http://127.0.0.1:2379/v2/keys/dir?dir=true' -XDELETE
```

5.1.8 查看目录

用户可以查看目录下包含的内容，这些内容可以是键名，也可以是其它子目录，假设要查看的是根目录，那么命令如下：

```
curl http://127.0.0.1:2379/v2/keys/
```

5.1.9 设置目录 TTL

除了为键值对设置生存周期，用户也可以针对目录设置生存周期 TTL，假设用户要为目录 dir 设置 TTL 为 30 秒，命令如下：

```
curl http://127.0.0.1:2379/v2/keys/dir -XPUT -d ttl=30 -d dir=true
```

5.1.10 比较并交换

用户在设置键值对之前，可以加入判断条件，只有当判断条件成立的时候，才进行键值对的设置，如果判断条件失败，则不设置键值对，这样的操作对于实现分布式锁是非常基本的。

比较的条件可以放在 http 请求的 query 字符串中传递给 etcd 服务器，假设用户希望创建一个键值对，但是希望这个键值对在创建的时候是不存在的，如果存在就不用创建了，键名为 "message"，则命令如下：

```
curl http://127.0.0.1:2379/v2/keys/foo?prevExist=false -XPUT -d value=three
```

5.1.11 比较并删除

类似于比较并交换，在删除一个键值对之前，也需要进行条件判断，只有条件成立，才能进行后续的删除操作，假设用户要删除的键名是 "message"，并且希望删除该键时，其键值是 "hello etcd"，如果键值不匹配，则放弃删除，则命令如下：

```
curl http://127.0.0.1:2379/v2/keys/message?prevValue="hello etcd" -XDELETE
```

5.1.12 响应错误代码

本章节描述了键空间“/v2/keys”中使用到的错误代码。可以导入项目“github.com/coreos/etcd/error”来使用。

错误代码被分成四类：

. 命令行相关的错误

名称	代码	解释
EcodeKeyNotFound	100	“键不存在”
EcodeTestFailed	101	“比较异常”
EcodeNotFile	102	“不是一个文件”
EcodeNotDir	104	“不是一个目录”
EcodeNodeExist	105	“键已经存在”
EcodeRootROOnly	107	“根只读”
EcodeDirNotEmpty	108	“目录非空”

. POST 表单的相关错误

名称	代码	解释
EcodePrevValueRequired	201	“需要在 POST 表单中指定前值”
EcodeTTLNaN	202	“POST 表单中指定的 TTL 不是一个数字”
EcodeIndexNaN	203	“POST 表单中指定的索引不是一个数字”
EcodeInvalidField	209	“非法字段”
EcodeInvalidForm	210	“非法 POST 表单”

. raft 相关的错误

名称	代码	解释
EcodeRaftInternal	300	“Raft 内部错误”
EcodeLeaderElect	301	“领导者选举过程中发生的错误”

. etcd 相关的错误

名称	代码	解释
EcodeWatcherCleared	400	“由于 etcd 故障恢复导致监视请求被清空”
EcodeEventIndexCleared	401	“被请求索引中的事件过时而且被清空了”

5.2 获取分析数据

除了常用的键值对操作，etcd 还通过 API 的方式，将集群自身的统计信息暴露给客户端，这些用于分析的数据包括延时，带宽，运行时间等，用户可以通过这些数据来了解集群的运行状态。

分析数据包含领导者统计数据，成员自身统计数据以及存储统计数据。

5.2.1 领导者数据

领导者负责管理整个 etcd 集群，它也掌握着整个集群的状态信息，比如到每个成员的网络延时，以及成功失败的 raft 请求数量，用户可以通过如下命令来获取这些信息：

```
curl http://127.0.0.1:2379/v2/stats/leader
```

响应 demo 如下：

```
{
  "followers": {
    "6e3bd23ae5f1eae0": {
      "counts": {
```

```
        "fail": 0,

        "success": 745

    },

    "latency": {

        "average": 0.017039507382550306,

        "current": 0.000138,

        "maximum": 1.007649,

        "minimum": 0,

        "standardDeviation": 0.05289178277920594

    }

},

"a8266ecf031671f3": {

    "counts": {

        "fail": 0,

        "success": 735

    },

    "latency": {

        "average": 0.012124141496598642,

        "current": 0.000559,

        "maximum": 0.791547,

        "minimum": 0,

        "standardDeviation": 0.04187900156583733

    }

}

},

"leader": "924e2e83e93f2560"

}
```

5.2.2 成员自身数据

Etcd 集群中的每个成员都会跟踪自己的状态，这些状态信息包括自身的 ID 号码，领导者是谁，接受到的 raft 请求数量等等，用户可以通过如下命令来获取这些信息：

```
curl http://127.0.0.1:2379/v2/stats/self
```

响应 demo 如下：

```
{
  "id": "eca0338f4ea31566",
  "leaderInfo": {
    "leader": "8a69d5f6b7814500",
    "startTime": "2014-10-24T13:15:51.186620747-07:00",
    "uptime": "10m59.322358947s"
  },
  "name": "node3",
  "recvAppendRequestCnt": 5944,
  "recvBandwidthRate": 570.6254930219969,
  "recvPkgRate": 9.00892789741075,
  "sendAppendRequestCnt": 0,
  "startTime": "2014-10-24T13:15:50.072007085-07:00",
  "state": "StateFollower"
}
```

5.2.3 存储数据

对 etcd 集群的键值对增删改查等操作都会被记录在存储数据中，这些统计数据在内存中维护，一旦成员停服，那么数据将丢失，用户可以通过下面的命令来获取存储数据：

```
curl http://127.0.0.1:2379/v2/stats/store
```

响应的 demo 如下：

```
{
```

```
"compareAndSwapFail": 0,
"compareAndSwapSuccess": 0,
"createFail": 0,
"createSuccess": 2,
"deleteFail": 0,
"deleteSuccess": 0,
"expireCount": 0,
"getsFail": 4,
"getsSuccess": 75,
"setsFail": 2,
"setsSuccess": 4,
"updateFail": 0,
"updateSuccess": 0,
"watchers": 0
}
```

5.3 监控键值对

对键值对的变化监控是 etcd 服务发现的基础，通过监控键值对，用户可以在键值对发生改变时立即做出响应，键值对的监控是基于 http 的长轮训机制来实现的。

通过设置属性 recursive=true，用户可以同时监控子键值对的变化。

下面通过两个终端的操作来演示如何监控键值对的变化：

打开终端 A，输入如下命令来监控键值对 foo：

```
curl http://127.0.0.1:2379/v2/keys/foo?wait=true
```

打开终端 B，将键 foo 的值设置为 bar：

```
curl http://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar
```

可以在终端 A 中看到类似如下的输出：

```
{
```

```
{
  "action": "set",
  "node": {
    "createdIndex": 7,
    "key": "/foo",
    "modifiedIndex": 7,
    "value": "bar"
  },
  "prevNode": {
    "createdIndex": 6,
    "key": "/foo",
    "modifiedIndex": 6,
    "value": "bar"
  }
}
```

上面是最基本的键值对监控的功能，通过指定索引等条件，用户可以监控到以前的键值对变化，举例来说，上面的例子中，本文已经将键 foo 的值设置成了 bar，生成的最新索引是 7，那么用户可以再开一个终端 C，在终端 C 中指定要监控索引为 7 的键值对变化情况：

```
curl 'http://127.0.0.1:2379/v2/keys/foo?wait=true&waitIndex=7'
```

此时终端会立即输出和上面同样的响应：

```
{
  "action": "set",
  "node": {
    "createdIndex": 7,
    "key": "/foo",
    "modifiedIndex": 7,
    "value": "bar"
  },
  "prevNode": {
    "createdIndex": 6,
```



```
{
  "key": "/foo",
  "modifiedIndex": 6,
  "value": "bar"
}
```

如果设置属性 waitIndex=1，那么该键值对从 1 到当前索引 7 的所有操作都会返回，以此类推。

在 etcd 中，只会维护最近 1000 条事件的响应，之前的响应都会丢失，所以用户在进行键值对监控的时候，一定要正确指定 waitIndex 属性，否则可能会出现错误，举例来说，假设键值对已经更改了 2000 次，现在用户通过如下命令监控第 8 次及以后的变化：

```
curl 'http://127.0.0.1:2379/v2/keys/foo?wait=true&waitIndex=8'
```

响应会返回错误如下：

```
{"errorCode":401,"message":"The event in requested index is outdated and cleared","cause":"the requested history has been cleared [1008/8]","index":2007}
```

为了避免这样的错误，正确的设置 waitIndex 属性，用户可以先获取该键值对的 X-Etcd-Index 属性值，通过如下命令来获取：

```
curl 'http://127.0.0.1:2379/v2/keys/foo' -vv
```

响应 demo 如下：

```
< HTTP/1.1 200 OK
< Content-Type: application/json
< X-Etcd-Cluster-Id: 7e27652122e8b2ae
< X-Etcd-Index: 2007
< X-Raft-Index: 2615
< X-Raft-Term: 2
< Date: Mon, 05 Jan 2015 18:54:43 GMT
< Transfer-Encoding: chunked
<
{"action":"get","node":{"key":"/foo","value":"bar","modifiedIndex":7,"createdIndex":7}}
```

上述响应意味着，当前对该键值对的更改已经达到了 2007 次，那么用户只能监控第 1007

次及以后的更改了。

5.4 ETCD 集群调优

在某些部署环境和应用场景中，默认的 etcd 参数可能不是最合适的，这时用户可以针对性的进行参数调整，优化 etcd 的性能和稳定性。

调优分成两部分，分别是时间参数调优以及快照参数调优。

5.4.1 时间参数调优

本地网络的延时一般比较低，此时 etcd 默认的设置应该绰绰有余。但是，如果要跨越多个数据中心或者跨越高延时网络来使用 etcd，那么用户需要相应调整心跳间隔以及选举超时等设置。

网络并不是导致延时的唯一原因。每一个请求和响应都可能会被领导者和服从者缓慢的磁盘 IO 所拖累。以上超时的时长包含了从请求生成到成功获取对端响应的全部时间。

如果领导者挂了或者离线了，底层的分布式一致性协议能够保证领导权的平稳移交，而一致性协议依赖于两个不同的时间参数。第一个参数叫做心跳间隔。这个时间是领导者通知其它成员其仍然存活的时间间隔。从工程上来说，这个时间的大小最好设置成成员之间的往返延时。默认情况下，etcd 将心跳间隔设置成 100 毫秒。

第二个参数是选举超时。如果服从者超过这个时长还没有听到领导者的心跳，那么它将尝试成为领导者。默认情况下，etcd 将选举超时设置成 1000 毫秒。

对这些数字的调整往往需要进行折中。建议将心跳间隔的大小设置成成员之间的最大往返延时，一般是往返延时的 0.5 到 1.5 倍。如果心跳间隔太小，etcd 会发送很多不必要的网络信息，这将增加 CPU 和网络的负载。另一方面，太高的心跳间隔将推高选举超时。过高的选举超时将不能即使发现领导者的失败。测量往返延时的最简单办法就是使用工具 ping。

选举超时的设置应该同时参考心跳间隔和成员之间的平均往返延时。选举超时至少应该是往返延时的 10 倍，这样才能对抗网络的抖动。举个例子，如果成员之间的往返延时是 10 毫秒，那么选举超时至少要设置成 100 毫秒。

选举超时至少也应该是心跳间隔的 5 到 10 倍，以对抗领导者响应的抖动。如果心跳间隔是 50 毫秒，那么选举超时应该位于 250 毫秒到 500 毫秒之间。

选举超时的上线是 50 秒，这个值只有在部署全球分布的 etcd 集群时使用。美国州际之间的往返延时大概是 130 毫秒，而且美国到日本的往返延时大概是 350 到 400 毫秒。如果网络抖动很厉害，或者经常发生大的延时与丢包，那么一次成功的数据包发送可能需要经过多次重传。所以 5 秒钟是一个合理的全球往返延时。由于选举超时应该比广播延时大一个数量级，对于广播延时等于 5 秒的全球分布集群而言，50 秒的选举超时是合理的。

对于集群的所有成员而言，心跳间隔和选举超时的大小分别应该设置成一样。成员之间的配置有差异会破坏集群的稳定性。

用户可以通过命令行来改写默认的值。

```
# Command line arguments:

$ etcd -heartbeat-interval=100 -election-timeout=500

# Environment variables:

$ ETCD_HEARTBEAT_INTERVAL=100 ETCD_ELECTION_TIMEOUT=500 etcd
```

这些值的单位是毫秒。

5.4.2 快照参数调优

Etcd 将所有的键操作都写入日志。该日志会持续线性增长。全量日志对于负载较轻的集群而言没什么问题，但是对于高负载的集群而言，日志会变得很大。

为了避免生成巨大的日志，etcd 会定时生成快照。通过保存当前的系统状态以及删除旧的日志，这快照使得 etcd 能够对日志进行压缩。

鉴于快照的生成是非常昂贵的，只有当更改次数达到一定数量之后才会打快照。默认情况下，每隔 10000 次键操作，会生成一次快照。如果 etcd 的内存使用率和磁盘使用率过高，用户可以降低打快照的阈值，设置方法如下：

```
# Command line arguments:

$ etcd -snapshot-count=5000
```

```
# Environment variables:

$ ETCD_SNAPSHOT_COUNT=5000 etcd
```

5.5 安全策略

在 etcd 的实际部署中，数据的安全是非常重要的，etcd 针对安全问题提供如下两个方面的策略，包括安全传输，以及访问控制。

5.5.1 安全传输

etcd 支持对传输数据的 SSL/TLS 加密，既可以加密客户端与 etcd 服务器的传输连接，同时也可以加密 etcd 成员之间的传输连接。

为了使用安全传输的功能，用户首先需要有一个 CA 证书，并且使用该证书来签发客户端和服务器的证书以及私钥，具体的方法请参考文章[如何构造自签名秘钥对](#)。

在创建好自签名秘钥对之后，用户可以通过配置 etcd 的参数来使用这些证书和秘钥，具体的配置参数涉及到如下几项：

- cert-file=<path>: 当前 etcd 服务器的证书路径，用于向客户端表明服务器的身份
- key-file=<path>: 当前 etcd 服务器的私钥路径，用于加解密与客户端的通信数据
- client-cert-auth: 是否对客户端进行认证
- trusted-ca-file=<path>: 如果对客户端进行认证，需要指定客户端 CA 的证书路径
- peer-cert-file=<path>: 当前 etcd 服务器的证书路径，用于向集群其它成员表明服务器的身份
- peer-key-file=<path>: 前 etcd 服务器的私钥路径，用于加解密与其它成员的通信数据
- peer-client-cert-auth: 是否对其它成员进行身份认证
- peer-trusted-ca-file=<path>: 如果对其它成员进行认证，那么需要指定其它成员的 CA 证书路径

下面通过三个具体的例子来描述，如何使用服务器认证，客户端认证以及成员节点认证。

【案例一：服务器认证】

启动 etcd 成员时，需要指定证书和私钥的文件位置：

```
$ etcd -name infra0 -data-dir infra0 \
  -cert-file=/path/to/server.crt -key-file=/path/to/server.key \
  -advertise-client-urls=https://127.0.0.1:2379 -listen-client-urls=https://127.0.0.1:2379
```

在客户端请求 etcd 服务器的时候，需要指定 CA 证书的位置：

```
$ curl --cacert /path/to/ca.crt https://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar -v
```

为了避免每次请求都指定 CA 证书，用户也可以将 CA 证书存放到根证书目录下，对于 linux 操作系统，该目录为/etc/ssl/certs，这样一来，用户在请求 etcd 集群时就无须指定 CA 证书了：

```
$ curl https://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar -v
```

【案例二：客户端认证】

在启动 etcd 成员时，需要指定对客户端进行认证，同时给出客户端的 CA 证书路径：

```
$ etcd -name infra0 -data-dir infra0 \
  -client-cert-auth -trusted-ca-file=/path/to/ca.crt -cert-file=/path/to/server.crt \
  -key-file=/path/to/server.key \
  -advertise-client-urls https://127.0.0.1:2379 -listen-client-urls https://127.0.0.1:2379
```

此时可以在客户端请求 etcd 服务器，请求时提供客户端自己的证书和私钥：

```
$ curl --cacert /path/to/ca.crt --cert /path/to/client.crt --key /path/to/client.key \
  -L https://127.0.0.1:2379/v2/keys/foo -XPUT -d value=bar -v
```

成功的响应 demo 如下：

```
{
  "action": "set",
  "node": {
    "createdIndex": 12,
    "key": "/foo",
    "modifiedIndex": 12,
```

```
"value": "bar"

}

}
```

【案例三：成员节点认证】

在启动各个集群成员时，需要在每个成员中指定成员节点认证，并且指定对端节点的 CA 证书路径，同时还需要加载自己的证书和私钥：

设置环境变量：

```
DISCOVERY_URL=... # from https://discovery.etcd.io/new
```

成员 1

```
$ etcd -name infra1 -data-dir infra1 \
    -peer-client-cert-auth                    -peer-trusted-ca-file=/path/to/ca.crt
-peer-cert-file=/path/to/member1.crt -peer-key-file=/path/to/member1.key \
    -initial-advertise-peer-urls=https://10.0.1.10:2380 -listen-peer-urls=https://10.0.1.10:2380 \
    -discovery ${DISCOVERY_URL}
```

成员 2

```
$ etcd -name infra2 -data-dir infra2 \
    -peer-client-cert-auth                    -peer-trusted-ca-file=/path/to/ca.crt
-peer-cert-file=/path/to/member2.crt -peer-key-file=/path/to/member2.key \
    -initial-advertise-peer-urls=https://10.0.1.11:2380 -listen-peer-urls=https://10.0.1.11:2380 \
    -discovery ${DISCOVERY_URL}
```

如果客户端和 etcd 集群之间有 etcd 代理进行请求转发，那么代理需要同时设置服务器认证以及成员节点认证，必要的情况下还需要配置客户端认证。

5.5.2 访问控制

Etcd 中存储了三种类型的资源：

1. 权限资源：包括用户以及角色，这会在本节接下来的内容中讲述。
2. 键值对资源：前面章节所涉及到的键值对资源。
3. 配置资源：包含安全设置，集群设置等等，在前面章节的高级配置选项中有涉及到。

下面开始描述 etcd 中的权限资源，以及与之配套的身份认证机制。权限资源中涉及到三个基本的概念：用户，角色以及读写权限。

用户就是常见的用户名，这是一个身份标识，etcd 中默认有一个 root 用户，如果要在 etcd 中开启身份认证的功能，首先需要创建一个 root 用户，这是一个管理员账号。

一个角色包含若干的读写权限，默认情况下，etcd 中包含了一个内置的 root 角色，该角色拥有所有三种资源的读写权限，同时还内置了一个 guest 角色，该角色拥有键值对资源的读写权限。

读写权限顾名思义，有两种类型，读与写，主要针对键值对或者目录的读写控制。

在明确了上述概念之后，下面开始描述如何使用 etcd 暴露给用户的 API 来进行身份认证的配置。

【获取当前的认证状态】

通过 http 的 GET 方法来获取当前的认证状态，确认认证是开启还是关闭：

```
GET /v2/auth/enable
```

无须设置 http 请求头部

可能的响应状态码：

```
200 OK
```

实体数据：

```
{
  "enabled": true
}
```

【使能身份认证】

通过 http 的 PUT 方法来使能身份认证：

```
PUT /v2/auth/enable
```

无须设置 http 请求头部。

可能的响应状态码：

200 OK

400 Bad Request (if root user has not been created)

409 Conflict (already enabled)

200 状态码实体数据: (空)

【关闭身份认证】

使用 http 的 DELETE 方法来关闭身份认证:

DELETE /v2/auth/enable

需要设置 http 请求头部:

Authorization: Basic <RootAuthString>

可能的响应状态码:

200 OK

401 Unauthorized (if not a root user)

409 Conflict (already disabled)

200 状态码实体数据: (空)

【获取用户列表】

通过 http 的 GET 或者 HEAD 方法来获取用户列表:

GET/HEAD /v2/auth/users

需要设置 http 请求头部:

Authorization: Basic <RootAuthString>

可能的响应状态码:

200 OK

401 Unauthorized

200 状态码响应头部:

Content-type: application/json

200 状态码响应实体数据:

```
{  
  "users": [  

```



```
{
  "user": "alice",
  "roles": [
    {
      "role": "root",
      "permissions": {
        "kv": {
          "read": ["/*"],
          "write": ["/*"]
        }
      }
    }
  ]
},
{
  "user": "bob",
  "roles": [
    {
      "role": "guest",
      "permissions": {
        "kv": {
          "read": ["/*"],
          "write": ["/*"]
        }
      }
    }
  ]
}
]
```

```
}
```

【获取单个用户的信息】

通过 http 的 GET 或者 HEAD 方法来获取单个用户的信息：

```
GET/HEAD /v2/auth/users/alice
```

需要设置 http 请求头部：

```
Authorization: Basic <RootAuthString>
```

可能的响应状态码：

```
200 OK
```

```
401 Unauthorized
```

```
404 Not Found
```

200 状态码响应头部：

```
Content-type: application/json
```

200 状态码响应实体数据：

```
{
  "user" : "alice",
  "roles" : [
    {
      "role": "fleet",
      "permissions" : {
        "kv" : {
          "read": [ "/fleet/" ],
          "write": [ "/fleet/" ]
        }
      }
    },
    {
      "role": "etcd",
      "permissions" : {
```

```

    "kv": {
      "read": [ "/"* ],
      "write": [ "/"* ]
    }
  }
}
]
}

```

【创建或者更新用户信息】

在创建用户时，必须提供的信息包括用户名和密码，绑定的角色信息是可选的。可以通过 http 的 PUT 方法来创建或者更新用户：

PUT /v2/auth/users/charlie

需要设置 http 请求头部：

Authorization: Basic <RootAuthString>

同时需要 http 的实体数据部分提供 JSON 格式的用户信息，信息的格式如下面的用户格式所示：

```

{
  "user": "userName",
  "password": "password",
  "roles": [
    "role1",
    "role2"
  ],
  "grant": [],
  "revoke": []
}

```

可能的响应状态码：

200 OK

201 Created

400 Bad Request

401 Unauthorized

404 Not Found (update non-existent users)

409 Conflict (when granting duplicated roles or revoking non-existent roles)

200 状态码响应头部:

Content-type: application/json

200 状态码响应实体数据:

如上面的用户格式所示

【删除用户】

通过 http 的 REMOVE 方法来删除用户信息:

DELETE /v2/auth/users/charlie

需要设置 http 请求头部:

Authorization: Basic <RootAuthString>

可能的响应状态码:

200 OK

401 Unauthorized

403 Forbidden (remove root user when auth is enabled)

404 Not Found

200 状态码响应实体数据: (空)

【获取某个角色信息】

使用 http 的 GET 或者 HEAD 方法来获取某个角色的信息:

GET/HEAD /v2/auth/roles/fleet

需要设置 http 请求头部:

Authorization: Basic <RootAuthString>

可能的响应状态码:

200 OK

401 Unauthorized

404 Not Found

200 状态码响应头部:

Content-type: application/json

200 状态码响应实体数据:

```
{
  "role": "fleet",
  "permissions": {
    "kv": {
      "read": [ "/fleet/" ],
      "write": [ "/fleet/" ]
    }
  }
}
```

【获取所有角色的列表】

使用 http 的 GET 或者 HEAD 方法来获取角色的列表:

GET/HEAD /v2/auth/roles

需要设置 http 请求头部:

Authorization: Basic <RootAuthString>

可能的响应状态码:

200 OK

401 Unauthorized

200 状态码响应头部:

Content-type: application/json

200 状态码响应实体数据:

```
{
  "roles": [
    {
```

```
    "role": "fleet",  
    "permissions": {  
      "kv": {  
        "read": ["/fleet/"],  
        "write": ["/fleet/"]  
      }  
    }  
  },  
  {  
    "role": "etcd",  
    "permissions": {  
      "kv": {  
        "read": ["/*"],  
        "write": ["/*"]  
      }  
    }  
  },  
  {  
    "role": "quay",  
    "permissions": {  
      "kv": {  
        "read": ["/*"],  
        "write": ["/*"]  
      }  
    }  
  }  
]  
}
```

【创建或者更新一个角色】

使用 http 的 PUT 方法来创建或者更新一个角色：

PUT /v2/auth/roles/rkt

需要设置 http 请求头部：

Authorization: Basic <RootAuthString>

PUT 请求的实体数据格式需要提供角色的信息，格式如下：

```
{ "role" : "fleet", "permissions" : { "kv" : { "read" : [ "/fleet/" ], "write" : [ "/fleet/" ] } }, "grant" : { "kv" : { "..."}, "revoke" : { "kv" : { "..."} } }
```

可能的响应状态码：

200 OK

201 Created

400 Bad Request

401 Unauthorized

404 Not Found (update non-existent roles)

409 Conflict (when granting duplicated permission or revoking non-existent permission)

200 状态码响应实体数据：

同上述角色的格式

【删除一个角色】

通过 http 的 REMOVE 方法来删除角色信息：

DELETE /v2/auth/roles/rkt

需要设置 http 请求头部：

Authorization: Basic <RootAuthString>

可能的响应状态码：

200 OK

401 Unauthorized

403 Forbidden (remove root)

404 Not Found

200 状态码响应头部：（空）

200 状态码响应实体数据：（空）

在了解了身份认证的配置方法之后，下面通过一个具体的例子来描述完整的流程。在这个例子中，有两个租户要使用 etcd 的服务。

首先创建 root 用户：

PUT /v2/auth/users/root

Put Body:

```
{"user": "root", "password": "betterRootPW!"}
```

使能身份认证：

PUT /v2/auth/enable

修改 guest 角色，撤销其写入权限：

PUT /v2/auth/roles/guest

http 头部信息：

Authorization: Basic <root:betterRootPW!>

表单数据：

```
{
  "role": "guest",
  "revoke": {
    "kv": {
      "write": [
        "/"
      ]
    }
  }
}
```

为两个租户分别创建角色：

PUT /v2/auth/roles/rkt

http 头部：

Authorization: Basic <root:betterRootPW!>

实体数据部分:

```
{
  "role": "rkt",
  "permissions": {
    "kv": {
      "read": [
        "/rkt/"
      ],
      "write": [
        "/rkt/"
      ]
    }
  }
}
```

PUT /v2/auth/roles/fleet

http 请求头部:

Authorization: Basic <root:betterRootPW!>

http 表单数据:

```
{
  "role": "fleet"
}
```

给角色授权:

PUT /v2/auth/roles/fleet

http 请求头部:

Authorization: Basic <root:betterRootPW!>

表单数据:

```
{
```

```
"role": "fleet",  
  
"grant": {  
  "kv": {  
    "read": [  
      "/rkt/fleet",  
      "/fleet/*"  
    ]  
  }  
}
```

创建用户:

```
PUT /v2/auth/users/fleetuser
```

http 请求头部:

```
Authorization: Basic <root:betterRootPW!>
```

表单数据:

```
{"user": "fleetuser", "grant": ["fleet"]}
```

给用户绑定角色:

```
PUT /v2/auth/users/fleetuser
```

请求头部:

```
Authorization: Basic <root:betterRootPW!>
```

表单数据:

```
{"user": "fleetuser", "grant": ["fleet"]}
```

到此, 已经将 fleetuser 用户加入到 etcd 的访问控制当中, 可以通过该用户的账号来操作 etcd 中的数据:

```
PUT /v2/keys/rkt/RktData
```

http 请求头部:

```
Authorization: Basic <rktuser:rktpw>
```

表单数据:

```
value=launch
```

如果操作合法目录意外的键值对，将会返回 401 的 http 响应状态码。

5.5 本章小结

本章首先描述了基本的 etcd 键值对操作接口，并对 etcd 集群的统计信息获取接口进行了分析，接着重点介绍了 etcd 中的键值对监控功能，以及 etcd 的集群调优方案，最后给出了 etcd 中安全与认证方案。

第六章 运行时配置重载

etcd 带有运行时配置重载的功能，通过该功能，用户可以对一个正在运行当中的 etcd 集群的配置进行更改，配置更改后立即生效，无须重启 etcd 服务，运行时配置重载的功能可以应用到每个成员节点上，也就是可以对每个成员的配置信息进行更改，本章将通过 systemd 服务和 etcdctl 命令的使用，来阐述基于 CoreOS 的 etcd 集群的配置重载或者故障恢复。

6.1 调整集群大小

在 CoreOS 节点上使用云配置方式来配置一个 etcd 成员时，中间会编译生成一个特殊的 drop-in 单元文件，该文件的名称为 /run/systemd/system/etcd2.service.d/20-cloudinit.conf。举个例子，如下的云配置文件：

```
#cloud-config

coreos:
  etcd2:
    advertise-client-urls: http://<PEER_ADDRESS>:2379
    initial-advertise-peer-urls: http://<PEER_ADDRESS>:2380
    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
    listen-peer-urls: http://0.0.0.0:2380
    discovery: https://discovery.etcd.io/<token>
```

会编译生成如下的 drop-in 文件：

```
[Service]
Environment="ETCD_ADVERTISE_CLIENT_URLS=http://<PEER_ADDRESS>:2379"
Environment="ETCD_DISCOVERY=https://discovery.etcd.io/<token>"
Environment="ETCD_INITIAL_ADVERTISE_PEER_URLS=http://<PEER_ADDRESS>:2380"
```

```
Environment="ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379,http://0.0.0.0:4001"
```

```
Environment="ETCD_LISTEN_PEER_URLS=http://0.0.0.0:2380"
```

如果 etcd 集群内部使用 TLS 来加密通信，在下面的命令行例子中，请用 https:// 来替换 http://。

假设用户已经创建了一个 5 节点的 etcd 集群，但是并未在发现 url 中指定集群的大小。鉴于以发现服务方式创建的集群默认大小为 3，那么剩下的 2 个节点将被配置成代理。用户希望在不重建集群的情况下，将这 2 个代理升级为集群成员。

可以重新配置一个已经运行的集群。运行命令：

```
etcdctl member add node4 http://10.0.1.4:2380
```

由于后续的步骤将使用到这个命令的输出，建议将输出复制和粘贴到一个文件中。成功添加成员后会输出如下的信息：

```
added member 9bf1b35fc7761a23 to cluster
```

```
ETCD_NAME="node4"
```

```
ETCD_INITIAL_CLUSTER="1dc800dbf6a732d8839bc71d0538bb99=http://10.0.1.1:2380,f961e5cb1b0cb8810ea6a6b7a7c8b5cf=http://10.0.1.2:2380,8982fae69ad09c623601b68c83818921=http://10.0.1.3:2380,node4=http://10.0.1.4:2380"
```

```
ETCD_INITIAL_CLUSTER_STATE=existing
```

由于 20-cloudinit.conf 文件中的 ETCD_DISCOVERY 环境变量与上述的 ETCD_INITIAL_CLUSTER 配置有冲突，所以第一步需要使用一个新的文件 99-restore.conf 来替换原先的 20-cloudinit.conf。文件 99-restore.conf 中包含了一个空值的 Environment="ETCD_DISCOVERY=" 字符串。

完整的例子如下所示。在名为 node4 的 CentOS 主机上，创建一个临时的 systemd drop-in 文件 /run/systemd/system/etcd2.service.d/99-restore.conf，文件内容如下所示，需要使用上述 etcd member add 命令的输出信息来填充该文件：

```
[Service]
```

```
# remove previously created proxy directory
```

```
ExecStartPre=/usr/bin/rm -rf /var/lib/etcd2/proxy
```

```
# NOTE: use this option if you would like to re-add broken etcd member into cluster
```

```
# Don't forget to make a backup before

#ExecStartPre=/usr/bin/rm -rf /var/lib/etcd2/member /var/lib/etcd2/proxy

# here we clean previously defined ETCD_DISCOVERY environment variable, we don't need it as
we've already bootstrapped etcd cluster and ETCD_DISCOVERY conflicts with
ETCD_INITIAL_CLUSTER environment variable
Environment="ETCD_DISCOVERY="
Environment="ETCD_NAME=node4"

# We use ETCD_INITIAL_CLUSTER variable value from previous step ("etcdctl member add"
output)
Environment="ETCD_INITIAL_CLUSTER=node1=http://10.0.1.1:2380,node2=http://10.0.1.2:2380,
node3=http://10.0.1.3:2380,node4=http://10.0.1.4:2380"
Environment="ETCD_INITIAL_CLUSTER_STATE=existing"
```

运行 bash 命令：

```
sudo systemctl daemon-reload
```

来解析刚才生成的单元文件。接着运行：

```
sudo journalctl _PID=1 -e -u etcd2
```

通过检查服务的日志信息来检查这个新的 drop-in 文件是否正确。如果配置正确，运行命令：

```
sudo systemctl restart etcd2
```

来激活修改。用户此时可以看到，之前的代理节点变成了集群成员：

```
etcdserver: start member 9bf1b35fc7761a23 in cluster 36cce781cb4f1292
```

一旦新的成员运行起来了，并且 etcdctl cluster-health 命令也显示集群状态健康，将临时的 drop-in 文件删除掉，并且重新解析服务：

```
sudo rm /run/systemd/system/etcd2.service.d/99-restore.conf && sudo systemctl
daemon-reload
```

6.2 替换失败成员

本小节将演示如何恢复一个失败的 etcd 成员。需要明确的是，仅仅通过一个发现 url 是无法恢复 etcd 集群的。发现 url 仅仅只在集群的启动阶段使用，集群启动完毕也就作废了。

在下面的例子中，有一个集群包含 3 个节点，虽然其中 1 个节点挂掉了，但是由于仍然有足够的合法成员，所以集群是正常运行的。一个 etcd 成员挂掉的原因可能有如下几个：磁盘空间已满，失败的重启，或者是底层的系统出现了问题。这个例子假设用户的集群是使用云配置和发现服务的方式启动的，其中云配置的内容如下：

```
#cloud-config

coreos:

  etcd2:

    advertise-client-urls: http://<PEER_ADDRESS>:2379

    initial-advertise-peer-urls: http://<PEER_ADDRESS>:2380

    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001

    listen-peer-urls: http://0.0.0.0:2380

    discovery: https://discovery.etcd.io/<token>
```

如果 etcd 集群内部使用 TLS 来加密数据，请在下面的例子中使用 https://来替换 http://。

假设用户的 etcd 集群中，失败的成员 IP 是 10.0.1.2:

```
$ etcdctl cluster-health
```

输出的响应如下：

```
member fe2f75dd51fa5ff is healthy: got healthy result from http://10.0.1.1:2379
failed to check the health of member 1609b5a3a078c227 on http://10.0.1.2:2379: Get
http://10.0.1.2:2379/health: dial tcp 10.0.1.2:2379: connection refused
member 1609b5a3a078c227 is unreachable: [http://10.0.1.2:2379] are all unreachable
member 60e8a32b09dc91f1 is healthy: got healthy result from http://10.0.1.3:2379
cluster is healthy
```

在一个正常工作的节点上运行 `etcdctl` 命令，或者在一个远程客户端上，使用 `ETCDCTL_ENDPOINT` 环境变量或者 `etcdctl` 的命令行选项，来向远端的健康成员发起请求。

将失败的成员 10.0.1.2 从集群中移除掉。`Etcdctl` 的删除子命令将会通知集群的其它成员，管理员已经认定该成员已经挂掉，而且已经不能正常通信了：

```
$ etcdctl member remove 1609b5a3a078c227
```

输出的响应如下：

```
Removed member 1609b5a3a078c227 from cluster
```

接着在失败的 10.0.1.2 节点上，杀死 `etcd` 进程：

```
$ sudo systemctl stop etcd2
```

清空 `/var/lib/etcd2` 目录：

```
$ sudo rm -rf /var/lib/etcd2/*
```

检查 `/var/lib/etcd2/` 目录是存在的，而且是空的。如果用户不小心删除了该目录，那么可以重新创建该目录，并赋予其适当的目录权限，命令如下：

```
$ sudo systemd-tmpfiles --create /usr/lib64/tmpfiles.d/etcd2.conf
```

接下来，重新初始化该失败的成员。需要注意 10.0.1.2 只是一个示范 IP。用户应该使用失败节点的真实 IP。

```
$ etcdctl member add node2 http://10.0.1.2:2380
```

输出的响应如下：

```
Added member named node2 with ID 4fb77509779cac99 to cluster
```

```
ETCD_NAME="node2"
```

```
ETCD_INITIAL_CLUSTER="52d2c433e31d54526cf3aa660304e8f1=http://10.0.1.1:2380,node2=http://10.0.1.2:2380,2cb7bb694606e5face87ee7a97041758=http://10.0.1.3:2380"
```

```
ETCD_INITIAL_CLUSTER_STATE="existing"
```

添加新的节点之后，创建一个新的 `systemd drop-in` 文件 `/run/systemd/system/etcd2.service.d/99-restore.conf`，使用之前 `etcdctl member add` 命令的输出信息来替换该文件中的节点信息。

```
[Service]
```



```
# here we clean previously defined ETCD_DISCOVERY environment variable, we don't need it as
we've already bootstrapped etcd cluster and ETCD_DISCOVERY conflicts with
ETCD_INITIAL_CLUSTER environment variable
Environment="ETCD_DISCOVERY="
Environment="ETCD_NAME=node2"
# We use ETCD_INITIAL_CLUSTER variable value from previous step ("etcdctl member add"
output)
Environment="ETCD_INITIAL_CLUSTER=52d2c433e31d54526cf3aa660304e8f1=http://10.0.1.1:2
380,node2=http://10.0.1.2:2380,2cb7bb694606e5face87ee7a97041758=http://10.0.1.3:2380"
Environment="ETCD_INITIAL_CLUSTER_STATE=existing"
```

备注：确保 ETCD_INITIAL_CLUSTER=条目后面不要残留多余的双引号。

解析新的 drop-in 文件：

```
$ sudo systemctl daemon-reload
```

检查新的 drop-in 文件是否解析正常：

```
sudo journalctl _PID=1 -e -u etcd2
```

最后，如果一切正常，启动 etcd2 服务：

```
$ sudo systemctl start etcd2
```

检查集群健康状态：

```
$ etcdctl cluster-health
```

如果集群的状态是健康的，说明 etcd 已经成功将集群配置信息写入到 /var/lib/etcd2 目录中去。

现在可以放心的删除临时的 /run/systemd/system/etcd2.service.d/99-restore.conf 文件了。

6.3 故障恢复

如果一个集群完全崩溃了，而且成员数量小于合法数量，那么就必须从头配置所有的成员。整个流程包含两步：

. 使用初始的数据目录来初始化一个单成员的 etcd 集群

. 通过添加新的成员来调整 etcd 集群的大小，具体的步骤请参照上一小节，更改 etcd 集群的大小。

本小节对官方的 etcd 故障恢复指南进行了些许变通，出于方便，使用的是 systemd drop-in 文件。

假设一个 3 节点的集群中，所有成员都挂掉了。首先，在所有成员上停止 etcd2 服务：

```
$ sudo systemctl stop etcd2
```

如果用户配置 etcd 代理节点，那么代理节点应该能够根据配置项 --proxy-refresh-interval 的值去自动刷新成员列表。

接着，选择一个成员节点，运行下面的命令来备份数据目录：

```
$ sudo etcdctl backup --data-dir /var/lib/etcd2 --backup-dir /var/lib/etcd2_backup
```

既然已经备份了目录，用户可以启动一个单节点的集群。创建一个名为 /run/systemd/system/etcd2.service.d/98-force-new-cluster.conf 的 drop-in 文件，内容如下：

```
[Service]
```

```
Environment="ETCD_FORCE_NEW_CLUSTER=true"
```

接下来运行：

```
sudo systemctl daemon-reload
```

通过查看日志中的错误信息，来检查该 drop-in 文件是否正确：

```
sudo journalctl _PID=1 -e -u etcd2
```

如果一切正常，启动 etcd2 后台进程：

```
sudo systemctl start etcd2。
```

检查集群状态：

```
$ etcdctl member list
```

输出响应如下：

```
e6c2bda2aa1f2dcf: name=1be6686cc2c842db035fdc21f56d1ad0 peerURLs=http://10.0.1.2:2380
clientURLs=http://10.0.1.2:2379
```

检查集群健康：

```
$ etcdctl cluster-health
```

输出响应如下：

```
member e6c2bda2aa1f2dcf is healthy: got healthy result from http://10.0.1.2:2379
cluster is healthy
```

如果输出中没有错误，删除

/run/systemd/system/etcd2.service.d/98-force-new-cluster.conf drop-in 文件，并且重新启动 systemd 服务：

```
sudo systemctl daemon-reload
```

完成这一步之后，没必要再重启 etcd2 服务了。

接下来的步骤和章节更改 etcd 集群大小是相同的：同时删除 /var/lib/etcd2/member 目录和 /var/lib/etcd2/proxy 目录。

6.4 更新成员

1) 更新通告的客户端 url

如果需要替换一个成员的通告客户端 url，可以简单的通过设置标识 `--advertise-client-urls` 或者环境变量 `ETCD_ADVERTISE_CLIENT_URLS` 来重启成员服务。重启的成员会自动发布新的 url。一个错误更新的客户端 url 并不会影响集群的健康状态。

2) 更新通告的对端 url

如果需要更新一个成员的通告对端 url，用户需要通过 `etcdctl` 的成员子命令更新 url，然后重启该成员的服务。同时还需要进行一些额外的操作，因为更新对端 url 也就更改了集群范围的配置，这有可能会影响集群的健康状态。

为了更新对端 url，首先，用户需要找到目标成员的 ID 号码。可以通过 `etcdctl` 命令来列举出所有成员的信息：

```
$ etcdctl member list
```

输出的响应如下：

```
6e3bd23ae5f1eae0: name=node2 peerURLs=http://localhost:23802
clientURLs=http://127.0.0.1:23792
```

```
924e2e83e93f2560: name=node3 peerURLs=http://localhost:23803
clientURLs=http://127.0.0.1:23793
a8266ecf031671f3: name=node1 peerURLs=http://localhost:23801
clientURLs=http://127.0.0.1:23791
```

在这个例子中，本文将更新 ID 号为 a8266ecf031671f3 的成员，并且将它的对端 url 替换成 http://10.0.1.10:2380。

```
$ etcdctl member update a8266ecf031671f3 http://10.0.1.10:2380
```

输出响应如下：

```
Updated member with ID a8266ecf031671f3 in cluster
```

6.5 删除成员

假设本文要删除的成员 ID 号码是 a8266ecf031671f3。可以使用 etcdctl 的 remove 子命令来完成操作：

```
$ etcdctl member remove a8266ecf031671f3
```

输出响应如下：

```
Removed member a8266ecf031671f3 from cluster
```

目标成员会停止自己的服务，并且在日志中打印删除的事件：

```
etcd: this member has been permanently removed from the cluster. Exiting.
```

删除领导者也是安全的，但是集群在领导选举的过程中是无法对外服务的。延续的时长一般是选举超时的时间加上选举的过程耗时。

6.6 添加成员

添加成员包含两步操作：

. 通过成员 API 或者 etcdctl member add 命令向集群中添加新的成员

. 使用新的集群配置来启动新的成员，该配置包含了更新后的成员列表（已经存在的成员+新的成员）。

通过在 etcdctl 命令中指定成员名称以及通告对端 url，本文向集群中添加新的成员：

```
$ etcdctl member add infra3 http://10.0.1.13:2380
```

输出响应如下：

```
added member 9bf1b35fc7761a23 to cluster
```

```
ETCD_NAME="infra3"
```

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380,infra3=http://10.0.1.13:2380"
```

```
ETCD_INITIAL_CLUSTER_STATE=existing
```

Etcdctl 会将新成员的信息通告给集群的其它成员，并且打印出成功启动所需要的环境变量信息。现在可以通过相关的标识来启动新成员的 etcd 服务：

```
$ export ETCD_NAME="infra3"
```

```
$ export
```

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380,infra3=http://10.0.1.13:2380"
```

```
$ export ETCD_INITIAL_CLUSTER_STATE=existing
```

```
$ etcd -listen-client-urls http://10.0.1.13:2379 -advertise-client-urls http://10.0.1.13:2379
```

```
-listen-peer-urls http://10.0.1.13:2380 -initial-advertise-peer-urls http://10.0.1.13:2380
```

```
-data-dir %data_dir%
```

该新成员会加入到集群中，并且立即和集群的其它成员取得联系。

如果要添加多个成员，最好是一次添加一个，并且在添加第二个成员之前，确认第一个成员已经正常启动。如果向一个单成员的集群中添加一个新的成员，在这个新的成员启动之前，这个集群是无法正常工作的，因为集群需要至少两个成员来完成一致性协议。这种状态开始于通过 etcdctl member add 命令将新成员的信息通告给集群，结束于新成员和唯一的节点建立了联系。

1) 添加成员过程中的错误案例

在下面的例子中，本文并未将新添加的成员包含到枚举的节点列表中去。如果这是一个新的集群，新节点必须被添加到初始化集群成员的列表中。

```
$ etcd -name infra3 \
  -initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380 \
  -initial-cluster-state existing
```

输出响应如下：

```
etcdserver: assign ids error: the member count is unequal
exit 1
```

在下面的案例中，本文原先要加入集群的成员 IP 从 10.0.1.13:2380 替换成了 10.0.1.14:2380。

```
$ etcd -name infra4 \
  -initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380,infra4=ht
tp://10.0.1.14:2380 \
  -initial-cluster-state existing
```

输出响应如下：

```
etcdserver: assign ids error: unmatched member while checking PeerURLs
exit 1
```

当用户使用被删除成员的数据目录来启动 etcd 的时候，一旦该进程和集群的任一节点建立了联系，那么它就会自动退出：

```
$ etcd
```

输出结果如下：

```
etcd: this member has been permanently removed from the cluster. Exiting.
exit 1
```

6.7 本章小结

本章介绍了 etcd 运行时配置重载的若干方面，包括调整集群大小，替换失败成员，故障恢复，更新成员，删除成员以及添加成员，通过运行时配置重载的功能，用户可以动态调整集群的规模，并且根据需要进行故障恢复。

第七章 etcd 集群监控

在前面的章节中，本文对 etcd 集群的创建，使用和管理都进行了详尽的介绍，但是对于 etcd 集群的监控尚未涉及，当 etcd 集群正常对外服务的时候，用户需要知道当前集群的内部状态信息，根据集群的监控数据来判断集群的实时状态，为此需要引入 etcd 集群的监控。

7.1 监控指标解读

etcd 服务器将实时的性能参数存储在内存中，并且通过指定的 API 接口暴露给用户，查看性能参数的最简单办法就是使用 curl 来请求 etcd 的 /metrics 目录，该请求的响应结果有着特殊的格式，而这个格式是开源的时间序列化工具 prometheus 能够理解的，关于 prometheus 的介绍会在接下来的章节中进行。

接下来对可以获取到的 etcd 性能参数进行分类介绍，主要包含如下几种类别的性能指标：服务器指标，wal 指标，http 请求指标，快照指标，raft 指标以及代理指标。

备注：一个指标参数名称包含一个 etcd 前缀作为命名空间，后面跟着的是子系统前缀（比如 wal 和 etcdserver）。

7.1.1 服务器指标

名称	描述	类型
file_descriptors_used_total	被占用的总文件句柄数量	Gauge
proposal_durations_seconds	提交的延时分布	Histogram
pending_proposal_total	排队的提议数量	Gauge
proposal_failed_total	失败的提议总数	Counter

如果文件句柄（file_descriptors_used_total）的使用率过高，逼近了进程的文件句

柄上线，那么就意味着文件句柄可能要被耗尽了。这会导致 etcd 无法创建新的 WAL 文件，最后发生崩溃。

提议时长 (proposal_durations_seconds) 用于指示提交的时间长度。提交的延时可能与网络和磁盘 IO 相关。

等待的提议 (pending_proposal_total) 可以提醒用户当前有多少提议在队列中排队，等待提交。如果这个参数的值不断增大，这意味着客户端负载过高，或者集群已经不稳定。

失败的提议 (proposal_failed_total) 通常和两个因素相关：由领导者选举导致的临时错误，或者集群合法成员不足导致的长时间停服。

7.1.2 wal 指标

名称	描述	类型
fsync_durations_seconds	写 wal 文件的同步时间分布	Histogram
last_index_saved	Wal 文件中保存的最后一个索引	Gauge

异常高的同步延时 (fsync_durations_seconds) 暗示着磁盘可能出问题了，这可能导致集群不稳定。

7.1.3 http 请求指标

这些性能指标描述的是请求处理的状况，这些请求由工作在非代理模式下的 etcd 成员处理，而且被处理请求不包含监视事件。性能指标包含总的请求数量，请求失败的数量以及处理延时（即 raft 一致性存储的耗时）。这些指标有助于追踪用户向 etcd 服务器发起的请求。

所有的这些指标都以 etcd_http_作为前缀：

名称	描述	类型
received_total	解析和认证后的总事件数量.	Counter(method)
failed_total	失败事件的总数.	Counter(method,error)

名称	描述	类型
successful_duration_second	请求的响应时间，包含 raft 一致性写的时间.	Histogram(method)

举几个 有用的 Prometheus 请求的案例，这些案例的数据来源于上述的指标，而且针对的是全集群的成员：

```
sum(rate(etcd_http_failed_total{job="etcd"}[1m]) by (method) /
sum(rate(etcd_http_events_received_total{job="etcd"}[1m]) by (method)
```

上述案例展示的是集群所有成员中，一分钟内失败的 http 请求的百分比。

```
sum(rate(etcd_http_received_total{job="etcd",method="GET"}[1m]) by
(method) sum(rate(etcd_http_received_total{job="etcd",method!="GET"}[1m]) by (method)
```

上述案例展示的是集群所有成员中，一分钟内只读/写入请求的成功率。

```
histogram_quantile(0.9,
sum(increase(etcd_http_successful_processing_seconds{job="etcd",method="GET"}[5m]) ) by
(le))histogram_quantile(0.9,
sum(increase(etcd_http_successful_processing_seconds{job="etcd",method!="GET"}[5m]) ) by
(le))
```

上述案例展示的是集群所有成员中，五分钟内读写请求处理的 90%延时，单位是秒。

7.1.4 快照指标

名称	描述	类型
snapshot_save_total_durations_seconds	打快照的总延时分布	Histogram

异常高的快照延时（snapshot_save_total_durations_seconds）暗示着磁盘可能出问题了，这可能导致集群不稳定。

7.1.5 raft 指标

名称	描述	类型	标签
message_sent_latency_seconds	消息发送的延迟分布	HistogramVec	sendingType, msgType, remoteID
message_sent_failed_total	发送失败的消息总数	Summary	sendingType, msgType, remoteID

异常高的消息延时 (message_sent_latency_seconds) 暗示着磁盘可能出问题了, 这可能导致集群不稳定。

消息失败总数的上升 (message_sent_failed_total) 暗示着更加严重的网络问题, 这可能会导致集群不稳定。

标签 sendingType 是发送消息的连接类型。标签值 message, msgapp 以及 messagev2 使用的是 http 串流, 而 pipeline 则是为每个消息都发送一个 http 请求。

标签 msgType 则是 raft 消息的类型。MsgApp 是日志备份的消息; MsgSnap 则是快照建立的消息; MsgProp 提交转发的消息; 其它的值则用于维护 raft 算法内部的状态。如果用户的快照很大, 可能与遇到非常大的 msgSnap 发送延时。对于其它类型的消息, 延时会相对较低, 如果网络带宽足够的话, 其大小和 ping 的延时是相当的。

标签 remoteID 是消息目标的成员的 ID 号码。

7.1.6 代理指标

工作在代理模式的 etcd 成员并不执行存储操作。它会将所有请求都转发给集群的其它成员。追踪来自代理的请求速率, 有助于用户定位读写负载最高的节点。

所有这些指标都以 etcd_proxy_ 作为前缀。

名称	描述	类型
requests_total	该代理进程的请求总数.	Counter(method)
handled_total	接收到成员响应的成功处理请求的总数.	Counter(method)

名称	描述	类型
dropped_total	由于转发错误导致的失败请求数量	Counter(method,error)
handling_duration_seconds	请求处理的响应时间	Histogram(method)

举几个 有用的 Prometheus 请求的案例，这些案例的数据来源于上述的指标，而且针对的是全集群的成员：

```
sum(rate(etcd_proxy_handled_total{job="etcd"}[1m])) by (method)
```

一分钟时间内，所有代理所处理的请求的速率，这些请求通过 http 的方法字段来识别。

```
histogram_quantile(0.9,
sum(increase(etcd_proxy_events_handling_time_seconds_bucket{job="etcd",method="GET"}[5m])) by (le))
histogram_quantile(0.9,
sum(increase(etcd_proxy_events_handling_time_seconds_bucket{job="etcd",method!="GET"}[5m])) by (le))
```

在五分钟的时间窗口内，所有代理节点处理客户端请求的延时，区间是前百分之九十，单位是秒。

```
sum(rate(etcd_proxy_dropped_total{job="etcd"}[1m])) by (proxying_error)
```

代理处理失败的请求数量。这个值应该是 0，否则代理到 etcd 集群的连接出问题了。

7.2 prometheus 简介

在上一小节的开头，本文提到了 prometheus，当用户请求 etcd 服务器的 /metrics 接口时，输出的响应数据格式是 prometheus 能理解的，这是因为 etcd 默认使用 Prometheus 来汇报服务器的性能参数。这些性能参数可以用作实时的监控和调试。Etcd 仅仅将这些数据存储在内存中。如果成员重启了，那么数据会被重置。

Prometheus 是一个开源的监报告警工具，基于 golang 语言编写，它可以通过 etcd 暴露出来的 /metrics 接口，定期从 etcd 集群抓取性能数据，并且将这些数据打上时间戳，也就是将数据序列化，序列化后的数据被持久化到磁盘中，另一方面 prometheus 同样提供数据的查询功能，通过其暴露出来的 API，用户可以请求到这些序列化后的数据，更常用的场

景是，通过第三方的可视化工具 grafana 来向 prometheus 抓取序列化的数据，最终在 web 端以图表的形式展现给用户。

下面对 prometheus 的安装，配置与使用做个简单的案例分析，在这个案例中，本文将使用 prometheus 来监控它自己的性能。

7.2.1 安装

Prometheus 提供了多种安装的方式，可以直接下载二进制文件，也可以源码编译安装，还可以通过 docker 来运行 prometheus 服务，本文推荐使用二进制文件的方式来安装 prometheus。

下载二进制文件

```
$ curl
https://github.com/prometheus/prometheus/releases/download/v1.0.1/prometheus-1.0.1.linux-amd64.tar.gz
```

解压

```
$ tar xvf prometheus-1.0.1.linux-amd64.tar.gz
```

进入解压目录

```
$ cd prometheus-1.0.1.linux-amd64
```

7.2.2 配置

在解压出来的目录下，包含一份样本 prometheus 配置文件 prometheus.yml，文件内容如下：

```
global:
  scrape_interval:    15s # By default, scrape targets every 15 seconds.

  # Attach these labels to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
```

```
monitor: 'codelab-monitor'

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from
  this config.
  - job_name: 'prometheus'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s

static_configs:
  - targets: ['localhost:9090']
```

该配置文件就是用于监控 prometheus 服务本身的，所以无须修改，可以直接使用。

如果用户使用 prometheus 来监控自己的 etcd 集群，则需要相应修改配置文件中的 targets 属性，将该属性设置成访问 etcd 服务的客户端 url 即可。

7.2.3 使用

配置好 prometheus 之后，用户可以运行下面的命令开启 prometheus 服务：

```
./prometheus -config.file=prometheus.yml
```

此时用户可以通过浏览器打开 <http://localhost:9090> 来访问 prometheus 的数据。

7.3 grafana 简介

前文提到，可以通过 prometheus 来监控 etcd 的性能，但是 prometheus 的可视化图标功能过于简单，推荐的解决方案是使用 grafana 来做数据可视化的功能。

Grafana 是一个开源的可视化工具，内置了对 prometheus 的支持，可以在 grafana 的界面中配置使用 prometheus 作为数据来源，并对数据进行绘图。

下面通过一个简单的案例来说 grafana 的使用。

7.3.1 安装

下载二进制包：

```
curl -L -O  
https://grafanarel.s3.amazonaws.com/builds/grafana-2.5.0.linux-x64.tar.gz
```

解压到本地目录：

```
cd grafana-2.5.0/  
./bin/grafana-server web
```

默认情况下，grafana 监听在 3000 端口，初始用户名和密码分别是 "admin" 与 "admin"。

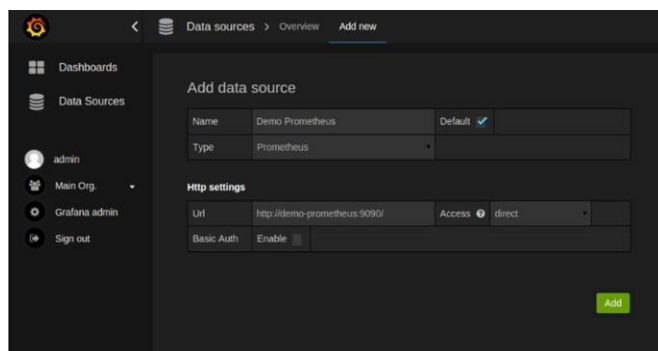
用户可以通过浏览器登录 [http://\[grafana_ip\]:3000/](http://[grafana_ip]:3000/) 来访问 grafana 的服务。

7.3.2 配置

在 grafana 的界面中，进行如下操作来配置 prometheus 的使用：

1. 点击 "Data Sources" 按钮；
2. 点击 "Add New" 按钮；
3. 在 type 下拉框中选择 "Prometheus"；
4. 设置 prometheus 服务的地址；
5. 点击 "Add" 来确认添加 prometheus。

最终的配置效果如下所示：



7.3.3 使用

在配置好 grafana 之后，可以开始绘制实时的动态图了，绘制步骤如下：

1. 先后点击“graph”和“Edit”按钮；
2. 在“Metrics”选项中选择上一小节配置的 prometheus 数据源；
3. 在“Query”方框中填入适当的 prometheus 表达式；
4. 在“Legend format”中输入{{method}} - {{status}}来查看指定的 http 请求和方法。

结果如下图所示：



7.4 本章小结

本章介绍了 etcd 集群的监控方案，通过使用 prometheus 来采集 etcd 的实时性能数据，并且在 grafana 中进行绘图，用户可以直观的获取到 etcd 集群的实时状态，并且根据状态来适当调整 etcd 集群。

本文至此，已经将所有 **etcd** 使用相关的部分都介绍完了，但是在实际的部署使用过程中，用户仍然需要评估 **etcd** 的性能，以确定 **etcd** 是否适用于特定的场景，本章将给出 **etcd** 的性能测试与结果分析，帮助用户决定是否选择 **etcd**。

Etcd 的性能测试涉及各个操作，本文只给出常用的键值对读取和键值对设置的性能。

8.1.1 测试工具

http boom 类似与 apache bench，有点是能够提供较为详尽的测试报告，缺点是同样的客户端资源，能够产生的压力负载有限，测试报告样本如下图，



8.1.2 测试方案

选取的物理节点参数如下：

1. 1 块专用 SSD 硬盘，挂载在 /var/lib/etcd 目录下，用于 etcd 的数据存取
2. 一块普通硬盘，用于操作系统的 IO
3. 1.8G 的内存
4. 2 块物理 CPU

etcd 集群包含三台节点，每个节点对应上述的一台物理节点，软件版本信息如下：

etcd 版本号：2.2.0-alpha.1+git

Git SHA: 59a5a7e

Go 版本：1.4.2

操作系统：linux/amd64

启动另外一台物理机，使用上一小节的 boom 来产生负载压力，请求 etcd 集群。

8.2 测试结果

读取一个键值对的性能如下：

键大小/bytes	客户端数量	目标 etcd 成员	每秒处理请求数量	90%的延时/毫秒
64	1	领导者	2804 (-5%)	0.4 (+0%)
64	64	领导者	17816 (+0%)	5.7 (-6%)
64	256	领导者	18667 (-6%)	20.4 (+2%)
256	1	领导者	2181 (-15%)	0.5 (+25%)
256	64	领导者	17435 (-7%)	6.0 (+9%)
256	256	领导者	18180 (-8%)	21.3 (+3%)
64	64	所有成员	46965 (-4%)	2.1 (+0%)

键大小/bytes	客户端数量	目标 etcd 成员	每秒处理请求数量	90%的延时/毫秒
64	256	所有成员	55286 (-6%)	7.4 (+6%)
256	64	所有成员	46603 (-6%)	2.1 (+5%)
256	256	所有成员	55291 (-6%)	7.3 (+4%)

写入一个键值对的性能:

键大小/bytes	客户端数量	目标 etcd 成员	每秒处理请求数量	90%的延时/毫秒
64	1	leader only	76 (+22%)	19.4 (-15%)
64	64	leader only	2461 (+45%)	31.8 (-32%)
64	256	领导者	4275 (+1%)	69.6 (-10%)
256	1	领导者	64 (+20%)	16.7 (-30%)
256	64	领导者	2385 (+30%)	31.5 (-19%)
256	256	领导者	4353 (-3%)	74.0 (+9%)
64	64	所有成员	2005 (+81%)	49.8 (-55%)
64	256	所有成员	4868 (+35%)	81.5 (-40%)
256	64	所有成员	1925 (+72%)	47.7 (-59%)
256	256	所有成员	4975 (+36%)	70.3 (-36%)

8.3 结果分析

在大多数情况下，键值对读取的每秒处理请求数量都有 5~8% 的下降。原因是 etcd 会为每一次的存储操作都记录性能指标。而这些性能指标对于监控以及调试是非常重要的，所以性能的损耗也是可以接受的。

键值对写入的每秒处理请求数量有 20~30% 的提升。这是由于 etcd 将 raft 的主循环与

日志写入的循环进行了解耦，这样子能避免它们相互阻塞。

当请求所有成员节点的时候，键值对写入的每秒请求数量有 30~80%的提升，这是因为跟随者能第一时直接收到最新的提交序列号，并且更快的写入该提交。

8.4 本章小结

本章给出了最基本的 etcd 集群性能测试方案以及结果，并对测试的结果进行了分析，以方便用户了解 etcd 集群的性能。

第九章 常见问题与解答

本章将对一些 etcd 使用过程中的常见问题进行解答，这些问题涵盖 etcd 的各个方面，能够帮助用户快速滤清关于 etcd 的一些疑惑。

9.1 一般性问题

1) 当大部分集群成员都停服之后，获取到旧版本的数据是怎么样的？

如果一个客户端连接上集群的小部分成员，默认情况下 etcd 认为集群的可用性高于一致性。这意味着即使数据可能已经过期，但是相比于空的响应，仍然会有数据返回给客户端。

为了确认读取到的数据是和集群的大部分成员保持最新的，客户端可以在读取键值对的时候指定参数 `quorum=true`。这意味着在读取数据返回之前，etcd 会去检验集群的合法节点数量，如果数量没有达到要求，那么读请求会超时失败。

2) 当设置 `quorum=false` 的时候，是否意味着，如果客户端请求的成员发生了变化，客户端可能会感知到集群的回滚？

是的，但是 etcd 客户端可以通过记住上一次获取到的索引来规避这种情况。这个索引是集群整个修改历史的不可更改的序列值。客户端可以记录下来最后一次获取到的索引值，然后通过比较当前获取到的索引值，进而判断当前获取到的键值对是否比上次看到的更加陈旧。

3) 如果监视被注册到小众成员会发生什么？

即使修改在大众成员中发生了，这个监视也会保持不触发。这是一个开放的问题，而且已经在版本 3 中得到了解决。有许多方法能够绕开监视无法触发的问题。

1. 创建一个独立于 etcd 的信号机制。可以简单的向客户端推送脉冲，重新发送一个带有 `quorum=true` 参数的 GET 请求，以获取最新版本的数据。

2. 在 `/v2/keys` 端节点进行 poll 轮询，检查确认 raft 索引是否在每一次超时后都会上升。

4) Etcdctl 的—endpoint 选项的工作原理是什么?

该—endpoint 标识可以在一个逗号分割的列表中指定集群中的任何成员。这个列表可以是集群成员的子集，也可以正好包含集群的所有成员，甚至包含一些不在集群中的成员。

如果—endpoint 标识仅仅制定了一个对端节点，etcdctl 通过该对端节点的成员列表来发现集群的剩余成员，接着会随机选取一个成员来使用。再一次声明，客户端在读取数据时可以使用 quorum=true 标志，这样一来如果使用的是小众成员，那么这次读取总会失败。

如果来自不同集群的成员同时出现在—endpiont 标识中，etcdctl 会随机选取一个成员，请求也会简单地被转发到某个集群中去。这也许不是用户想要的结果。

备注：--peers 标识已经被弃用，相反用户应该使用—endpiont 标识，因为原先的标识会误导用户提供对端节点的 url。

5) 当使用对端 tls 加解密时，为什么我的集群无法正常工作?

版本 2.0.x 的 etcd 内部使用的短连接的 http 进行数据传输。所以当用户使用对端 tls 加解密的时候，需要相应提高心跳间隔以及选举超时，以减小集群内部的扰动。一般来说，可以将参数设置成 -heartbeat-interval 500 -election-timeout 2500。到了 2.1.x 版本的 etcd，通过设计上的优化，减小了连接的数量，这个问题得到了解决。

9.2 etcd 如何处理成员关系以及健康检查

Etcd 的设计目标是通过简单的 API 来实现配置重载，而健康检查以及成员的添加/删除则取决于应用自身以及应用如何使用配置重载 API。

如此一来，即使一个成员永远宕机了，它也不会自动从 etcd 集群的成员列表中被删除。

这是合情合理的，因为这种需求往往由应用层面或者管理动作去决定是否基于集群的健康状况来执行配置重载。

更多的信息可以参考运行时配置重载的设计文档。

9.3 etcd 代理的用途是什么

一个代理是到 etcd 集群的转发服务器。代理将来自客户端的请求重定向到注册的 etcd 集群上面去。一个典型的应用是在一个机子上启动一个代理，并且在第一次启动的时候，指定 `--proxy` 标识，以及 `--initial-cluster` 标识。

代理启动之后，任何 `etcdctl` 客户端都会和本地的代理进行通信，代理进而将操作的指令转发给最初配置的 etcd 集群。

在版本 2 的 etcd 中，代理不能被提升为集群的成员。它们同样也不能被提升为集群的跟随者，任何情况下也不会成为集群日志备份的参与者。

9.4 etcd 与 zookeeper 的比较

zookeeper 可用于配置管理，分布式锁，使用 java 编写，集群成员之间通过 zab 协议来保持一致性，当网络出现分区状况时，系统将不可用。

相比之下，etcd 是一个用于配置管理和服务发现的高可用键值对存储系统，使用 go 语言开发，集群成员之间通过 raft 一致性算法来保证一致性，对外暴露 restful 的 API，由于需要一个非常强的领导者，所以当网络出现分区时，对集群的写入和读取操作可能返回失败，但是不会返回不一致的数据。

在原生接口和服务提供方面，etcd 更适合做配置管理，用以存储大量的数据，zookeeper 则更适合做分布式的协调服务，在实现分布式锁的功能时较 etcd 更加简单方便。

建议用户根据实际的需求来进行选择。

9.5 本章小结

本章对一些 etcd 使用过程中的常见问题进行了解答，最后给出了 etcd 与 zookeeper 的对比，通过本章的内容，用户能够更快的解答一些 etcd 的疑惑，并且更加理智的认识 etcd。

致谢

在白皮书编撰的过程中，得到了很多人的支持和帮助，其中陈永刚最先提出了白皮书的需求，并且在整个撰写过程中给出非常积极的指导意见和建议，同时，赖东林在文档的整理过程中，也贡献了大量的时间，很多素材也来源于此。

由于作者水平有限，文中难免有差错和不足之处，欢迎各位读者指正。