KEYSPACE

2025 / 北京

Valkey高可用切换技术演进及最佳实践

杨博东

阿里云研发，Valkey-Java Maintainer

Valkey

Agenda

- 高可用技术的重要性
- 高可用切换技术演进
- 无感切换技术实现
- SDK的最佳实践

Valkey

高可用技术的重要性

Valkey

故障场景的业务稳定性

- Valkey的使用场景：**热数据缓存、会话、限流、排行榜、配置缓存**。
- 业务特征：**高QPS、高并发、在线业务、SLA要求高**。
- 高可用技术扮演的角色：切换的快慢与可靠直接决定了业务受损的情况，与业务稳定性挂钩。

Valkey

高可用的衡量指标

- RTO：（恢复时间目标）：主从切换要控制在多少秒以内。
- RPO：（数据恢复点目标）：允许丢多少数据。
- 常见设计冲突：切换越快 -> 误判风险越大；减少数据丢失 -> 切换流程越重。

Valkey

主动运维场景的业务稳定性

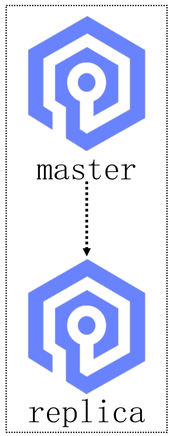
- 主动运维的场景：实例主动切换、小版本升级、故障机器的提前预警下线。
- 主动切换 VS 故障切换的比例：8：2
- 举例：某游戏客户，自己实现客户端，在主动运维切换后由于断连，客户端未重连导致连接协议错乱，从而读取数据失败，影响游戏页面展示。

Valkey

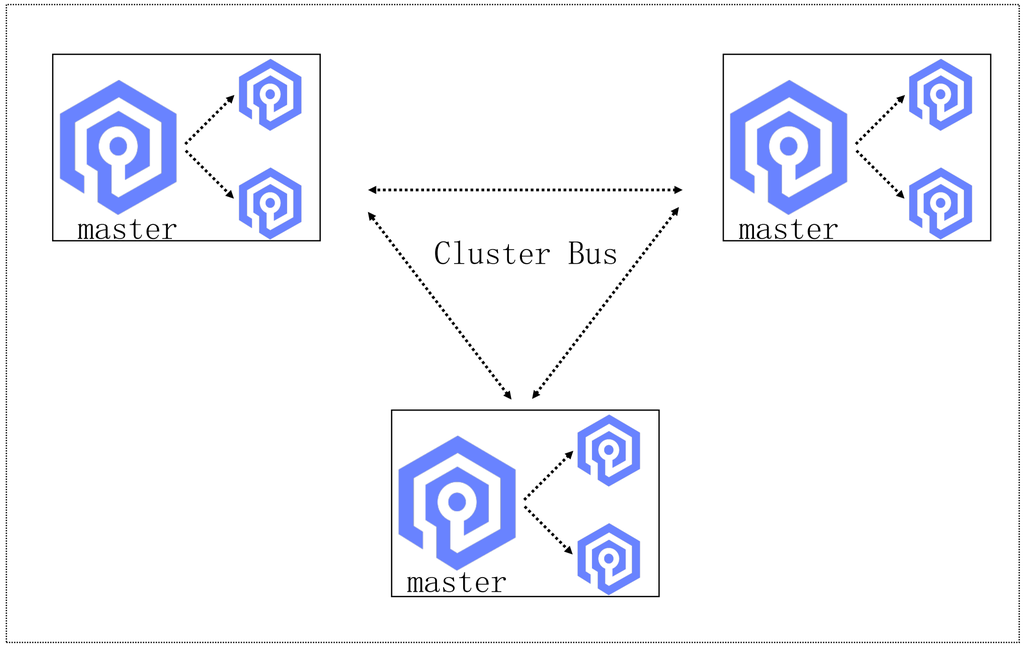
高可用切换技术演进

Valkey

Valkey的两种架构



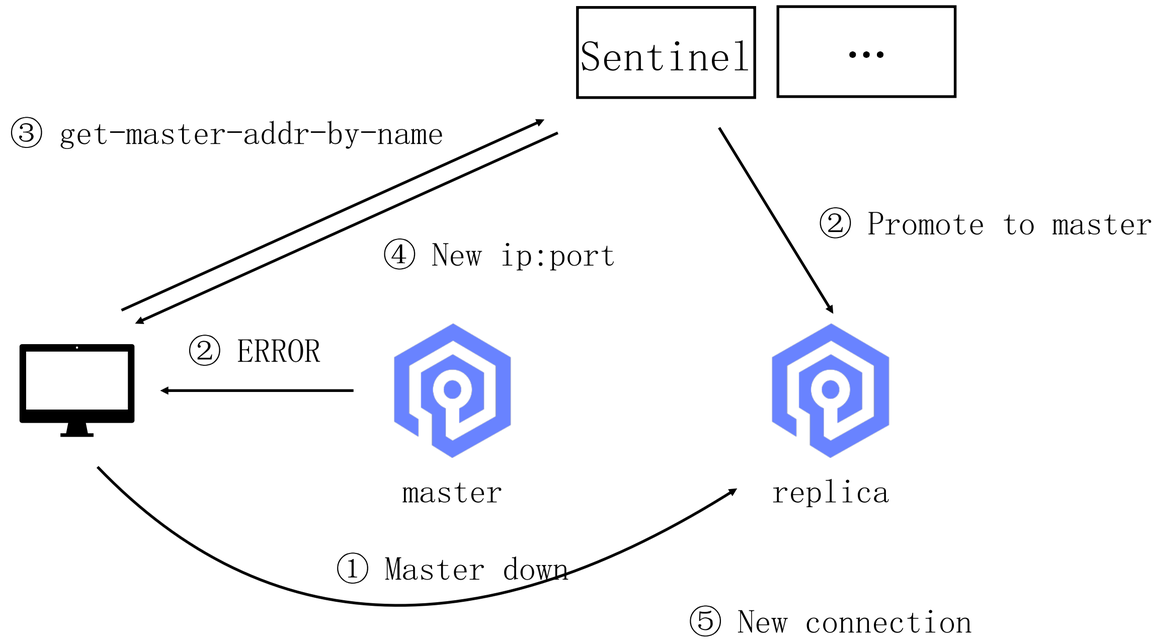
Standalone



Cluster

Valkey

高可用架构：Sentinel



优势

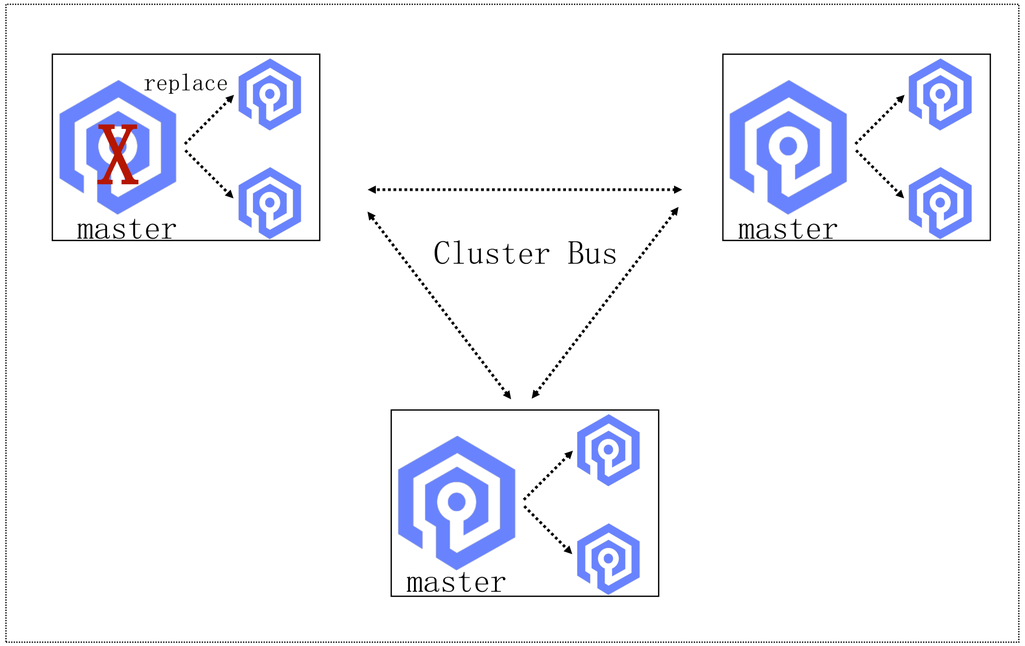
1. 官方原生、部署相对简单。
2. 自动故障检测与主从切换。
3. 为客户端提供统一主库发现能力。

劣势

1. 只适用于主从。
2. 切换速度较慢。
3. Sentinel集群本身运维门槛高。

Valkey

高可用架构：Cluster



Valkey Cluster

优势

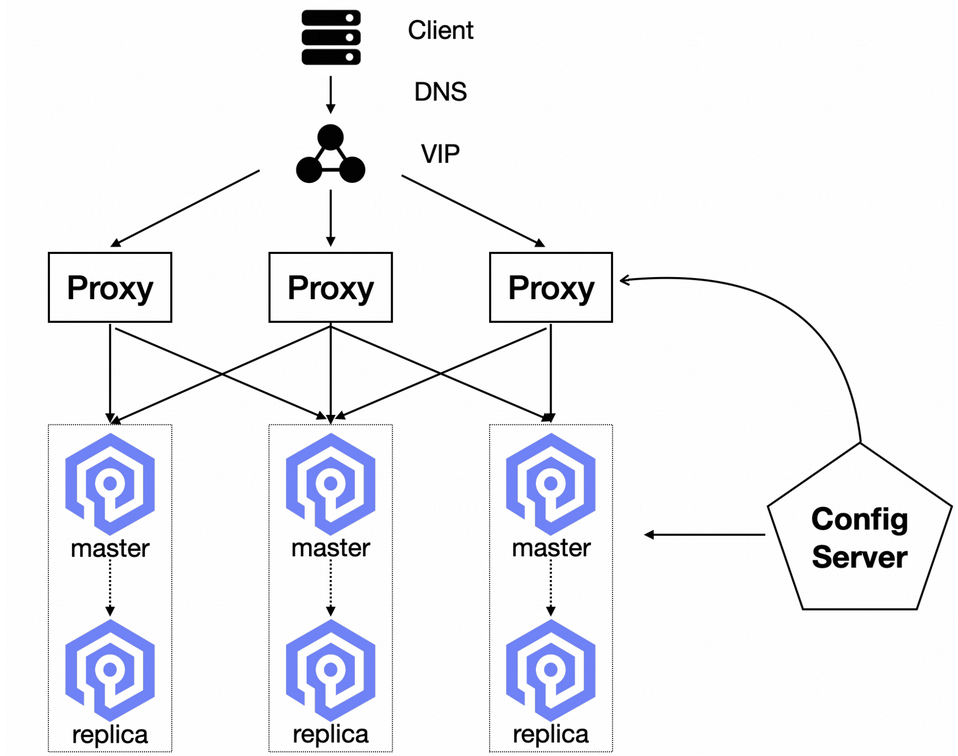
1. 原生分片和高可用一体。
2. 自动主从切换，节点角色自管理。
3. 路由表分布式存储，抗单点失败能力强。

劣势

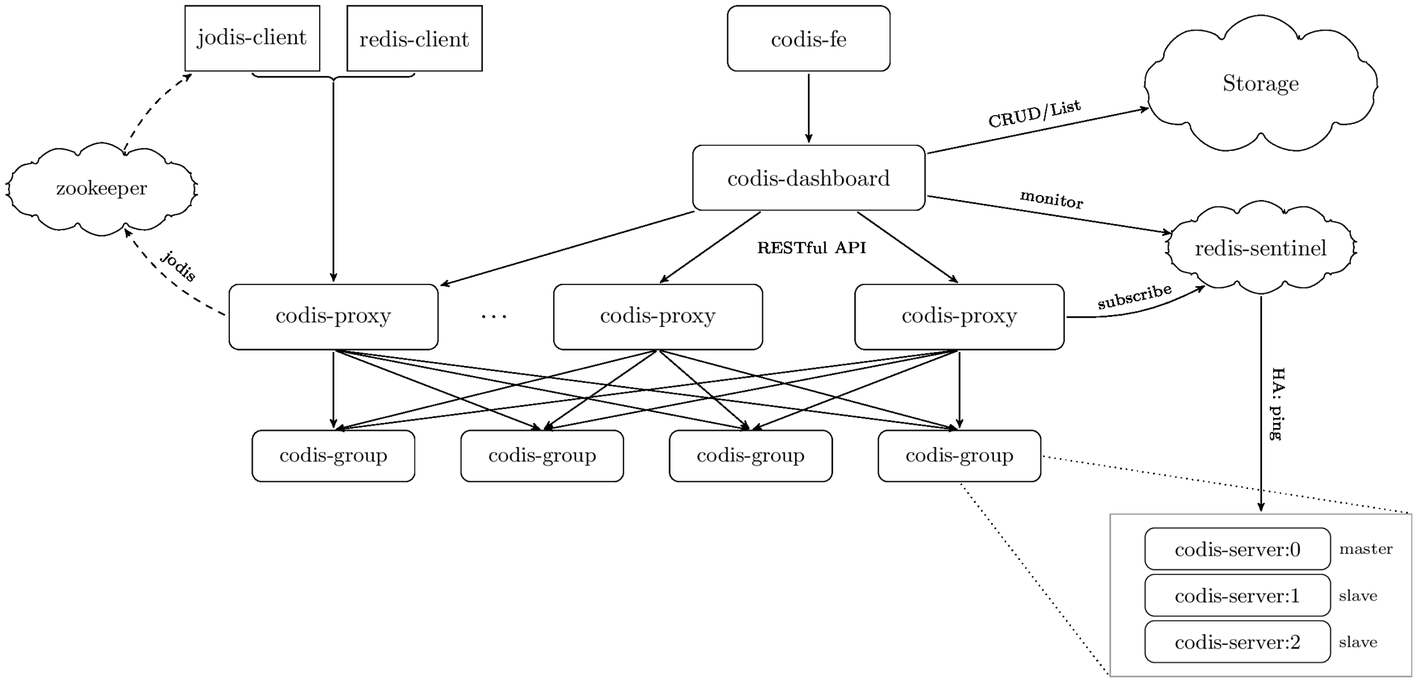
1. 大集群下Gossip收敛速度慢。
2. Cluster Bus对网络稳定性要求高，容易误判。
3. 出现问题排查困难，需要在多个节点追踪日志。

Valkey

高可用架构：Cloud/Built-in Service



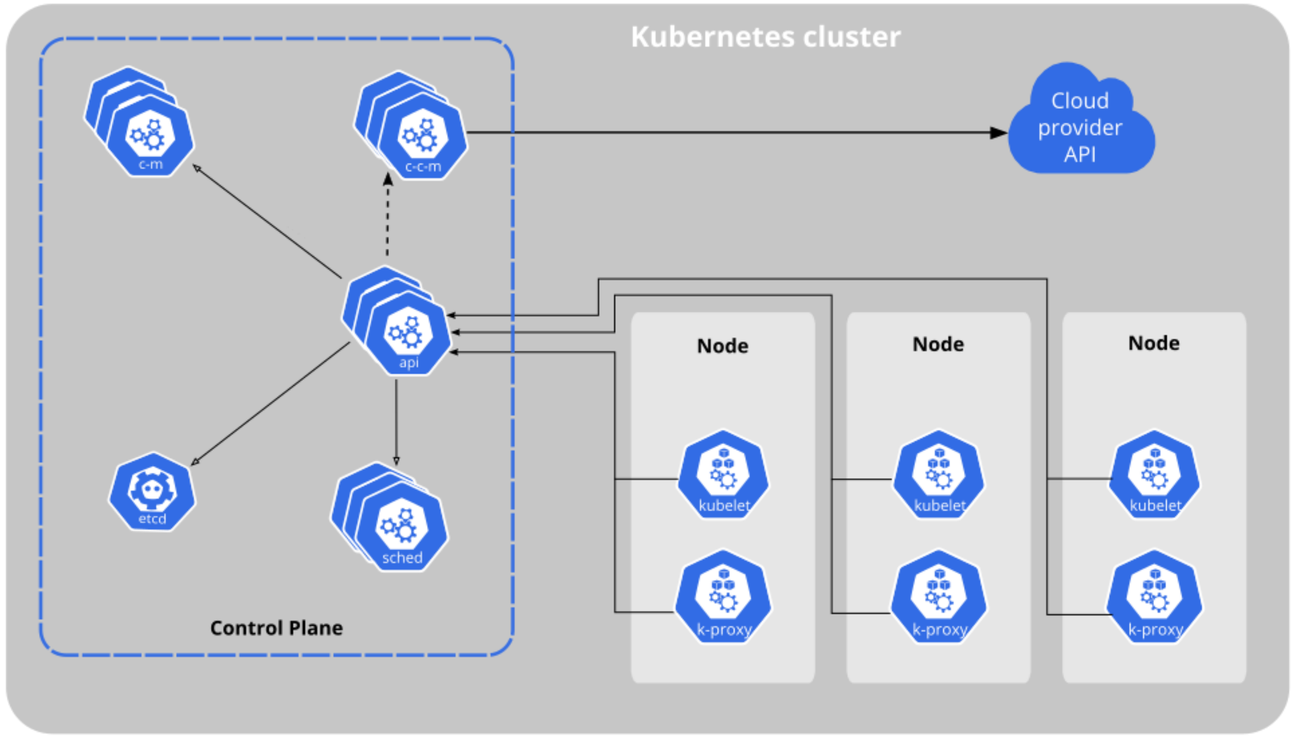
Cloud Architecture



Codis

Valkey

高可用架构：Cloud Native(K8S)



- 基于 K8S 的 List-Watch 模型，operator 等能力实现数据库等有状态服务的高可用。
- https://github.com/apecloud/kubeblocks
- https://github.com/hyperspike/valkey-operator

Valkey

无感切换技术实现

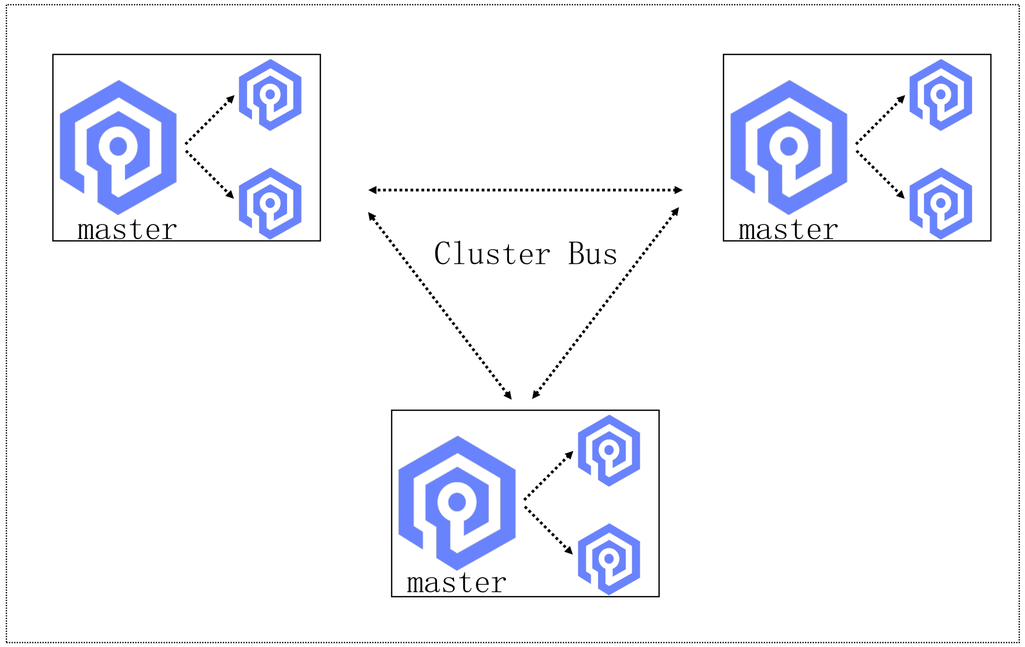
Valkey

无感切换的含义

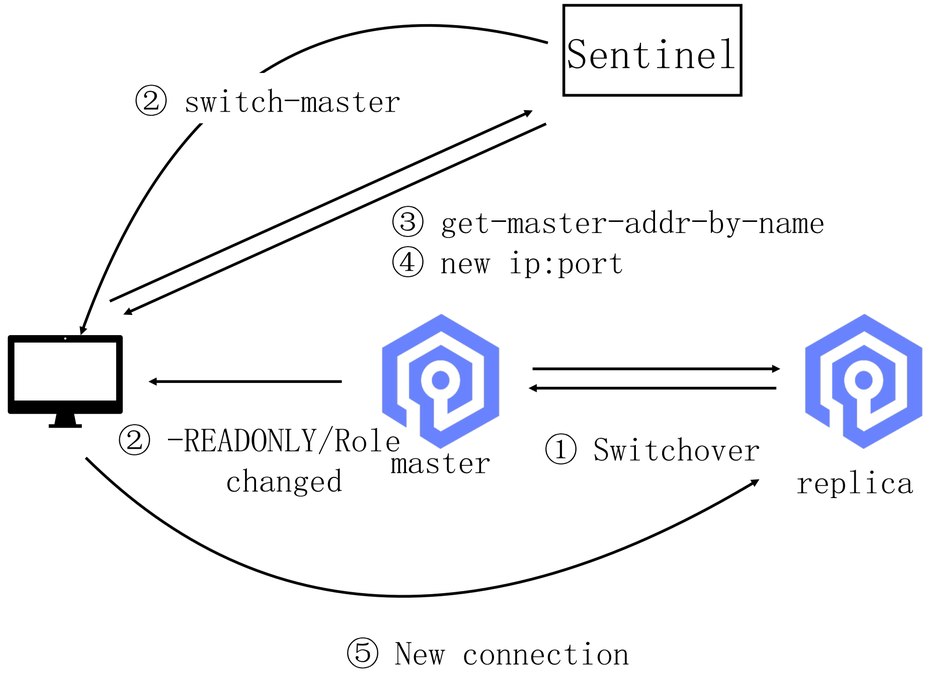
- **无感切换的优势如下：**
- 客户业务不会遇到连接断开错误。
- 业务只会遇到短暂的RT上升，不会出现长时间不可用。
- 降低业务不可用时长90%以上。

Valkey

集群和Sentinel的优势与问题



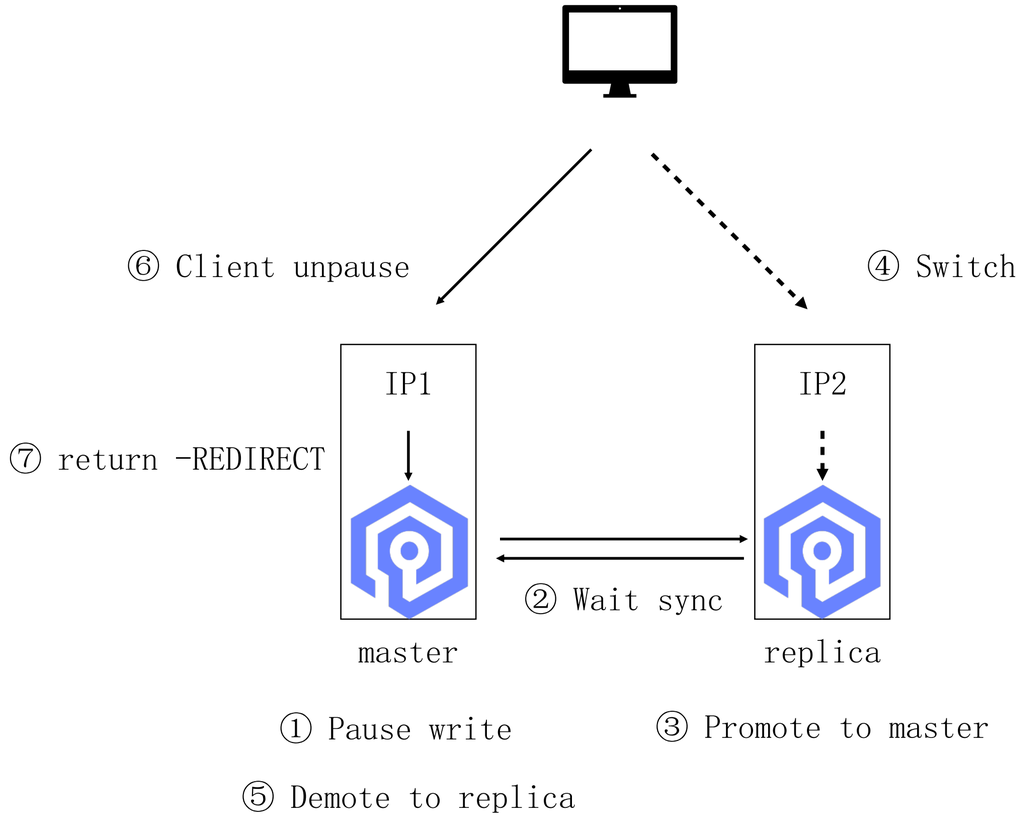
Cluster架构天然支持无感切换，MOVED协议平滑



Sentinel 主动切换，时效慢&不平滑

Valkey

主从无感切换架构



1. 对主库执行Pause write
2. 等待主备数据同步
3. 将备库提升为主库
4. 切换访问，新连接将建连到IP2
5. 将IP1变更为replica，挂载到IP2
6. 在IP1执行client unpause
7. 对于之前的写命令返回 REDIRECT IP2 PORT2
8. 客户端重发重连并且重新执行命令。

Valkey

无感切换：客户端支持

```
private final ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
private final Lock r = rwl.readLock();
private final Lock w = rwl.writeLock();
private final Lock rediscoverLock = new ReentrantLock();
```

```
public void renewPool(Connection connection, HostAndPort targetNode) {
  if (rediscoverLock.tryLock()) {
    try {
      // if memberOf is closed, means old flight connection return redirect exception, just return.
      if (connection != null && connection.getMemberOf().isClosed()) {
        return;
      }
      // if targetNode is not null, update the new address
      HostAndPort oldNode = hostAndPort;
      if (targetNode != null) {
        this.hostAndPort = targetNode;
      }
      // close old pool and create new pool
      w.lock();
      try {
        if (!pool.isClosed()) {
          try {
            pool.close();
          } catch (JedisException e) {
            logger.warn("close pool get exception, hostAndPort:{}", oldNode, e);
          }
        }
        this.pool = new ConnectionPool(hostAndPort, clientConfig, poolConfig);
      } finally {
        w.unlock();
      }
    } finally {
      rediscoverLock.unlock();
    }
  }
}
```

- 已经支持主从无感切换的客户端
- ✓ https://github.com/valkey-io/valkey-java
- ✓ https://github.com/valkey-io/valkey-go
- https://github.com/valkey-io/valkey-py
- https://github.com/valkey-io/valkey-glide

Valkey-Java 如何处理In flight的连接：对于连接池中同时并发访问的链接，如果某条或者多条同时遇到了-REDIRECT之后，通过一个二段锁来控制重新初始化连接池的过程：

1. 第一层锁使用ReentrantLock的tryLock，当tryLock成功时候才可以进入候选Renew连接池的可能。
2. 第二层锁使用ReentrantReadWriteLock，主要用来控制连接池的的读写，但加入写锁之后，将会block外部API请求，直到更新完毕连接池。

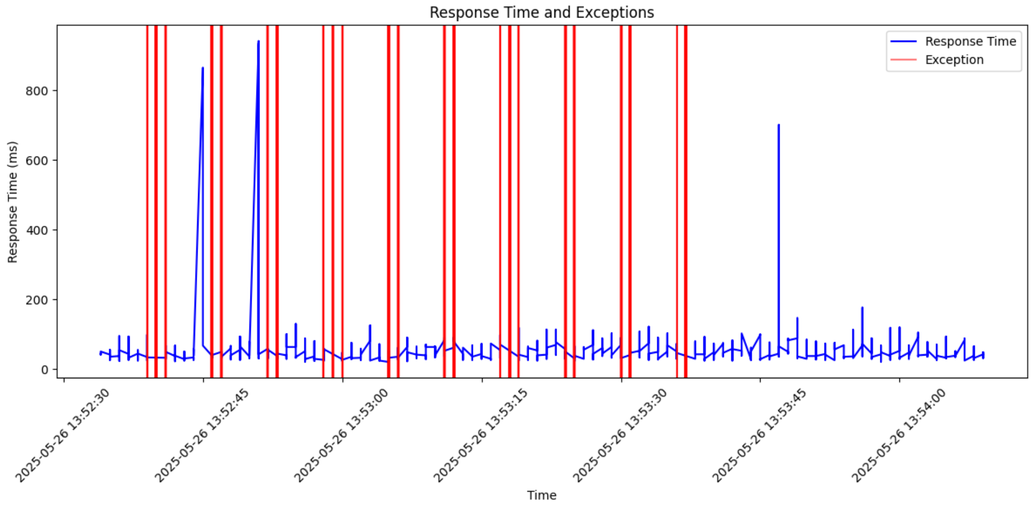
无感切换：测试效果

测试流程：

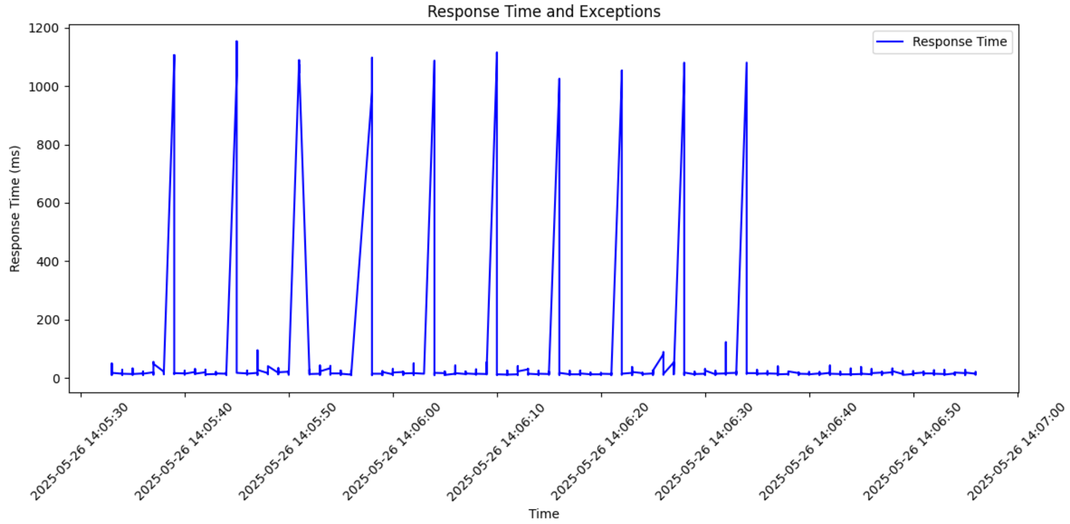
1. 程序1不断访问Valkey，记录每次操作的rt，如果遇到任何异常进行记录，表示本次访问失败。

2. 程序2对Valkey进行10次无感和普通切换，如果是无感切换，采用pause+redirect切换，如果是普通切换，则使用禁写+断连接的方式。

普通切换的情况如下：其中蓝色的线表示RT，红色的部分是遇到Exception的情况。



普通切换



无感切换

SDK最佳实践

Valkey

Jedis/Valkey-Java在Cluster下的一些坑

- 总体版本建议：使用Jedis 4.x.x或5.x.x 版本，请升级至Jedis最新版本；使用Jedis 2.x.x或3.x.x版本，请升级至Jedis 3.10.0及以上版本。
- 当集群中初始化的endpoint均失效时候，会导致实例无法成功连接到任一节点，修复PR：https://github.com/redis/jedis/pull/3370
- 当集群中没有任何访问，但后端节点发生迁移时候，客户端无法感知到新的路由表，解决方法是打开周期性刷新路由表的机制：https://github.com/redis/jedis/pull/3596
- 当Jedis访问的DNS地址后端有多个节点时，为了均衡访问，需要升级到4.1.0及以上，https://github.com/redis/jedis/pull/2722

Valkey

Thanks & QA