# **Kubernetes应用跨可用区实战**

<https://developer.aliyun.com/article/620279>

# 管理多个k8s 集群

<https://github.com/kubernetes-sigs/kubefed>

# 阿里万级规模k8s集群

**[当 K8s 集群达到万级规模，阿里巴巴如何解决系统各组件性能问题？](https://www.cnblogs.com/alisystemsoftware/p/11570806.html)**

<https://www.cnblogs.com/alisystemsoftware/p/11570806.html>

# 大规模的k8s集群如何构建？

https://openai.com/blog/scaling-kubernetes-to-2500-nodes/

[Kubernetes扩容到7,500节点的历程](https://mp.weixin.qq.com/s/l7bVpCt64QJSxlh4r1Hqyw)

<https://www.cnblogs.com/davygeek/p/6761090.html>

etcd集群2

etcd集群1

Make mirror

# Etcd 调优

参见：<https://zhuanlan.zhihu.com/p/29806621>

    etcd默认设置适用于网络低延迟。但网络延迟高，如网络跨域数据中心，那么心跳间隔和选举超时的设置就需要优化(tuning).

    网络慢不仅仅由于延迟，还可能被Leader和follower的低速磁盘IO影响。每一次超时设置应该包含一个请求发出到响应成功的时间。

**时间参数**

    当一个节点挂掉(stalls)变慢或者下线，分布式一致性协议(distributed consensus protocol)依赖两个时间参数保证领导力切换。

    第一个参数叫**心跳间隔**(*Heartbeat Interval*)。它代表leader通知所有的followers，他还是Leader的频率。最佳实践，它应该被设置为节点之间网络往返时间(round-trip time)。etcd默认心跳间隔是100ms。

    第二个参数是**选举超时**(*Election Timeout*RTT)。它表示follower在多久后还没有收到leader的心跳，它自己就变成Leader。etcd的默认选举超时是1000ms。

调整参数需要做一些权衡。心跳间隔推荐设置为节点之前的最大RTT，一般可设置为RTT的0.5-1.5倍。如果心跳间隔太短，etcd会发送没必要心跳增加CPU和网络的使用率。另外，过长的心跳间隔会增长选举超时时间。过长的选举超时导致要更长的时间才能发现Leader故障。测量RTT最简单方法就是用PING工具。

    选举超时应该基于心跳间隔和节点的平均RTT去设置。选举超时应该至少是RTT的10倍，这样才能视为网络延迟出现方差(variance)。例如，节点间的RTT是10ms，那么超时时间至少应该是100ms。

    选举超时时间最大限制是50000ms(50s), 只有etcd被部署在全球范围内时，才应该使用这个值。美国大陆的RTT是130ms，美国和日本的RTT是350-400ms。如果不均匀的网络性能或者常规的网络延迟和丢失，会引起多次网络重试，所以5s是一个安全的RTT最高值。5s是心跳间隔的最高值，所以超时时间最高值应该是50s。

    一个集群中的所有节点应该设置一样的心跳间隔和选举超时。如果设置不一样可能使集群不稳定。

    默认值可以被命令行参数或环境参数覆盖，单位是ms。

# 令行参数:

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

# 环境参数:

$ ETCD\_HEARTBEAT\_INTERVAL=100 ETCD\_ELECTION\_TIMEOUT=500 etcd

**快照(Snapshots)**

    etcd追加key的变更到日志中。这些日志用一行记录一个key的变更，日志就不断的增长。当简单使用etcd时这些日志增长不会有问题，但大集群的时候日志就会越来越多。

为了避免大量日志，etcd会定期制作快照。这些快照通过保存日志里的修改到当前状态和移除老的日志去压缩日志。

**快照优化**

    创建快照对于v2版是昂贵的，所有只有当更改记录操作一定数量后，才会制作快照。默认，是每10000次更改才会制作快照。如果etcd的内存和磁盘使用率过高，可以降低这个阀值(threshold)。

# Command line arguments:

$ etcd --snapshot-count=5000

# Environment variables:

$ ETCD\_SNAPSHOT\_COUNT=5000 etcd

**磁盘**

    etcd集群对磁盘的延迟非常敏感。因为etcd需要存储变更日志，多个进程操作磁盘可能引起更高的fsync延迟。这些可能引起etcd丢失心跳，请求超时或者Leader临时丢失。可以通过提高etcd进程的磁盘优先级。

    在Linux, etcd的磁盘优先级可以通过ionice去配置

# best effort, highest priority

$ sudo ionice -c2 -n0 -p `pgrep etcd`

**网络**

    如果etcd的Leader服务大量并发客户端，这就会导致follower的请求的处理被延迟因为网络延迟。follower的send buffer中能看到错误的列表。

dropped MsgProp to 247ae21ff9436b2d since streamMsg's sending buffer is full

dropped MsgAppResp to 247ae21ff9436b2d since streamMsg's sending buffer is full

    这些错误可以通过提高Leader的网络优先级来提高follower的请求的响应。可以通过流量控制机制来提高

tc qdisc add dev eth0 root handle 1: prio bands 3

tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip sport 2380 0xffff flowid 1:1

tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip dport 2380 0xffff flowid 1:1

tc filter add dev eth0 parent 1: protocol ip prio 2 u32 match ip sport 2739 0xffff flowid 1:1

tc filter add dev eth0 parent 1: protocol ip prio 2 u32 match ip dport 2739 0xffff flowid 1:1

# 腾讯云国内各机房延迟值

北京到上海：38ms  
上海到广州：40ms  
北京到广州：53ms



# 磁盘IO延迟

Even a cheap consumer **SSD** can at least sustain about 5000+ **IOPS** with only a 0.15 millisecond (150 microseconds) **latency**. That **latency** is about 40x better than the best **latency** of an enterprise 15K RPM hard drive. **Solid state drives** can often handle **I/O** requests in parallel。