# Aerospike 技术分享

## 目录

本次分享的三个问题：

1，什么是aerospike？

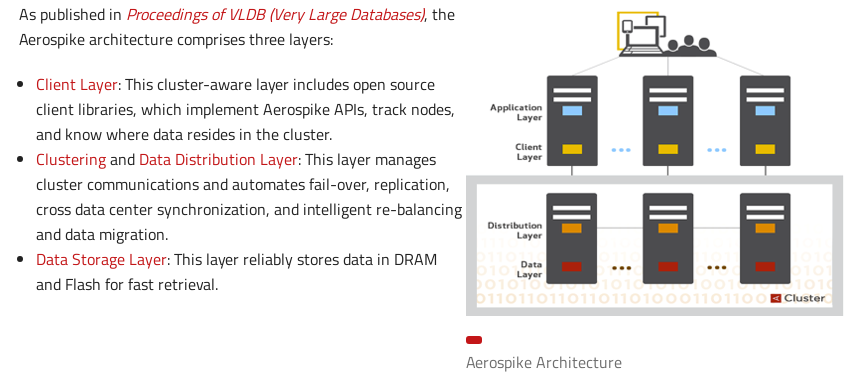
2，同类产品中，它的优势是什么？

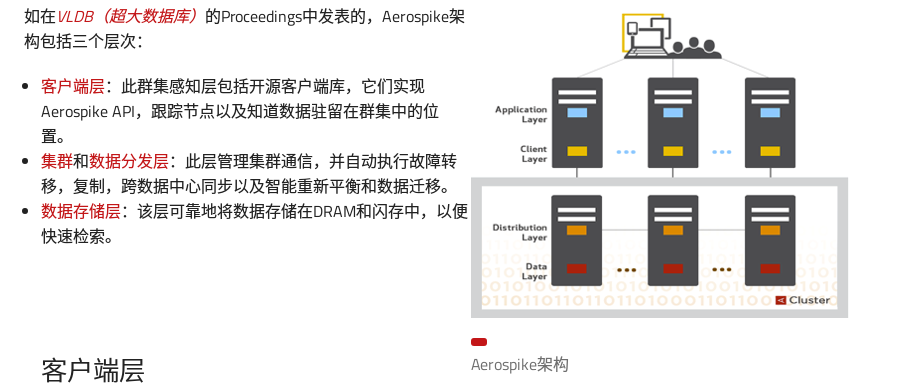
3，怎么用？

### 1，什么是aerospike？

Aerospike是一个分布式，可扩展的NoSQL数据库。

了解一下aerospike的架构





#### 数据模型

##### Namespace

Namespace是aerospike中的顶级容器，相当于mysql的database。

命名空间包含记录，索引和策略。策略规定命名空间行为，包括：

* 可以设置存储在RAM或磁盘上。
* 可以设置一条记录，有多少个副本。
* 可以设置当前命名空间中，每条记录的默认到期时间。

详情可以看：<http://www.aerospike.com/docs/operations/configure/namespace>

##### Sets

在命名空间中，记录属于set。可以理解成mysql的table。

Set集成namespace的策略，并且可以定义特定于集合的其他策略或操作，例如，可以仅针对特定集合的数据指定辅助索引，或者可以对特定集合执行扫描操作。

##### Records

Aerospike数据库是一个专门针对个别记录的行存储。记录是数据库中的基本存储单元。记录可以属于命名空间或命名空间中的集合。记录使用密钥作为其唯一标识符。

记录包括以下内容：

Key：唯一标识符。记录可以使用其键的哈希来寻址，称为摘要。

Metadata：记录版本信息（生成）和配置的到期时间，称为*生存时间*（TTL）。

Bins：每条记录都可以有很多字段。相当于mysql，表中的字段。但却是针对行的字段，每行可以有自己的，不同于其他行的字段数，字段名。

Metadata：

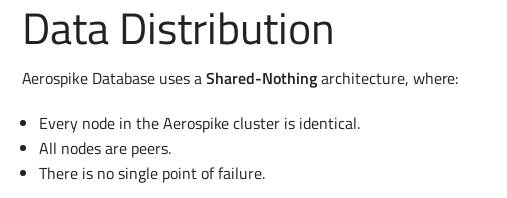
每个记录包含以下元数据：

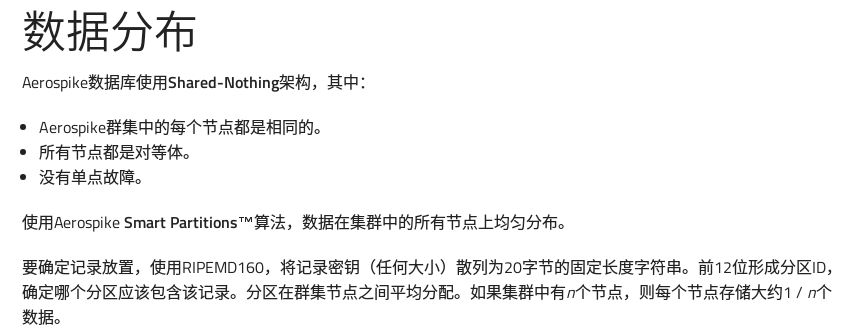
生成轨迹记录修改周期。该数字在读取时返回给应用程序，可以使用它来确保自上次读取以来没有修改正在写入的数据。

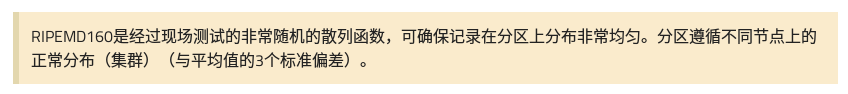
生存时间（TTL）指定记录到期。Aerospike会根据TTL自动过期记录。每次写入记录时，TTL都会增加。对于服务器版本3.10.1及更高版本，客户端可以设置更新记录时不更新TTL的策略。有关详细信息，请参阅相应的客户端API文档。

最后更新时间（LUT）指定时间戳记记录已更新。这是数据库内部的元数据，不会返回给客户端。

##### 数据分布

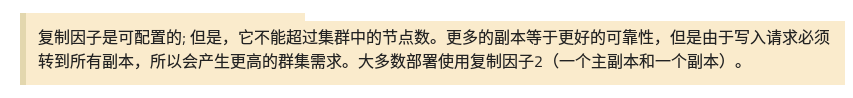












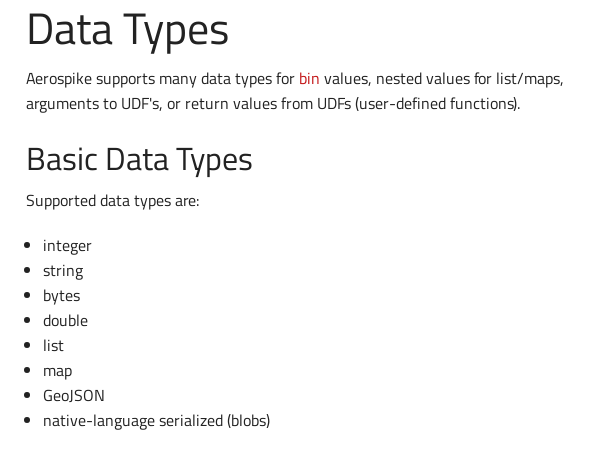
每台机器的数据量=集群数据总量÷节点数×副本数(复制因子)

所有复本key也会被加载到当前节点的内存中。

每个key固定占用64bit内存。

内存使用量(GB)=当前节点记录数+当前节点副本记录数×64/1024/1024/1024

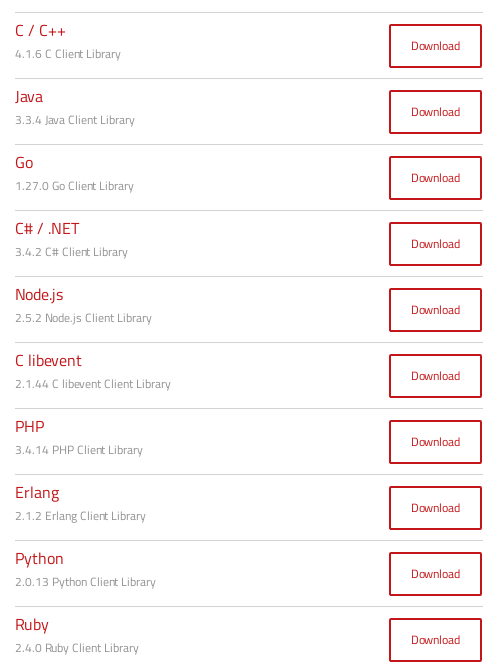
##### 数据类型

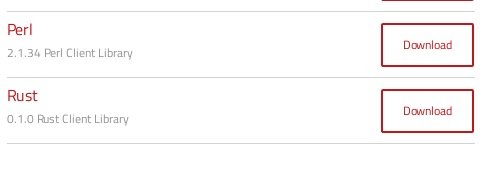


每种类型的详细介绍和空间占用可以参考：

<http://www.aerospike.com/docs/guide/data-types.html>

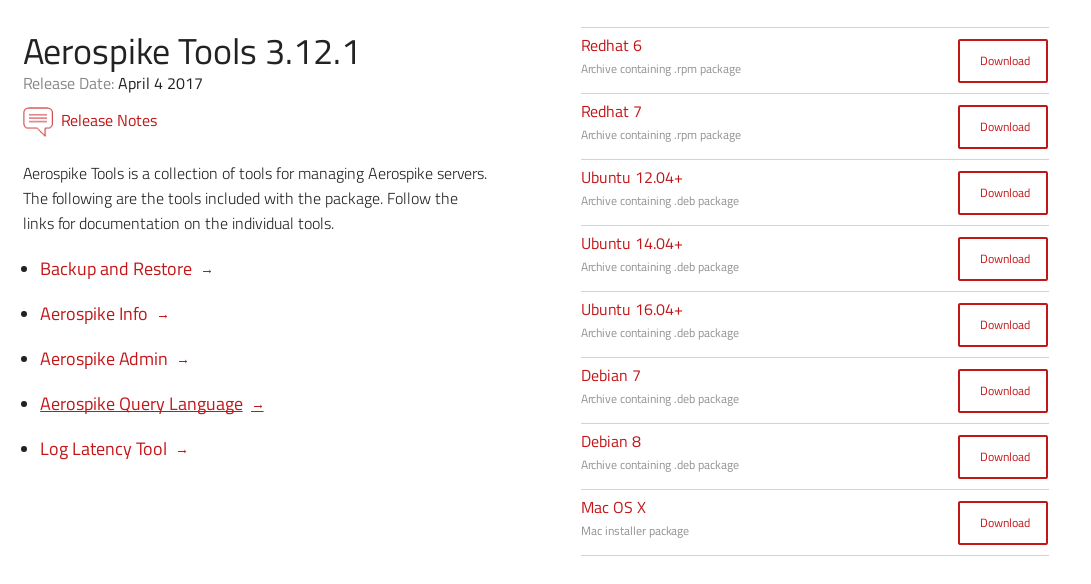
Aerospike 客户端的实现都支持哪些语言？





Url:http://www.aerospike.com/download/client/

Aerospike提供了哪些工具？



### 2，同类产品中，它的优势是什么？

在当下nosql产品中，大家耳熟能详，并且公认性能优越的nosql应该是redis无疑了，今天我们就针对redis和aerospike做个对比：

鉴于我这种小人物，说再多再好都是胡说八道，所以今天我们直接基于业界非常具有权威性的机构，基于aerospike和redis做的基准测试来做说明。（主要是懒）

这个具有权威性的测试是：

联合拥有大数据和云架构师、AWS社区英雄、谷歌云开发专家、微软MVP（SQL Server）等众多头衔的[Lynn Langit](http://lynnlangit.com/about/)在AWS云的虚拟机上对AerospikeDB和Redis进行了基准测试。

地址：<https://lynnlangit.com/2015/01/28/lessons-learned-benchmarking-nosql-on-the-aws-cloud-aerospikedb-and-redis/>

区别：

关于数据分片，和服务扩展的可管理性：

Redis：

1，必须自己实现分片算法，确保数据分布在所有redis节点的均衡和正确性。

2，一些java客户端（jedis应算是最常用的）虽然提供了这种操作，但您必须检查，以确保数据正确和平衡。

3，如果redis集群要增加新的机器，重新设计和验证分片算法，并且要停机，那将会是噩梦。

4，如果复制或故障切换，需要在应用层自己同步数据。

Aerospike：

1，自动处理等效的分片。

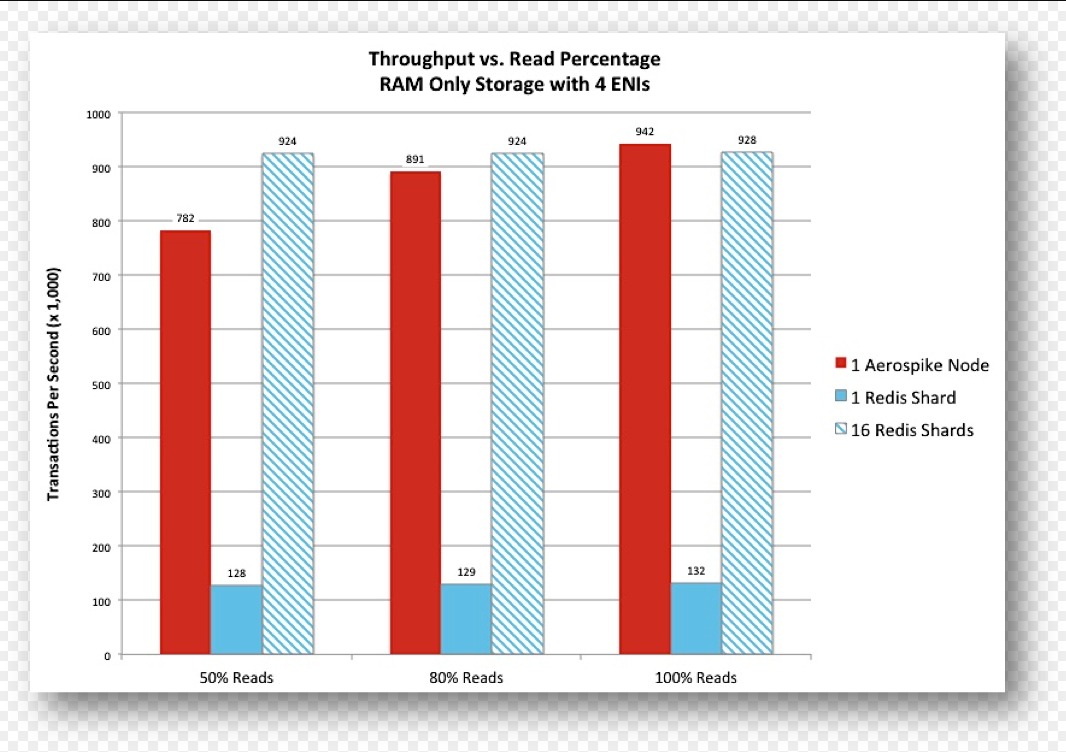
2，不需要停机，就可以动态的添加删除节点。Aerospike会自动重新平衡数据和流量（请求）。

3，复制因子设置为2，或更多，服务配置为同步复制，即可保持数据的一致性。

4，如果一个节点服务掉线，aerospike可以透明的处理故障转移。

5，可以只在内存中运行，也可以利用ssd/flash 进行存储，但会有轻微的延迟，平均延迟在0.1ms。

结果：



### 3，怎么用？

### 服务端安装：

下载：

<http://www.aerospike.com/download/server/>

这个地址会自动跳到最新的版本下载页面，client的下载也在这里。

下载了tar包后，解压，root权限运行 ./asinstall脚本即可。

服务端安装结束。

### 配置服务端：

配置参数说明：

<http://www.aerospike.com/docs/reference/configuration>

下面是在我们optimadRTB项目线上的配置说明：

目前没有太多的机器用来做实验，所以优化可能还不够好，当前还在优化中。

这里只整理了一些我认为可能重要的，但不是全部。

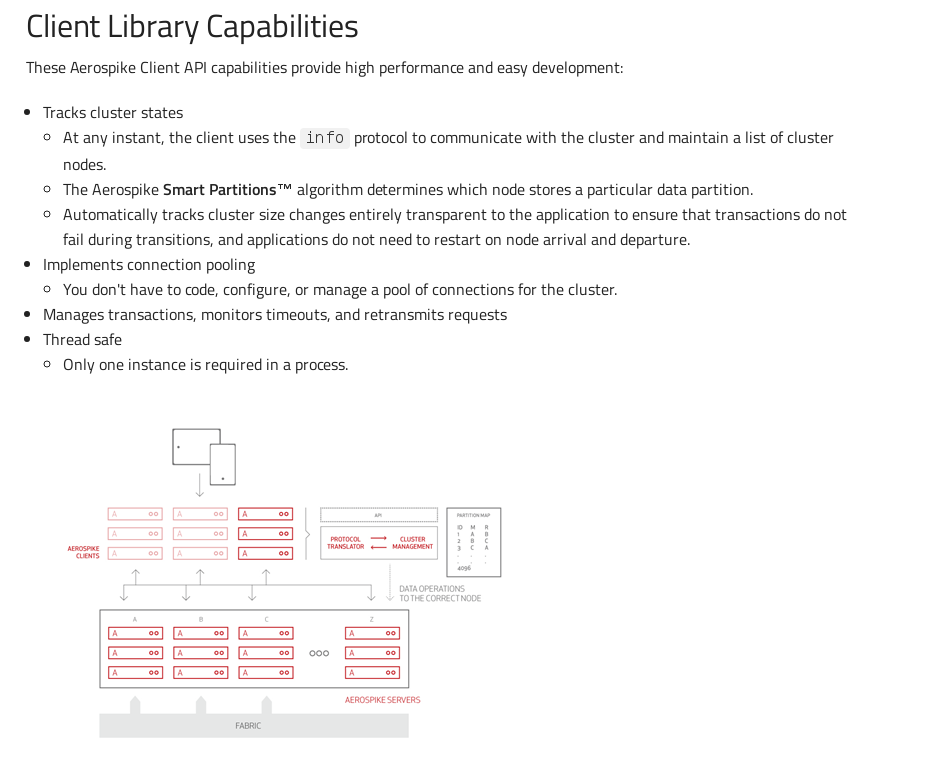
还好当时做的时候写了一些注释，准备这个分享文档的时候省了不少时间。

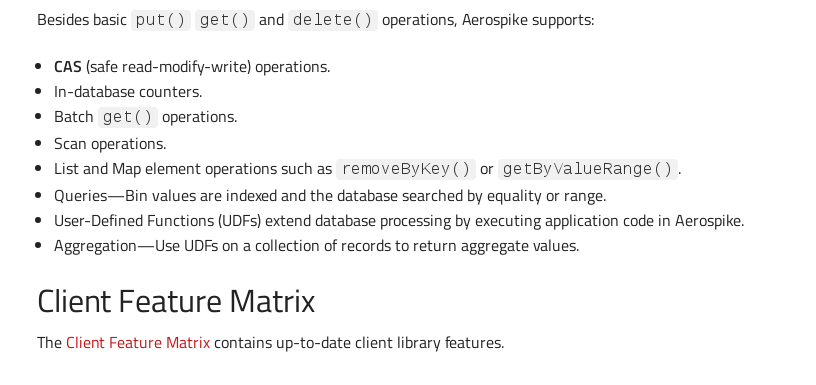
# Aerospike database configuration file.  
  
service {  
 user root  
  group root  
  # 当heartbeat.protocol = v3时，可以通过cluster-name来设置节点需要加入的集群  
  cluster-name optimad  
  # 设置aerospike运行时是前台运行，还是后台运行，true为后台运行  
  run-as-daemon true  
  # 来自客户端的最大连接数 默认15000  
  proto-fd-max 20000  
  # 如果集群大小小于等于这个值，将只保留数据的一个副本，也就是没有副本。  
  paxos-single-replica-limit 1  
  pidfile /var/run/aerospike/asd.pid  
  service-threads 6  
  # 客户端请求事务的队列数，基于cpu核心数即可  
  transaction-queues 6  
  # 处理客户端请求事务队列的线程数，基于cpu核心数×2即可  
  transaction-threads-per-queue 12  
  proto-fd-max 15000  
  # 用于数据迁移的线程数，默认1.由于将集群节点之间的通道数量增加了，这里也增加一些线程，提高数据迁移的效率。  
  migrate-threads 4  
  # 数据迁移时，当前节点可同时接收的分区数，适当减小可以减少阻塞，从3.10.1版本开始从256降到了4，这里适当加一些到8  
  migrate-max-num-incoming 8  
  # 来自客户端的事务重试时间 默认1000毫秒，3.10.1之后这个配置不再影响数据迁移的事务重试时间，不知道应该减少还是增加，增加了怕事务处理的慢，减少了怕失败。  
  transaction-retry-ms 1000  
  transaction-max-ms 1000  
  # 可以排队等待处理的最大待处理事务数。默认20.  
  transaction-pending-limit 200  
  # 在事务线程中运行查询，而不是使用查询线程。 当希望查询在短时间内运行或命名空间在内存中时，设置为“1”。 如果您期望更长的运行查询或命名空间使用磁盘存储空间，请将其设置为“0”。  
  # 我们的namespace使用硬盘存储，这里设置为0  
  query-in-transaction-thread false  
  # 查询线程的优先级  
  query-priority 10  
  # 优先查询后服务暂停时间，单位毫秒  
  query-priority-sleep-us 0  
  # 查询专用的线程数 cpu核心数×2  
  query-threads 6  
  # 节点专用的io线程数  
  query-worker-threads 12  
  # Amount of disk I/O a query performs per I/O request. 不太懂  
  query-batch-size 100  
  # 一次查询使用的最大I/O线程数  
  query-req-max-inflight 100  
  # 长查询的分界线  
  query-threshold 10  
  # 短查询最大的返回记录数  
  query-short-q-max-size 500  
  # 长查询最大的返回记录数  
  query-long-q-max-size 500  
  # 缓存数据的时间范围，单位秒，标记启用/禁用 read，write\_master,write\_reply,proxy 默认1800秒  
  hist-track-back 1800  
  # Comma-separated bucket (ms) values to track, must be powers of 2. For example : 1,4,16,64.  
  hist-track-thresholds 1,8,64  
  # 直方图数据缓存时间  单位秒  
  hist-track-slice 10  
  # 每个节点最大可处理的批量请求数。  
  batch-max-requests 5000  
  # Number of sequential commands before yielding. A higher number gives a higher priority. Only applies to old batch direct protocol.  
  batch-priority 200  
  # Number of batch direct worker threads. Batch direct is the old batch protocol. These threads process the full batch requests. There is one batch queue for all batch threads.  
  batch-threads 4  
  # Tuning parameter for how often to run retransmit checks for paxos.  
  paxos-retransmit-period 5  
  # 当集群出现脑裂，自动回复master数据  
  paxos-recovery-policy auto-reset-master  
  # 连接的最大空闲时间，默认60000ms 一分钟  
  proto-fd-idle-ms 60000  
  # 允许的最大scan命令的数量  
  scan-max-active 100  
  # 维持完成的scan的监控的最大数。  
  scan-max-done 100  
  # scan的线程数 默认4  
  scan-threads 6  
  # cpu的控制  
  # none - relying on kernel's irqbalance.  
  # cpu - CPU pinning - ASD controls the interrupt affinity of all NIC queue interrupts.  
  # numa - CPU and Numa Pinning - restrict memory and cpu usage of ASD to a single NUMA node.  
  auto-pin none  
  service{  
 # v1 = Original protocol version  
    # v2 = Expandable cluster size protocol version  
    # v3 = SIndex query node protocol version  
    # v4 = Rack Aware protocol version  
    # none = Used only for dynamically changing protocol  
    paxos-protocol v3  
  }  
}  
  
logging {  
 # Log file must be an absolute path.  
  file /var/log/aerospike/aerospike.log {  
 context any info  
  }  
}  
  
network {  
 service {  
 address any  
    port 3000  
  }  
  
 heartbeat {  
 mode multicast  
    multicast-group 239.1.99.222  
    port 9918  
    #       mode mesh  
    #       address 10.11.40.97  
    #       port 3002  
    #       mesh-seed-address-port 10.11.40.97 3002  
    #       mesh-seed-address-port 10.11.40.101 3002  
  
    # To use unicast-mesh heartbeats, remove the 3 lines above, and see  
    # aerospike\_mesh.conf for alternative.  
    protocol v3  
    interval 150  
    timeout 10  
  }  
  
 fabric {  
 port 3001  
    # 与相邻的节点之间建立的sockets数，用来平衡期间的数据迁移  
    channel-bulk-fds 4  
    # 处理sockets接受到的消息 的线程数  
    channel-bulk-recv-threads 8  
    channel-meta-fds 2  
    channel-meta-recv-threads 8  
    channel-rw-fds 8  
    channel-rw-recv-threads 16  
    keepalive-enabled true  
    send-threads 8  
  }  
  
 info {  
 port 3003  
  }  
}  
  
  
  
namespace user {  
 replication-factor 2  
  memory-size 35G  
  high-water-memory-pct 95  
  high-water-disk-pct 95  
  stop-writes-pct 95  
  partition-tree-locks 4  
  partition-tree-sprigs 32  
  high-water-memory-pct 50  
  high-water-disk-pct 50  
  # 如果设置为true 将不允许一条记录有多个列  
  single-bin false  
  # 记录冲突的解决方案使用 last-update-time ,这里有两个选项分别是last-update-time和generation  
  conflict-resolution-policy last-update-time  
  
  default-ttl 0  
  
  #storage-engine memory  
  # To use file storage backing, comment out the line above and use the  
  # following lines instead.  
  storage-engine device {  
 #file /var/aerospike/data/user.dat  
    #filesize 136G  
    device /dev/vdb1  
    max-write-cache 128M  
    defrag-sleep 100  
    defrag-lwm-pct 70  
    data-in-memory false #tore data in memory in addition to file.  
    write-block-size 128K  
    post-write-queue 512  
  }  
}

### 客户端介绍：

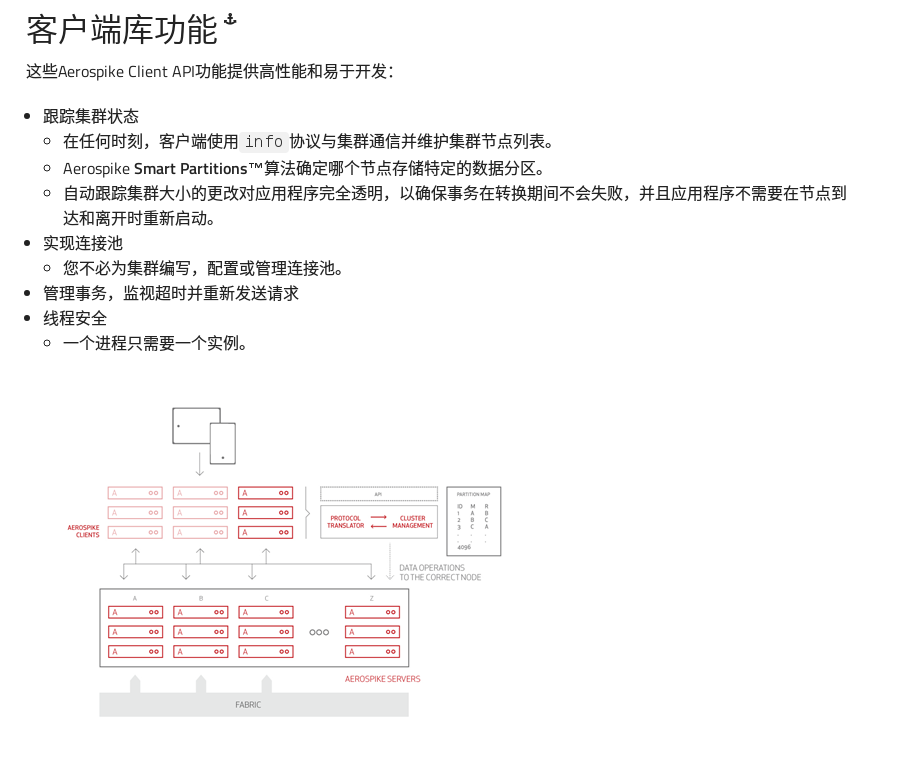
#### 客户端的架构：

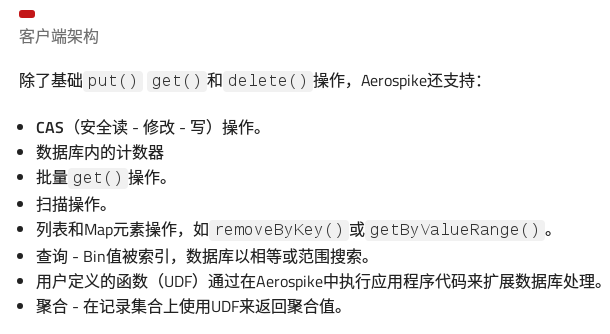
<http://www.aerospike.com/docs/architecture/clients.html>





还是装逼的图，我看的图是下面的：





#### 客户端的优化：

客户端提供了2个不同包下的客户端，分别是

com.aerospike.client.AerospikeClient

com.aerospike.client.async.AsyncClient

##### 创建客户端：

创建客户端很简单，构造函数只有2个参数，分别是

1，Policy对象

com.aerospike.client.async.AsyncClientPolicy (用于异步客户端的构造函数)

com.aerospike.client.policy.ClientPolicy (用于同步客户端的构造函数)

2，Host对象：

Host对象的构造如下。

public Host(String name, int port)；

So tm easy！！！

这里以异步客户端为例，简介其配置参数：

###### buildAsyncAerospikeClientPolicy

private AsyncClientPolicy buildAsyncAerospikeClientPolicy() {  
  
 int cpuNumber = Runtime.*getRuntime*().availableProcessors();  
// System.out.println("CPU核心数：" + cpuNumber);  
 AsyncClientPolicy asyncClientPolicy = new AsyncClientPolicy();  
 // 当达到最大并发命令数的时候，使用的策略。  
 // RTB现在用的是REJECT,这里尝试使用ACCEPT。  
 asyncClientPolicy.asyncMaxCommandAction = MaxCommandAction.*ACCEPT*;  
 // 每个as节点最多同时处理的命令数（client to server的连接数 = maxCommands × nodes）  
 asyncClientPolicy.asyncMaxCommands = cpuNumber \* 4;  
 // 异步网络选择器的超时时间，单位毫秒。0：不可以超时。  
 asyncClientPolicy.asyncSelectorTimeout = 10;  
 // 通常是单线程的，为了更好的发挥多核cpu的性能，可以根据cpu核心数调整。这里假设是4核cpu，设置为8.  
 asyncClientPolicy.asyncSelectorThreads = cpuNumber;  
 // 创建线程池让aerospikeClient使用，核心线程4个，最大同时运行8个线程，每个活动线程最大存活时间10毫秒，使用10000个元素的链表队列存储新增的线程。  
 ThreadPoolExecutor poolExecutor = new ThreadPoolExecutor(cpuNumber \* 2, cpuNumber \* 6, 10, TimeUnit.*MILLISECONDS*, new LinkedBlockingQueue<Runnable>(100000), Executors.*defaultThreadFactory*());  
  
 // 如果不设置线程池，maxCommands依旧使用1个线程工作。这里使用官方提供的一个可变大小的线程池例子  
 asyncClientPolicy.asyncTaskThreadPool = poolExecutor;  
// asyncClientPolicy.asyncTaskThreadPool = Executors.newCachedThreadPool(new ThreadFactory() {  
// public final Thread newThread(Runnable runnable) {  
// Thread thread = new Thread(runnable);  
// thread.setDaemon(true);  
// return thread;  
// }  
// });  
 // 设置read策略  
 asyncClientPolicy.asyncReadPolicyDefault = buildReadPolicy();  
 // 设置write策略  
 asyncClientPolicy.asyncWritePolicyDefault = buildWritePolicyServerConf();  
 // 设置scanPolicy  
 asyncClientPolicy.asyncScanPolicyDefault = buildScanPolicy();  
 // 设置queryPolicy  
 asyncClientPolicy.asyncQueryPolicyDefault = buildQueryPolicy();  
 // 设置异步批处理策略  
 asyncClientPolicy.asyncBatchPolicyDefault = buildBatchPolicy();  
 return asyncClientPolicy;  
 }

可以看到这里有很多Policy可以设置，具体怎么设置这些策略，以及其含义是什么？看下面的代码：

###### buildReadPolicy

private Policy buildReadPolicy() {  
 if (readPolicy != null) {  
 return readPolicy;  
 }  
 // 创建一个read 策略  
 readPolicy = new Policy();  
 readPolicy.consistencyLevel = ConsistencyLevel.*CONSISTENCY\_ONE*;  
 // 不重试  
 readPolicy.maxRetries = 0;  
 // 重试的休息时间  
 readPolicy.sleepBetweenRetries = 0;  
 // 使用默认值，读取的目标副本,写操作不收这个参数的影响。  
 readPolicy.replica = Replica.*MASTER*;  
 // 使用默认值，超时时间0  
 readPolicy.timeout = READ\_TIMEOUT;  
 // 超时之前关闭套接字的时间，使用默认值0.  
 readPolicy.timeoutDelay = 0;  
 // 超时后，不重试，这是默认值  
 readPolicy.retryOnTimeout = READ\_RETRY;  
 // 发送用户定义的key,这里使用默认值。  
 readPolicy.sendKey = false;  
 return readPolicy;  
}

对于WritePolicy我这里写了3个方法，根据具体需要分别在执行client.put命令时使用。

###### buildWritePolicyServerConf

private WritePolicy buildWritePolicyServerConf() {  
 if (writePolicyServerConf != null)  
 return writePolicyServerConf;  
 writePolicyServerConf = buildCommonWritePolicy();  
 writePolicyServerConf.expiration = ExpireType.*USE\_SERVER\_CONF*.getType();  
 return writePolicyServerConf;  
}

###### buildWritePolicyNerverExpire

private WritePolicy buildWritePolicyNerverExpire() {  
 if (writePolicyNerverExpire != null)  
 return writePolicyNerverExpire;  
 writePolicyNerverExpire = buildCommonWritePolicy();  
 // 设置记录的TTL(过期时间)， 0：使用服务器的配置. -1:永不过期. -2：不改变TTL的同时,更新数据(服务端版本要大于3.10.1).  
 writePolicyNerverExpire.expiration = ExpireType.*NEVER\_EXPIRE*.getType();  
 return writePolicyNerverExpire;  
}

###### buildWritePolicyNotChangeExpire

private WritePolicy buildWritePolicyNotChangeExpire() {  
 if (writePolicyNotChangeExpire != null)  
 return writePolicyNotChangeExpire;  
 writePolicyNotChangeExpire = buildCommonWritePolicy();  
 // 设置记录的TTL(过期时间)， 0：使用服务器的配置. -1:永不过期. -2：不改变TTL的同时,更新数据(服务端版本要大于3.10.1).  
 writePolicyNotChangeExpire.expiration = ExpireType.*NOT\_CHANGE\_EXPIRE*.getType();  
 return writePolicyNotChangeExpire;  
}

###### buildCommonWritePolicy

private WritePolicy buildCommonWritePolicy() {  
 // read策略中read操作特有的属性不会被write使用。但是各种重试规则，超时规则可以重用。  
 WritePolicy commonWritePolicy = new WritePolicy(buildReadPolicy());  
 // 没有记录就创建，有就更新。  
 commonWritePolicy.recordExistsAction = RecordExistsAction.*UPDATE*;  
 // 不使用历史记录来限制写操作  
 commonWritePolicy.generationPolicy = GenerationPolicy.*NONE*;  
 // 仅提交到master副本，server在返回之前会创建异步的同步线程去同步所有副本，这样配置是为了加快写入速度，  
 commonWritePolicy.commitLevel = CommitLevel.*COMMIT\_MASTER*;  
 // 我们不使用历史记录限制写操作，所以每次都把写的数据记录为第一次写入。  
 commonWritePolicy.generation = 0;  
 // 设置是否要对每个操作都返回操作结果，作用是当执行clear操作的时候，默认是没有结果返回的，很难确定是否执行成功。  
 // 需要注意的是，当对map进行操作的时候，此字段会强制修改为true。  
 commonWritePolicy.respondAllOps = false;  
 // 单词tombstone不知道怎么理解，好像是备份的意思，  
 // 我的理解是，当事务的结果被删除后，被删掉的数据有可能还可以读取到，将这个字段设为true，会把持久的数据也删除掉，从而避免读取到被删除的数据。  
 // *TODO 不知道这里设置为true，会不会影响性能，或者增加响应时间，这里设为false。在频控的功能上也许会用到。* commonWritePolicy.durableDelete = false;  
 commonWritePolicy.timeout = WRITE\_TIMEOUT;  
 commonWritePolicy.retryOnTimeout = WRITE\_RETRY;  
 return commonWritePolicy;  
}

###### buildScanPolicy

private ScanPolicy buildScanPolicy() {  
 if (scanPolicy != null)  
 return scanPolicy;  
 // 扫描应该用不到吧。。但看到有些字段还是要设置一下的。  
 scanPolicy = new ScanPolicy();  
 // 扫描数据的百分比。0~100的值  
 scanPolicy.scanPercent = 100;  
 // 用来指定使用多少个节点执行scan，0表示全部节点。  
 scanPolicy.maxConcurrentNodes = 0;  
 // 设置socket超时时间为30s  
 scanPolicy.socketTimeout = 30000;  
 scanPolicy.timeout = SCAN\_TIMEOUT;  
 scanPolicy.retryOnTimeout = SCAN\_RETRY;  
 // 设置启用并行scan  
 scanPolicy.concurrentNodes = true;  
 // 设置是否取回bin数据。  
 scanPolicy.includeBinData = true;  
 // 是否取回大数据类型的value。false，不返回，true，返回。  
 scanPolicy.includeLDT = false;  
 // *TODO 当集群波动较大时，终止scan* scanPolicy.failOnClusterChange = true;  
 return scanPolicy;  
}

###### buildQueryPolicy

private QueryPolicy buildQueryPolicy() {  
 QueryPolicy queryPolicy = new QueryPolicy();  
 // 设置使用的最大节点数，0表示所有节点。  
 queryPolicy.maxConcurrentNodes = 1;  
 // *TODO 一条记录被多个线程扫描，会把这个数据放到本地的一个队列中。如果队列满了，将会阻塞，直到数据被消费。* queryPolicy.recordQueueSize = 10000;  
 queryPolicy.timeout = READ\_TIMEOUT;  
 queryPolicy.retryOnTimeout = READ\_RETRY;  
 return queryPolicy;  
}

###### buildBatchPolicy

private BatchPolicy buildBatchPolicy() {  
 if (batchPolicy != null)  
 return batchPolicy;  
 batchPolicy = new BatchPolicy();  
 // 性能更好，详细的看k字段说明吧。。  
 batchPolicy.maxConcurrentThreads = 0;  
 // 使用内联  
 batchPolicy.allowInline = true;  
 // 安全策略  
 batchPolicy.sendSetName = false;  
 batchPolicy.timeout = BATCH\_TIMEOUT;  
 batchPolicy.retryOnTimeout = BATCH\_RETRY;  
 return batchPolicy;  
}

##### 使用客户端：

好的，客户端创建完了，我们看一些常用操作的代码：

###### getByAsync(批量)

*/\*\*  
 \* 使用异步的方式，查询多个key，  
 \*  
 \** ***@param*** *keys 异步查询的key  
 \** ***@param*** *binNames 查询的bin name  
 \** ***@return*** *返回查到的所有记录  
 \*/*private Record[] getByAsync(Key[] keys, String... binNames) {  
 return asyncClient.get(batchPolicy, keys, binNames);  
}

###### getByAsync(单个)

*/\*\*  
 \* 使用异步的方式查询一个key  
 \*  
 \** ***@param*** *key 需要查询的key  
 \** ***@param*** *binNames 需要查询的bin name  
 \** ***@return*** *返回查询的记录，没有查到返回null  
 \*/*public Record getByAsync(Key key, String binNames) {  
 return asyncClient.get(readPolicy, key, binNames);  
}

###### getBySync

private Record[] getBySync(Key[] keys, String... binNames) {  
 return syncClient.get(batchPolicy, keys, binNames);  
}

###### getBySync

public Record getBySync(Key key, String binNames) {  
 return syncClient.get(readPolicy, key, binNames);  
}

因为使用的Policy不同，所以这里异步和同步分别做了2个get方法。

###### putByAsync

public void putByAsync(ExpireType expire, Key key, Bin... bins) throws AerospikeException {  
 if (expire.getType() == ExpireType.*NEVER\_EXPIRE*.getType()) {  
 asyncClient.put(buildWritePolicyNerverExpire(), key, bins);  
  
 } else if (expire.getType() == ExpireType.*NOT\_CHANGE\_EXPIRE*.getType()) {  
 asyncClient.put(buildWritePolicyNotChangeExpire(), key, bins);  
  
 } else {  
 asyncClient.put(buildWritePolicyServerConf(), key, bins);  
 }  
}

### 知识补充：

#### 基准测试

基准测试 （Benchmark Test, 即BMT）

基准测试是指通过设计科学的测试方法、测试工具和测试系统，实现对一类测试对象的某项性能指标进行定量的和可对比的测试。例如，对计算机CPU进行[浮点运算](http://baike.baidu.com/item/%E6%B5%AE%E7%82%B9%E8%BF%90%E7%AE%97" \t "/home/hamlin/文档\\x/_blank)、数据访问的带宽和延迟等指标的基准测试，可以使用户清楚地了解每一款CPU的运算性能及作业吞吐能力是否满足应用程序的要求；再如对[数据库管理系统](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "/home/hamlin/文档\\x/_blank)的ACID（Atomicity, Consistency, Isolation, Durability, 原子性、一致性、独立性和持久性）、查询时间和[联机事务处理](http://baike.baidu.com/item/%E8%81%94%E6%9C%BA%E4%BA%8B%E5%8A%A1%E5%A4%84%E7%90%86" \t "/home/hamlin/文档\\x/_blank)能力等方面的性能指标进行基准测试，也有助于使用者挑选最符合自己需求的[数据库系统](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "/home/hamlin/文档\\x/_blank)。从以上两个例子我们可以看出，可测量、可重复、可对比是基准测试的三大原则，其中可测量是指测试的输入和输出之间是可达的，也就是[测试过程](http://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E8%BF%87%E7%A8%8B" \t "/home/hamlin/文档\\x/_blank)是可以实现的，并且测试的结果可以量化表现；可重复是指按照测试过程实现的结果是相同的或处于可接受的[置信区间](http://baike.baidu.com/item/%E7%BD%AE%E4%BF%A1%E5%8C%BA%E9%97%B4/7442583" \t "/home/hamlin/文档\\x/_blank)之内，而不受测试的时间、地点和执行者的影响；可对比是指一类测试对象的测试结果具有[线性关系](http://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%85%B3%E7%B3%BB/1653156" \t "/home/hamlin/文档\\x/_blank)，测试结果的大小直接决定性能的高低。

对于可[再现性](http://baike.baidu.com/item/%E5%86%8D%E7%8E%B0%E6%80%A7/9225956" \t "/home/hamlin/文档\\x/_blank)，基准测试是最好的方法。

在开发阶段前期，应该使用基准测试来确定应用程序中是否出现性能倒退。基准测试可以在一个相对短的时间内收集可重复的结果。进行基准测试的最好方法是，每次测试改变一个且只改变一个参数。例如，如果想知道增加[JVM](http://baike.baidu.com/item/JVM" \t "/home/hamlin/文档\\x/_blank)内存是否会影响应用程序的性能，就逐次递增JVM内存（例如，从1024 MB增至1224 MB，然后是1524 MB，最后是2024 MB），在每个阶段收集结果和环境数据，记录信息，然后转到下一阶段。这样在分析测试结果时就有迹可循。

基准测试的关键是要获得一致的、可再现的结果。可再现的结果有两个好处：减少重新运行测试的次数；对测试的产品和产生的数字更为确信。使用的[性能测试](http://baike.baidu.com/item/%E6%80%A7%E8%83%BD%E6%B5%8B%E8%AF%95" \t "/home/hamlin/文档\\x/_blank)工具可能会对测试结果产生很大影响。假定测试的两个指标是服务器的响应时间和吞吐量，它们会受到服务器上的负载的影响。服务器上的负载受两个因素影响：同时与服务器通信的连接（或虚拟用户）的数目，以及每个虚拟用户请求之间的考虑时间的长短。很明显，与服务器通信的用户越多，负载就越大。同样，请求之间的考虑时间越短，负载也越大。这两个因素的不同组合会产生不同的服务器负载等级。记住，随着服务器上负载的增加，吞吐量会不断攀升，直到到达一个点。

当软件系统中增加了1个新模块，此时需要做基准测试，以判断新的模块对整个软件系统的性能影响。

### 讨论：

#### 分布式和集群

Aerospike开头我们就说了，根据官方说明aerospike是分布式的。

真的吗？

我们来讨论一下？

下面是我整理的一些分布式和集群的区别。

这里可以讨论下分布式和集群的区别。

集群的概念：

###### 两大特性：

1，可扩展性，集群的性能不取决于单一的实例，新的实例可以动态地加入到集群中，从而增强集群的性能。

2，高可用性－－在集群中，同样的服务可以由多个服务实体提供。如果一个服务实体失败了，另一个服务实体会接管失败的服务实体。集群提供了从一个出错的服务实体恢复到另一个服务实体的功能增强了应用的可用性。

我们当前的服务是这样吗？

###### 两大能力：

1，负载均衡－－负载均衡能把任务比较均衡地分布到集群环境中的资源上。

2，错误恢复－－由于某种原因，执行某个任务的资源出现故障，另一服务实体中执行同一任务的资源接着完成任务。这种由于一个实体中的资源不能工作，另一个实体中的资源透明的继续完成任务的过程叫错误恢复。

我们当前的服务是这样吗？

###### 两大技术

1，集群地址－－具有单一集群地址（nginx/HA服务器）是集群的一个基本特征。维护集群地址的设置被称为负载均衡器。负载均衡器内部负责管理各个服务实体的加入和退出，外部负责集群地址向内部服务实体地址的转换。当正在工作的服务实体发生故障时，负载均衡器把后来的任务转向另外一个服务实体。

2，内部通信－－为了能协同工作、实现负载均衡和错误恢复，集群各实体间必须时常通信，比如负载均衡器对服务实体心跳测试信息、服务实体间任务执行上下文信息的通信。

我们当前的服务是这样吗？

###### 分布式与集群的联系与区别

分布式是指将不同的业务分布在不同的地方。   
 而集群指的是将几台服务器集中在一起，实现同一业务。   
 分布式中的每一个节点，都可以做集群。   
 而集群并不一定就是分布式的。

而分布式，从窄意上理解，也跟集群差不多， 但是它的组织比较松散，不像集群，有一个组织性，一台服务器垮了，其它的服务器可以顶上来。   
 分布式的每一个节点，都完成不同的业务，一个节点垮了，哪这个业务就不可访问了。