

# Apache Kudu 1.4.0 中文文档

---

来源: [ApacheCN](#)

# Apache Kudu 1.4.0 中文文档

---

来源: [ApacheCN](#)

# 介绍 Kudu

---

原文链接：<http://kudu.apache.org/docs/index.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813605>

贡献者：那伊抹微笑，[ApacheCN](#)，[Apache中文网](#)

**Kudu** 是一个针对 **Apache Hadoop** 平台而开发的列式存储管理器。**Kudu** 共享 **Hadoop** 生态系统应用的常见技术特性：它在 **commodity hardware**（商品硬件）上运行，**horizontally scalable**（水平可扩展），并支持 **highly available**（高可用）性操作。

此外，**Kudu** 还有更多优化的特点：

- **OLAP** 工作的快速处理。
- 与 **MapReduce**，**Spark** 和其他 **Hadoop** 生态系统组件集成。
- 与 **Apache Impala**（**incubating**）紧密集成，使其与 **Apache Parquet** 一起使用 **HDFS** 成为一个很好的可变的替代方案。
- 强大而灵活的一致性模型，允许您根据每个 **per-request**（请求选择）一致性要求，包括 **strict-serializable**（严格可序列化）一致性的选项。
- 针对同时运行顺序和随机工作负载的情况性能很好。
- 使用 **Cloudera Manager** 轻松维护和管理。
- **High availability**（高可用性）。**Tablet server** 和 **Master** 使用 [Raft Consensus Algorithm](#) 来保证节点的高可用，确保只要有一

半以上的副本可用，该 **tablet** 便可用于读写。例如，如果 **3** 个副本中有 **2** 个或 **5** 个副本中的 **3** 个可用，则该 **tablet** 可用。即使在 **leader tablet** 出现故障的情况下，读取功能也可以通过 **read-only**（只读的）**follower tablets** 来进行服务。

- 结构化数据模型。

通过结合这些所有的特性，**Kudu** 的目标是支持应用家庭中那些难以在当前**Hadoop** 存储技术中实现的应用。**Kudu** 常见的几个应用场景：

- 实时更新的应用。刚刚到达的数据就马上要被终端用户使用访问到。
- 时间序列相关的应用，需要同时支持：
  - 根据海量历史数据查询。
  - 必须非常快地返回关于单个实体的细粒度查询。
- 实时预测模型的应用，支持根据所有历史数据周期地更新模型。
- 有关这些和其他方案的更多信息，请参阅 [Example Use Cases](#)。

## Kudu-Impala 集成特性

### CREATE/ALTER/DROP TABLE

**Impala** 支持使用 **Kudu** 作为持久层来 **creating**（创建），**altering**（修改）和 **dropping**（删除）表。这些表遵循与 **Impala** 中其他表格相同的 **Internal / external**（内部 / 外部）方法，允许灵活的数据采集和查询。

### INSERT

数据可以使用与那些使用 **HDFS** 或 **HBase** 持久性的任何其他 **Impala** 表相同的语法插入 **Impala** 中的 **Kudu** 表。

## UPDATE / DELETE

**Impala** 支持 **UPDATE** 和 **DELETE SQL** 命令逐行或批处理修改 **Kudu** 表中的已有的数据。选择 **SQL** 命令的语法与现有标准尽可能兼容。除了简单 **DELETE** 或 **UPDATE** 命令之外，还可以 **FROM** 在子查询中指定带有子句的复杂连接。

## Flexible Partitioning（灵活分区）

与 **Hive** 中的表分区类似，**Kudu** 允许您通过 **hash** 或范围动态预分割成预定义数量的 **tablets**，以便在集群中均匀分布写入和查询。您可以通过任意数量的 **primary key**（主键）列，任意数量的 **hashes** 和可选的 **list of split rows** 来进行分区。参见[模式设计](#)。

## Parallel Scan（并行扫描）

为了在现代硬件上实现最高的性能，**Impala** 使用的 **Kudu** 客户端可以跨多个 **tablets** 扫描。

## High-efficiency queries（高效查询）

在可能的情况下，**Impala** 将谓词评估下推到 **Kudu**，以便使谓词评估为尽可能接近数据。在许多任务中，查询性能与 **Parquet** 相当。

有关使用 **Impala** 查询存储在 **Kudu** 中的数据的更多详细信息，请参阅 **Impala** 文档。

## 概念和术语

### Columnar Data Store（列式数据存储）

**Kudu** 是一个 **columnar data store**（列式数据存储）。列式数据存

储在强类型列中。由于几个原因，通过适当的设计，**Kudu** 对 **analytical**（分析）或 **warehousing**（数据仓库）工作会非常出色。

## Read Efficiency（高效读取）

对于分析查询，允许读取单个列或该列的一部分同时忽略其他列，这意味着您可以在磁盘上读取更少块来完成查询。与基于行的存储相比，即使只返回几列的值，仍需要读取整行数据。

## Data Compression（数据压缩）

由于给定的列只包含一种类型的数据，基于模式的压缩比压缩混合数据类型（在基于行的解决方案中使用）时更有效几个数量级。结合从列读取数据的效率，压缩允许您在从磁盘读取更少的块时完成查询。请参阅 [数据压缩](#)

## Table（表）

一张 **table** 是数据存储在 **Kudu** 的位置。表具有 **schema** 和全局有序的 **primary key**（主键）。**table** 被分成称为 **tablets** 的 **segments**。

## Tablet

一个 **tablet** 是一张 **table** 连续的 **segment**，与其它数据存储引擎或关系型数据库中的 **partition**（分区）相似。给定的 **tablet** 冗余到多个 **tablet** 服务器上，并且在任何给定的时间点，其中一个副本被认为是 **leader tablet**。任何副本都可以对读取进行服务，并且写入时需要在为 **tablet** 服务的一组 **tablet server** 之间达成一致。

## Tablet Server

一个 **tablet server** 存储 **tablet** 和为 **tablet** 向 **client** 提供服务。对于给定的 **tablet**，一个 **tablet server** 充当 **leader**，其他 **tablet server** 充当该 **tablet** 的 **follower** 副本。只有 **leader** 服务写请求，然而 **leader** 或 **followers** 为每个服务提供读请求。**leader** 使用 [Raft Consensus Algorithm](#) 来进行选举。一个 **tablet server** 可以服务多个 **tablets**，并且一个 **tablet** 可以被多个 **tablet servers** 服务着。

## Master

该 **master** 保持跟踪所有的 **tablets**，**tablet servers**，[Catalog Table](#) 和其它与集群相关的 **metadata**。在给定的时间点，只能有一个起作用的 **master**（也就是 **leader**）。如果当前的 **leader** 消失，则选举出一个新的 **master**，使用 [Raft Consensus Algorithm](#) 来进行选举。

**master** 还协调客户端的 **metadata operations**（元数据操作）。例如，当创建新表时，客户端内部将请求发送给 **master**。**master** 将新表的元数据写入 **catalog table**，并协调在 **tablet server** 上创建 **tablet** 的过程。

所有 **master** 的数据都存储在一个 **tablet** 中，可以复制到所有其他候选的 **master**。

**tablet server** 以设定的间隔向 **master** 发出心跳（默认值为每秒一次）。

## Raft Consensus Algorithm

**Kudu** 使用 [Raft consensus algorithm](#) 作为确保常规 **tablet** 和 **master** 数据的容错性和一致性的手段。通过 **Raft**, **tablet** 的多个副本选举出 **leader**, 它负责接受以及复制到 **follower** 副本的写入。一旦写入的数据在大多数副本中持久化后, 就会向客户确认。给定的一组  $N$  副本 (通常为 **3** 或 **5** 个) 能够接受最多  $(N - 1)/2$  错误的副本的写入。

## Catalog Table (目录表)

**catalog table** 是 **Kudu** 的 **metadata** (元数据) 的中心位置。它存储有关 **tables** 和 **tablets** 的信息。该 **catalog table** (目录表) 可能不会被直接读取或写入。相反, 它只能通过客户端 **API** 中公开的元数据操作访问。**catalog table** 存储两类元数据。

## Tables

**table schemas, locations, and states** (表结构, 位置 和状态)

## Tablets

现有 **tablet** 的列表, 每个 **tablet** 的副本所在哪些 **tablet server**, **tablet** 的当前状态以及开始和结束的 **keys** (键)。

## Logical Replication (逻辑复制)

**Kudu** 复制操作, 不是磁盘上的数据。这被称为 **logical replication** (逻辑复制), 而不是 **physical replication** (物理复制)。这有几个优点:

- 虽然 **insert** (插入) 和 **update** (更新) 确实通过网络传输数



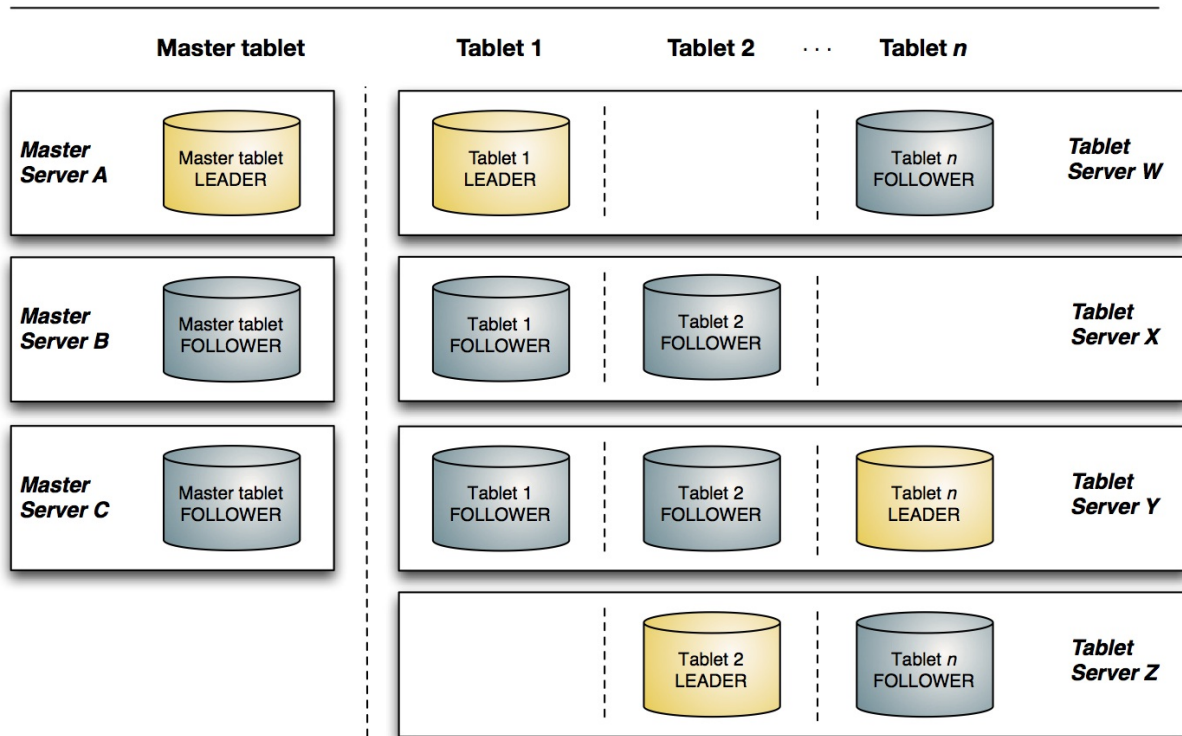
据，**deletes**（删除）不需要移动任何数据。**delete**（删除）操作被发送到每个 **tablet server**，它在本地执行删除。

- 物理操作，如 **compaction**，不需要通过 **Kudu** 的网络传输数据。这与使用 **HDFS** 的存储系统不同，其中 **blocks**（块）需要通过网络传输以满足所需数量的副本。
- **tablet** 不需要在同一时间或相同的时间表上执行压缩，或者在物理存储层上保持同步。这会减少由于压缩或大量写入负载而导致所有 **tablet server** 同时遇到高延迟的机会。

## 架构概述

下图显示了一个具有三个 **master** 和多个 **tablet server** 的 **Kudu** 集群，每个服务器都支持多个 **tablet**。它说明了如何使用 **Raft** 共识来允许 **master** 和 **tablet server** 的 **leader** 和 **follower**。此外，**tablet server** 可以成为某些 **tablet** 的 **leader**，也可以是其他 **tablet** 的 **follower**。**leader** 以金色显示，而 **follower** 则显示为蓝色。

## Kudu network architecture



## 案例示例

**Streaming Input with Near Real Time Availability**（具有近实时可用性的流输入）

数据分析中的一个共同挑战就是新数据快速而不断地到达，同样的数据需要靠近实时的读取，扫描和更新。Kudu 通过高效的列式扫描提供了快速插入和更新的强大组合，从而在单个存储层上实现了实时分析用例。

**Time-series application with widely varying access patterns**（具有广泛变化的访问模式的时间序列应用）

**time-series**（时间序列）模式是根据其发生时间组织和键入数据点的

模式。这可以用于随着时间的推移调查指标的性能，或者根据过去的数  
据尝试预测未来的行为。例如，时间序列的客户数据可以用于存储购买  
点击流历史并预测未来的购买，或由客户支持代表使用。虽然这些不同  
类型的分析正在发生，插入和更换也可能单独和批量地发生，并且立即  
可用于读取工作负载。**Kudu** 可以用 **scalable**（可扩展）和 **efficient**  
（高效的）方式同时处理所有这些访问模式。由于一些原因，**Kudu** 非  
常适合时间序列的工作负载。随着 **Kudu** 对基于 **hash** 的分区的支持，  
结合其对复合 **row keys**（行键）的本地支持，将许多服务器上的表设  
置成很简单，而不会在使用范围分区时通常观察到“**hotspotting**（热  
点）”的风险。**Kudu** 的列式存储引擎在这种情况下也是有益的，因为许  
多时间序列工作负载只读取了几列，而不是整行。过去，您可能需要  
使用多个数据存储来处理不同的数据访问模式。这种做法增加了应用程  
序和操作的复杂性，并重复了数据，使所需存储量增加了一倍（或更  
糟）。**Kudu** 可以本地和高效地处理所有这些访问模式，而无需将工作  
卸载到其他数据存储。

**Predictive Modeling**（预测建模） 数据科学家经常从大量数据中开发  
预测学习模型。模型和数据可能需要在发生学习或随着建模情况的变  
化而经常更新或修改。此外，科学家可能想改变模型中的一个或多个因  
素，看看随着时间的推移会发生什么。在 **HDFS** 中更新存储在文件中的  
大量数据是资源密集型的，因为每个文件需要被完全重写。在  
**Kudu**，更新发生在近乎实时。科学家可以调整值，重新运行查询，并  
以秒或分钟而不是几小时或几天刷新图形。此外，批处理或增量算法可  
以随时在数据上运行，具有接近实时的结果。

**Combining Data In Kudu With Legacy Systems**（结合 **Kudu** 与遗  
留系统的数据）

公司从多个来源生成数据并将其存储在各种系统和格式中。例如，您  
的一些数据可能存储在 **Kudu**，一些在传统的 **RDBMS** 中，一些在  
**HDFS** 中的文件中。您可以使用 **Impala** 访问和查询所有这些源和格

式，而无需更改旧版系统。

## 下一步

- [Kudu 入门指南](#)
- [安装指南](#)

# Kudu 入门指南

原文链接：<http://kudu.apache.org/docs/quickstart.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813610>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## Apache Kudu 快速开始

按照这些说明设置和运行 **Kudu VM**，并在几分钟之内从 **Kudu**，**Kudu\_Impala** 和 **CDH** 开始。

## 获得 Kudu 快速开始 VM

先决条件

1. 安装 **Oracle Virtualbox**。**VM** 已经通过测试，可以在 **Ubuntu 14.04** 上使用 **VirtualBox 4.3** 版本，并在 **OSX 10.9** 上使用 **VirtualBox 5** 的版本。**VirtualBox** 也包含在大多数软件包管理器中：**apt-get**，**brew** 等。
2. 安装完成后，使用 **which VBoxManage** 命令确保 **VBoxManage** 在您的 **PATH** 中。

安装

要下载并启动 **VM**，请在终端窗口中执行以下命令。

```
$ curl -s https://raw.githubusercontent.com/cloudera/kudu-examples
```

此命令下载一个 **shell** 脚本，这个脚本克隆了 **kudu-example Git** 仓库，然后将大约 **1.2 GB** 大小的 **VM** 映像下载到当前的工作目录中。您可以在此脚本下载下来之后检查这个脚本通过删除上面脚本的 **| bash** 部分。安装完成之后，您可以通过 **SSH** 连接到 **guest** 虚拟机来验证一切是否正常：

```
$ ssh demo@quickstart.cloudera
```

上面的 **username** 和 **password** 都是演示。此外，演示用户具有无密码的 **sudo** 权限，以便您可以安装其他软件和管理 **guest OS**。您还可以访问 **kudu-examples** 作为 **/home/demo/kudu-examples/** 中的共享文件夹，或者在 **host** 上的 **VirtualBox** 共享文件夹位置。这是使脚本或数据对 **guest** 可见的一种快捷方式。

你可以通过执行以下命令快速验证 **Kudu** 和 **Impala** 是否正在运行：

```
$ ps aux | grep kudu  
$ ps aux | grep impalad
```

如果连接到 **VM** 或其中一个进程没有运行时出现问题，请务必参阅“[疑难解答](#)”部分。

## 加载数据

要使用 **Kudu** 和 **Impala** 进行一些典型的操作，我们将使用 **San Francisco MTA GPS dataset**。该数据集包含从 **SF MTA** 车队的总线上安装的传感器周期性传输的原始位置数据。

### 1. 下载示例数据并将其加载到 **HDFS** 中

首先我们将下载示例数据集，准备并将其上传到 **HDFS** 集群中。

**SF MTA** 的站点通常有点慢，所以我们从数据集中镜像了一个 **CSV**

文件样本，网址为 <http://kudu-sample-data.s3.amazonaws.com/sfmtaAVLRawData01012013.csv.gz>  
原始数据集使用 **DOS** 类型的行结尾，因此我们将在上传过程中使用 **tr** 将其转换为 **UNIX** 风格。

```
$ wget http://kudu-sample-data.s3.amazonaws.com/sfmtaAVLRawData01012013.csv.gz
$ hdfs dfs -mkdir /sfmta
$ zcat sfmtaAVLRawData01012013.csv.gz | tr -d '\r' | hadoop fs -put - /sfmta
```

2. 创建一个新的外部 **Impala** 表以访问纯文本数据。要在虚拟机中连接到 **Impala**，请执行以下命令：

```
ssh demo@quickstart.cloudera -t impala-shell
```

现在，您可以执行以下命令：

```
CREATE EXTERNAL TABLE sfmta_raw (
  revision int,
  report_time string,
  vehicle_tag int,
  longitude float,
  latitude float,
  speed float,
  heading float
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/sfmta/'
TBLPROPERTIES ('skip.header.line.count'='1');
```

3. 验证数据是否实际加载运行以下命令：

```
SELECT count(*) FROM sfmta_raw;
```

```
+-----+
| count(*) |
+-----+
| 859086   |
```

```
+-----+
```

4. 接下来，我们将创建一个 **Kudu** 表并加载数据。请注意，我们将字符串 **report\_time** 字段转换为 **unix** 风格的时间戳，以便更有效的存储。

```
CREATE TABLE sfmta
PRIMARY KEY (report_time, vehicle_tag)
PARTITION BY HASH(report_time) PARTITIONS 8
STORED AS KUDU
AS SELECT
  UNIX_TIMESTAMP(report_time, 'MM/dd/yyyy HH:mm:ss') AS report_time,
  vehicle_tag,
  longitude,
  latitude,
  speed,
  heading
FROM sfmta_raw;
```

```
+-----+
| summary                |
+-----+
| Inserted 859086 row(s) |
+-----+
| Fetched 1 row(s) in 5.75s |
+-----+
```

创建的表使用复合主键。有关 **Impala** 的扩展 **SQL** 语法的更详细的介绍，请参阅 [Kudu Impala 集成](#)。

## 读取和修改数据

现在数据存储在 **Kudu** 中，您可以对其执行查询。以下查询查找包含最高记录车速的数据点。

```
SELECT * FROM sfmta ORDER BY speed DESC LIMIT 1;
```

```
+-----+-----+-----+-----+
| report_time | vehicle_tag | longitude | latitude |
+-----+-----+-----+-----+
```



1357022342	5411	-122.3968811035156	37.766658782958
------------	------	--------------------	-----------------

通过快速的 **Google** 搜索，我们可以看到这辆公共汽车在 **68MPH** 的 **16** 街上东行。乍一看，这似乎不太可能是真实的。也许我们做一些研究，发现这辆公共汽车的传感器设备被打破了，我们决定删除数据。使用 **Kudu** 和 标准 **SQL** 这是非常容易纠正的：

```
DELETE FROM sfmta WHERE vehicle_tag = '5411';  
  
-- Modified 1169 row(s), 0 row error(s) in 0.25s
```

## 下一步

以上示例显示了如何使用 **Impala** 和 **Kudu** 加载，查询和变更静态数据集。然而，**Kudu** 的真正实力是以流媒体方式吸收和突变数据的能力。

作为学习 **Kudu** 编程 **API** 的练习，请尝试实现一个使用 **SFMTA XML** 数据 **Feed** 的程序，将该相同的数据集实时摄取到 **Kudu** 表中。

## 故障排除

### 通过 **SSH** 访问 **VM** 的问题

- 确保主机已安装 **SSH** 客户端。
- 确保 **VM** 正在运行，通过运行以下命令并检查名为 **kudu-demo** 的虚拟机：

```
$ VBoxManage list runningvms
```

- 验证 **VM** 的 **IP** 地址是否包含在主机的 **/etc/hosts** 文件中。您应该看到一行包含一个 **IP** 地址，后跟主机名 **quickstart.cloudera**。要

检查正在运行的 **VM** 的 **IP** 地址，请使用下面的 **VBoxManage** 命令。

```
$ VBoxManage guestproperty get kudu-demo /VirtualBox/GuestInfo/Value: 192.168.56.100
```

- 如果您以前使用过 **Cloudera QuickStart VM**，则 **.ssh/known\_hosts** 文件可能包含对先前 **VM** 的 **SSH** 凭据的引用。从此文件删除对 **quickstart.cloudera** 的任何引用。

在 **VirtualBox** 中运行时缺少 **SSE4.2** 支持

- 运行 **Kudu** 目前需要一个支持 **SSE4.2** ( **Nehalem** 或者更高版本的 **Intel** ) 的 **CPU**。要通过 **SSE4.2** 支持进入 **guest** 虚拟机，请参阅 [VirtualBox 文档](#)。

## 下一步

- 安装 **Kudu**
- 配置 **Kudu**

# 安装指南

---

原文链接 :<http://kudu.apache.org/docs/installation.html>

译文链接 : <http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813613>

贡献者 : 小瑶 [ApacheCN](#) [Apache](#) 中文网

## 安装 **Apache Kudu**

您可以使用软件包在集群上部署 **Kudu** , 也可以从源代码构建 **Kudu** 。要运行 **Kudu** 而不安装任何东西, 请使用 **Kudu QuickStart VM** 。

注意

**Kudu** 目前更容易使用 **Cloudera Manager 5.4.7** 或更高版本进行安装和管理。如果您使用 **Cloudera Manager** , 请参阅 **Cloudera** 的 **Kudu** 文档。

## 升级 **Kudu**

要从以前的版本升级 **Kudu** , 请参阅从以前版本的 **Kudu** 升级。

## 先决条件和要求

硬件

- 一个或者多个主机运行 **Kudu masters**。建议使用一个主站 ( **master** ) ( 无容错 ), 三个 **masters** ( 可以容忍一个故障 ) 或 五个 **masters** ( 可以容忍两个故障 )。

- 运行 **Kudu tablet servers** 的一个或多个主机。使用副本时，至少需要三个 **tablet servers** 。

## 操作系统要求

### Linux

- **RHEL 6** , **RHEL 7** , **CentOS 6** , **CentOS 7** , **Ubuntu 14.04 (Trusty)** , **Ubuntu 16.04 (Xenial)** , **Debian 8 (Jessie)** , 或 **SLES 12** 。
- 支持 **hole punching** ( 打孔的 ) 内核和文件系统。打孔是使用 **FALLOC\_FL\_PUNCH\_HOLE** 选项设置的 **fallocate(2)** 系统调用。有关详细信息，请参阅 [故障排除 hole punching](#) 。
- **ntp** 。
- **xfs** 或者 **ext4** 格式化的驱动器。

### macOS

- **OS X 10.10 Yosemite** , **OS X 10.11 El Capitan** , 或 **macOS Sierra** 。
- 未提供预建的 **macOS** 包。

### Windows

- **Microsoft Windows** 不受支持。

### Storage ( 存储 )

- 如果固态存储可用，在这种高性能介质上存储 **Kudu WAL** 可以显著改善 **Kudu** 配置为其最高耐久性水平时的延迟。

## Management ( 管理 )

- 如果您使用 **Cloudera Manager** 和 **CDH**，则需要 **Cloudera Manager 5.4.3** 或更高版本。**Cloudera Manager 5.4.7** 及更新版本提供更好的监控和管理选项。

## 使用包安装

您可以使用操作系统管理的软件包来安装 **Kudu**。

表1.**Kudu** 包位置

OS ( 操作系统 )	Repository ( 存储库 )	Individual Packages ( 单个包 )
RHEL 或 CentOS	<a href="#">RHEL 6 or CentOS 6, RHEL 7 or CentOS 7</a>	<a href="#">RHEL 6 or CentOS 6, RHEL 7 or CentOS 7</a>
SLES	<a href="#">SLES 12</a>	<a href="#">SLES 12</a>
Ubuntu	<a href="#">Trusty, Xenial</a>	<a href="#">Trusty, Xenial</a>
Debian	<a href="#">Jessie</a>	<a href="#">Jessie</a>

安装在 **RHEL** 或 **CentOS** 主机上

1. 下载并配置您的操作系统的 **Kudu** 存储库，或使用 **Kudu Package Locations** 的相应链接手动下载各个 **RPM**。
2. 如果使用 **Yum** 存储库，请在将 **cloudera-kudu.repo** 文件保存到 **/etc/yum.repos.d/** 之后，使用以下命令在每个主机上安装 **Kudu** 软件包。

```
sudo yum install kudu # Base Kudu files
sudo yum install kudu-master # Kudu master in
sudo yum install kudu-tserver # Kudu tablet ser
sudo yum install kudu-client0 # Kudu C++ clien
sudo yum install kudu-client-devel # Kudu C++ clien
```

3. 要手动安装 **Kudu RPM**，首先下载它们，然后使用命令 **sudo rpm -ivh <RPM安装>** 来安装。
4. 注意：**kudu-master** 和 **kudu-tserver** 软件包仅在分别有 **master** 或 **tserver** 的主机上需要 ( 如果使用 **Cloudera Manager** 则完全不必要 )。每个提供配置文件和一个 **init.d** 脚本来管理相应的 **Kudu** 进程。一旦安装，**Kudu** 进程将在主机启动并关闭时自动启动和停止。

安装在 **SLES** 主机上

1. 下载并配置您的操作系统的 **Kudu** 存储库，或使用 **Kudu Package Locations** 的相应链接手动下载各个 **RPM**。
2. 如果使用 **Zypper** 存储库，请在将 **cloudera-kudu.repo** 文件保存到 **/etc/zypper/repos.d** 之后，使用以下命令在每个主机上安装 **Kudu** 软件包。

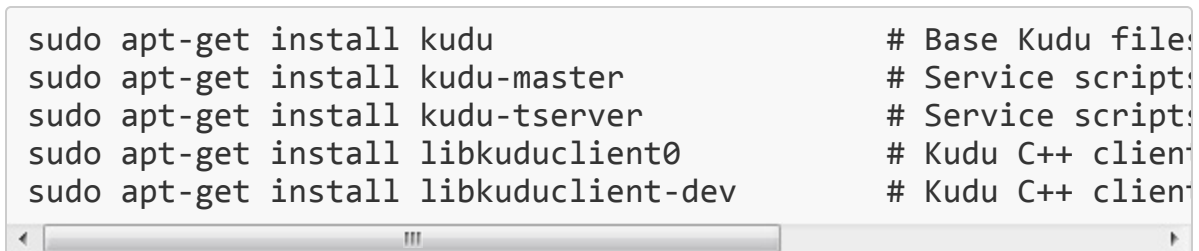
```
sudo zypper install kudu # Base Kudu files
sudo zypper install kudu-master # Kudu master in
sudo zypper install kudu-tserver # Kudu tablet ser
sudo zypper install kudu-client0 # Kudu C++ clien
sudo zypper install kudu-client-devel # Kudu C++ clien
```

3. 要手动安装 **Kudu RPM**，首先下载它们，然后使用命令 **sudo rpm -ivh <RPM安装>** 来安装。

4. 注意: **kudu-master** 和 **kudu-tserver** 软件包仅在分别有 **master** 或 **tserver** 的主机上需要 ( 如果使用 **Cloudera Manager** 则完全不必要 )。每个提供配置文件和一个 **init.d** 脚本来管理相应的 **Kudu** 进程。一旦安装, **Kudu** 进程将在主机启动并关闭时自动启动和停止。

安装在 **Ubuntu** 或 **Debian** 主机上

1. 如果使用 **ubuntu** 或 **debian** 存储库, 请在将 **cloudera.list** 文件保存到 **/etc/apt/sources.list.d/** 之后, 使用以下命令在每个主机上安装 **Kudu** 软件包。



```
sudo apt-get install kudu # Base Kudu files
sudo apt-get install kudu-master # Service scripts
sudo apt-get install kudu-tserver # Service scripts
sudo apt-get install libkuduclient0 # Kudu C++ client
sudo apt-get install libkuduclient-dev # Kudu C++ client
```

2. 要手动安装 **Kudu RPM**, 首先下载它们, 然后使用命令 **sudo rpm -ivh <RPM安装>** 来安装。
3. 注意: **kudu-master** 和 **kudu-tserver** 软件包仅在分别有 **master** 或 **tserver** 的主机上需要 ( 如果使用 **Cloudera Manager** 则完全不必要 )。每个提供配置文件和一个 **init.d** 脚本来管理相应的 **Kudu** 进程。一旦安装, **Kudu** 进程将在主机启动并关闭时自动启动和停止。

验证安装

1. 使用以下方法之一验证服务是否正在运行:
  - \* 在服务器上检查 **ps** 命令的输出, 以验证 **kudu-master** 或 **kudu-**

**tserver** 进程之一或两者正在运行。

\* 打开 **Master** 或者 **Tablet Server web UI**，方法是打开 **http://<\_host\_name>:8051/ for master** 或者 **http://<\_host\_name>:8050/ for tablet servers**。

2. 如果 **Kudu** 没有运行，请查看 **'/var/log/kudu'** 中的日志文件，如果有一个以 **'.FATAL'** 结尾的文件，那意味着 **Kudu** 无法启动。

\* 如果 **error** 是 **" Error during hole punch test"**，可能是 [您的操作系统的问题](#)。

\* 如果 **error** 是 **" Couldn't get the current time "**，这是 [ntp 的问题](#)。

\* 如果这是其他似乎不明显的东西，或者如果您没有运气尝试上述解决方案，您可以在 [用户邮件列表](#) 中寻求帮助。

## 必需配置

在启动 **Kudu** 服务之前，需要在每个主机上执行其他配置步骤。

1. 这些包在操作系统的备用数据库中创建一个 **kudu-conf** 条目，并且它们运送内置的 **conf.dist** 替代。要调整配置，您可以直接编辑 **/etc/kudu/conf/** 中的文件，或者使用操作系统实用程序创建一个新的选项，确保它是 **/etc/kudu/conf/** 指向的链接，在那里创建自定义配置文件配置的某些部分也配置在 **/etc/default/kudu-master** 和 **/etc/default/kudu-tserver** 文件中。如果创建自定义配置文件，您应该包括或者复制这些配置选项。

查看配置，包括默认的 **WAL** 和数据目录位置，并根据您的要求进行调整。

2. 使用以下命令启动 **Kudu** 服务：

```
$ sudo service kudu-master start
```



```
$ sudo service kudu-tserver start
```

3. 要停止 **Kudu** 服务，请使用以下命令：

```
$ sudo service kudu-master stop  
$ sudo service kudu-tserver stop
```

4. 将 **Kudu** 服务配置为在服务器启动时自动启动，将其添加到默认运行级别：

```
$ sudo chkconfig kudu-master on           # RHEL / CentOS  
$ sudo chkconfig kudu-tserver on          # RHEL / CentOS  
  
$ sudo update-rc.d kudu-master defaults   # Debian / Ubuntu  
$ sudo update-rc.d kudu-tserver defaults  # Debian / Ubuntu
```

5. 有关 **Kudu** 服务的其他配置，请参阅 [配置 Kudu](#) 。

## 从源代码构建

如果使用包裹或者软件包安装 **Kudu** 不能提供所需的灵活性，则可以从源代码构建 **Kudu** 。您可以在任何支持的操作系统上从源代码构建。

注意

已知的构建问题

- 不可能在 **Microsoft Windows** 上构建 **Kudu** 。
- 需要一个 **C++ 11** 编译器 ( **GCC 4.8** ) 。

## RHEL 或者 CentOS

需要 **RHEL** 或者 **CentOS 6.6** 或更高版本从源代码构建 **Kudu** 。要建立在 **7.0** 以上的版本上，必须安装 **Red Hat Developer Toolset** ( 才能访问支持 **C++ 11** 的编译器 )。

1. 安装必备库( 如果没有安装 )。

```
$ sudo yum install autoconf automake cyrus-sasl-devel cyrus-sasl  
cyrus-sasl-plain gcc gcc-c++ gdb git krb5-server krb5-workstation  
make openssl-devel patch pkgconfig redhat-lsb-core rsync unzip
```

2. 如果建立在 **7.0** 以上的 **RHEL** 或 **CentOS** 上，请安装 **Red Hat Developer Toolset**

```
$ DTLS_RPM=rhsc1-devtoolset-3-epel-6-x86_64-1-2.noarch.rpm  
$ DTLS_RPM_URL=https://www.softwarecollections.org/repos/rhsc1/  
$ wget ${DTLS_RPM_URL} -O ${DTLS_RPM}  
$ sudo yum install -y scl-utils ${DTLS_RPM}  
$ sudo yum install -y devtoolset-3-toolchain
```

3. 可选：如果您计划构建文档，请安装其他包，包括 **ruby**。

```
$ sudo yum install doxygen gem graphviz ruby-devel zlib-devel
```

注意

如果建立在 **7.0** 以上的 **RHEL** 或者 **CentOS** 上，则可能需要使用 **rubygems** 替换 **gem** 包

4. 克隆 **Git** 存储库并切换到新的 **kudu** 目录

```
$ git clone https://github.com/apache/kudu  
$ cd kudu
```

5. 使用 **build-if-necessary.sh** 脚本构建任何缺少的第三方要求。不使用 **devtoolset** 会导致主机编译器似乎需要 **libatomic**，但找不到它。

```
$ build-support/enable_devtoolset.sh thirdparty/build-if-necessary
```

6. 使用上一步中安装的实用程序构建 **Kudu** 。为中间输出选择一个构建目录，除了 **Kudu** 目录本身之外，它可以在文件系统中的任何位置。请注意，**devtoolset** 仍然必须指定，否则您会得到 **cc1plus: 错误: unrecognized command line option "-std=c++11"** 。

```
mkdir -p build/release
cd build/release
../../build-support/enable_devtoolset.sh \
  ../../thirdparty/installed/common/bin/cmake \
  -DCMAKE_BUILD_TYPE=release \
  ../../
make -j4
```

7. 可选：安装 **Kudu** 二进制文件，库和标题。如果不通过 **DESTDIR** 环境变量指定安装目录，则默认为 **/usr/local/** 。

```
sudo make DESTDIR=/opt/kudu install
```

8. 可选：构建文档。注意：此命令构建不适合上传到 **Kudu** 网站的本地文档。

```
$ make docs
```

## 示例 1 . RHEL / CentOS 构建脚本

此脚本概述了在新安装的 **RHEL** 或 **CentOS** 主机上构建 **Kudu** 的过程，可用作自动部署场景的基础。它跳过上面标记为可选步骤的步骤。

```
#!/bin/bash
```

```
sudo yum -y install autoconf automake cyrus-sasl-devel cyrus-sasl-  
cyrus-sasl-plain gcc gcc-c++ gdb git krb5-server krb5-workstation  
make openssl-devel patch pkgconfig redhat-lsb-core rsync unzip v  
DTLS_RPM=rhsc1-devtoolset-3-epel-6-x86_64-1-2.noarch.rpm  
DTLS_RPM_URL=https://www.softwarecollections.org/repos/rhsc1/devto  
wget ${DTLS_RPM_URL} -O ${DTLS_RPM}  
sudo yum install -y scl-utils ${DTLS_RPM}  
sudo yum install -y devtoolset-3-toolchain  
git clone https://github.com/apache/kudu  
cd kudu  
build-support/enable_devtoolset.sh thirdparty/build-if-necessary.s  
mkdir -p build/release  
cd build/release  
../../build-support/enable_devtoolset.sh \  
    ../../thirdparty/installed/common/bin/cmake \  
    -DCMAKE_BUILD_TYPE=release \  
    ../../..  
make -j4
```

## Ubuntu 或者 Debian

### 1. 安装必备库 ( 如果它们没有安装 )

```
$ sudo apt-get install autoconf automake curl g++ gcc gdb git  
krb5-admin-server krb5-kdc krb5-user libkrb5-dev libsasl2-dev  
libsasl2-modules-gssapi-mit libssl-dev libtool lsb-release ma  
patch pkg-config python rsync unzip vim-common
```

### 2. 可选: 如果您计划构建文档, 请安装其他包, 包括 **ruby** 。

```
$ sudo apt-get install doxygen gem graphviz ruby-dev xsltproc
```

3. 克隆 **Git** 存储库并切换到新的 **kudu** 目录。

```
$ git clone https://github.com/apache/kudu  
$ cd kudu
```

4. 使用 **build-if-necessary.sh** 脚本构建任何缺少的第三方要求。

```
$ thirdparty/build-if-necessary.sh
```

5. 使用上一步中安装的实用程序构建 **Kudu**。为中间输出选择一个构建目录，除了 **kudu** 目录本身之外，它可以在文件系统中的任何位置。

```
mkdir -p build/release  
cd build/release  
../../thirdparty/installed/common/bin/cmake -DCMAKE_BUILD_TYPE=Release  
make -j4
```

6. 可选：安装 **Kudu** 二进制文件，库和标题。如果不通过 **DESTDIR** 环境变量指定安装目录，则默认为 **/usr/local/**。

```
sudo make DESTDIR=/opt/kudu install
```

7. 可选：构建文档。注意：此命令构建不适合上传到 **Kudu** 网站的本地文档。

```
$ make docs
```

## 示例 2 . Ubuntu 或者 Debian 构建脚本

该脚本概述了在 **Ubuntu** 上构建 **Kudu** 的过程，可以作为自动部署场景的基础。它跳过上面标记为可选步骤的步骤。

```
#!/bin/bash

sudo apt-get -y install autoconf automake curl g++ gcc gdb git \
    krb5-admin-server krb5-kdc krb5-user libkrb5-dev libsasl2-dev lib\
    libsasl2-modules-gssapi-mit libssl-dev libtool lsb-release make\
    patch pkg-config python rsync unzip vim-common
git clone https://github.com/apache/kudu
cd kudu
thirdparty/build-if-necessary.sh
mkdir -p build/release
cd build/release
../../thirdparty/installed/common/bin/cmake \
    -DCMAKE_BUILD_TYPE=release \
    ../../..
make -j4
```

## SUSE Linux Enterprise Server

1. 安装必备库（如果没有安装）。

```
$ sudo zypper install autoconf automake curl cyrus-sasl-devel \
    cyrus-sasl-gssapi gcc gcc-c++ gdb git krb5-devel libtool lsb\
    openssl-devel patch pkg-config python rsync unzip vim
```

2. 克隆 **Git** 存储库并切换到新的 **kudu** 目录。

```
$ git clone https://github.com/apache/kudu
```

```
$ cd kudu
```

3. 使用 **build-if-necessary.sh** 脚本构建任何缺少的第三方要求。

```
$ thirdparty/build-if-necessary.sh
```

4. 使用上一步中安装的实用程序构建 **Kudu**。为中间输出选择一个构建目录，除了 **kudu** 目录本身之外，它可以在文件系统中的任何位置。

```
mkdir -p build/release
cd build/release
../../thirdparty/installed/common/bin/cmake \
  -DCMAKE_BUILD_TYPE=release \
  ../../..
make -j4
```

5. 可选：安装 **Kudu** 二进制文件，库和标题。如果不通过 **DESTDIR** 环境变量指定安装目录，则默认为 **/usr/local/**。

```
sudo make DESTDIR=/opt/kudu install
```

### 示例 3 .SLES 构建脚本

此脚本概述了在 **SLES** 上构建 **Kudu** 的过程，并可用作自动部署场景的基础。它跳过上面对标记为可选步骤的步骤。

```
#!/bin/bash

sudo zypper install -y autoconf automake curl cyrus-sasl-devel \
  cyrus-sasl-gssapi gcc gcc-c++ gdb git krb5-devel libtool lsb-rel
  openssl-devel patch pkg-config python rsync unzip vim
git clone https://github.com/apache/kudu
cd kudu
thirdparty/build-if-necessary.sh
mkdir -p build/release
cd build/release
../../thirdparty/installed/common/bin/cmake \
```

```
-DCMAKE_BUILD_TYPE=release \  
../..  
make -j4
```

## macOS

**Xcode** 包是编译 **Kudu** 所必需的。下面的一些说明使用 [Homebrew](#) 来安装依赖关系，但手动依赖安装是可能的。

注意

### macOS 已知问题

**Kudu** 对 **macOS** 的支持是实验性的，只能用于开发。有关详细信息，请参阅 [macOS 限制和已知问题](#)。

1. 安装必备库（如果没有安装）。

```
$ brew tap homebrew/dupes  
$ brew install autoconf automake cmake git krb5 libtool openssl
```

2. 可选：如果您计划构建文档，请安装其他包，包括 **ruby** 。

```
$ brew install doxygen graphviz ruby  
$ brew install gnu-sed --with-default-names #The macOS default
```

3. 克隆 **Git** 存储库并切换到新的 **kudu** 目录。

```
$ git clone https://github.com/apache/kudu  
$ cd kudu
```

4. 使用 **build-if-necessary.sh** 脚本构建任何缺少的第三方要求。

```
$ thirdparty/build-if-necessary.sh
```



- 如果不同版本的依赖关系在调用 **thirdParty / build-if-necessary.sh** 时安装并使用，则可能会遇到类似于以下内容的输出：

```
./configure: line 16299: error near unexpected token `newli
./configure: line 16299: `  PKG_CHECK_MODULES('
```

第三方构建可能被缓存，并可能反映不正确版本的依赖关系。确保您具有步骤1中列出的正确依赖项，清理工作区，然后尝试重新构建。

```
$ git clean -fdx
$ thirdparty/build-if-necessary.sh
```

- 自制安装和系统升级的一些组合可能会导致不同种类的错误：

```
libtool: Version mismatch error. This is libtool 2.4.6, bu
libtool: definition of this LT_INIT comes from libtool 2.4.
libtool: You should recreate aclocal.m4 with macros from li
libtool: and run autoconf again.
```

如 [本线程](#) 所述，可能的修复是卸载并重新安装 **libtool**：

```
$ brew uninstall libtool && brew install libtool
```

5. 构建 **Kudu** 。为中间输出选择一个构建目录，除了 **kudu** 目录本身之外，它可以在文件系统中的任何位置。

```
mkdir -p build/release
cd build/release
../../thirdparty/installed/common/bin/cmake \
  -DCMAKE_BUILD_TYPE=release \
```

```
-DOPENSSL_ROOT_DIR=/usr/local/opt/openssl \  
../..  
make -j4
```

## 示例 4. macOS 构建脚本

此脚本概述了在 **MacOS** 上构建 **Kudu** 的过程，可用作自动部署场景的基础。它假定安装了 **Xcode** 和 **Homebrew**。

```
#!/bin/bash  
  
brew tap homebrew/dupes  
brew install autoconf automake cmake git krb5 libtool openssl pkg-  
git clone https://github.com/apache/kudu  
cd kudu  
thirdparty/build-if-necessary.sh  
mkdir -p build/release  
cd build/release  
../..../thirdparty/installed/common/bin/cmake \  
-DCMAKE_BUILD_TYPE=release \  
-DOPENSSL_ROOT_DIR=/usr/local/opt/openssl \  
../..  
make -j4
```

## 安装 C++ 客户端库

如果您需要访问 **Kudu** 客户端库进行开发，请为您的平台安装 **kudu-client** 和 **kudu-client-devel** 软件包。请参阅 [使用软件包安装](#)。

### 注意

仅针对客户端库和头（**kudu\_client.so** 和 **client.h**）构建。其他图书馆和标题是 **Kudu** 的内部，没有稳定的保证。

## 构建 Java 客户端

### 要求

- **JDK 7**
- **Apache Maven 3.x**

要构建 **Java** 客户端，请克隆 **Kudu Git** 存储库，更改为 **java** 目录，并发出以下命令：

```
$ mvn install -DskipTests
```

有关构建 **Java API** 以及 **Eclipse** 集成的更多信息，请参阅 **java/README.md** 。

## 查看 **API** 文档

### **C++ API** 文档

您可以在线查看 **C++客户端API文档**。或者，在 [从源代码构建 Kudu](#) 后，您还可以另外构建 **doxygen** 目标（例如，如果使用 **make**，则运行 **make doxygen**），并通过在您最喜欢的 **Web** 中打开 **docs/doxygen/client\_api/html/index.html** 文件来使用本地生成的 **API** 文档浏览器。

### 注意

为了构建 **doxygen** 目标，有必要在您的构建机器上安装带有 **Dot**（**graphviz**）支持的 **doxygen**。如果您从源代码构建 **Kudu** 后安装了 **doxygen**，则需要再次运行 **cmake** 以获取 **doxygen** 位置并生成适当的目标。

### **Java API** 文档

您可以在线查看 **Java API 文档**。或者，在 [构建 Java 客户端](#) 之后，**Java API** 文档可以在 **java/kudu-**

<client/target/apidocs/index.html> 中找到。

## 从以前版本的 **Kudu** 升级

升级之前，您应该阅读要安装的 **Kudu** 版本的发行说明。密切关注不兼容性，升级和降级在那里记录的笔记。

注意

目前不支持滚动升级。升级软件之前，请关闭所有的 **Kudu** 服务。

升级程序

1. 停止 **Kudu master** 和 **tablet server** 服务：

```
$ sudo service kudu-master stop  
$ sudo service kudu-tserver stop
```

2. 升级软件包：

\* 在 **RHEL** 或者 **CentOS** 主机上：

```
sudo yum clean all  
sudo yum upgrade kudu
```

\* 在 **SLES** 主机上：

```
sudo zypper clean --all  
sudo zypper update kudu
```

\* 在 **Ubuntu** 或者 **Debian** 主机上：

```
sudo apt-get update  
sudo apt-get install kudu
```

3. 启动 **Kudu master** 和 **tablet server** 服务：

```
$ sudo service kudu-master start  
$ sudo service kudu-tserver start
```

## 下一步

- 配置 **Kudu**
- 管理 **Kudu**

# 配置 Kudu

---

原文链接：<http://kudu.apache.org/docs/configuration.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813616>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## 配置 Apache Kudu

注意

**Kudu** 与 **Cloudera Manager** 相比，在独立安装中更容易配置。有关使用 **Kudu** 与 **Cloudera Manager** 的更多详细信息，请参阅 **Cloudera** 的 **Kudu** 文档。

## 配置 Kudu

配置基础

要配置每个 **Kudu** 进程的行为，您可以在启动时传递命令行标志，或者通过使用一个或多个 **--flagfile = <file>** 选项传递它们从配置文件读取这些选项。您甚至可以在配置文件中包含 **--flagfile** 选项以包含其他文件。通过阅读 [其文档](#) 了解有关 **gflags** 的更多信息。

您可以将 **masters** 和 **tablet servers** 的选项放置在相同的配置文件中，并且每个将忽略不适用的选项。

**Flags** 可以以一个或两个字符为前缀。本文档标准化在两个：**--example\_flag**。

## 发现配置选项

只有最常见的配置选项在此记录。有关配置选项的更详尽的列表，请参阅 [配置参考](#)。

要查看给定可执行文件的所有 **configuration flags** ( 配置标志 )，请使用 **--help** 选项运行它。配置未记录的标志时要小心，因为并非所有可能的配置都已经经过测试，并且无法保证在未来版本中保留未记录的选项。

## 配置 Kudu Master

要查看 **kudu-master** 可执行文件的所有可用配置选项，请使用 **--help** 选项运行它：

```
$ kudu-master --help
```

表1. 支持 **Kudu Masters** 的配置标志

Flag( 标记 )	Valid Options ( 有效选项 )	Default( 默认 )	Description ( 说明 )
--master_addresses	string	localhost	<b>Master</b> 共享配置的所有 <b>RPC</b> 地址的逗号分隔列表。如果未指定，假定为 <b>standalone Master</b>
--fs_data_dirs	string		<b>Master</b> 将放置其数据块的目录的逗号分隔列表。
			<b>Master</b> 将其预写日志的目录。可能与 -

--fs_wal_dir	string		<b>-fs_data_dirs</b> 中列出的目录之一相同，但不能与数据目录的子目录相同。
--log_dir	string	/tmp	存储主日志文件的目录。

有关 **masters** 标志的完整列表，请参见 "[Kudu Master 配置参考](#)"。

## 配置 Tablet Servers

要查看 **kudu tserver** 可执行文件的所有可用配置选项，请使用 **--help** 选项运行它：

```
$ kudu-tserver --help
```

表2 . 支持 **Kudu Tablet Servers** 的配置标志

Flag( 标记 )	Valid Options ( 有效选项 )	Default( 默认 )	Description ( 说明 )
--fs_data_dirs	string		<b>Tablet Server</b> 配置其数据的目录的号分隔列表。
			<b>Tablet Server</b> 配置预写日志的目录。能与 <b>--fs_data_</b>



<code>--fs_wal_dir</code>	string		中列出的录之一相同，但不与数据目的子目录同。
<code>--log_dir</code>	string	/tmp	存储 <b>Tablet Server</b> 文件的目录。
<code>--tserver_master_addrs</code>	string	127.0.0.1:7051	<b>Tablet Server</b> 连接的主机逗号分隔地址。 <b>masters</b> 读这个标志。
<code>--block_cache_capacity_mb</code>	integer	512	分配给 <b>Tablet Server</b> 缓存的最大容量。
<code>--memory_limit_hard_bytes</code>	integer	4294967296	<b>Tablet Server</b> 在开始挂所有传入之前消耗的最大内存量。

有关 **tablet servers** 的标志的完整列表，请参阅 "[Kudu Tablet Server 配置参考](#)"。

## 下一步

- [Kudu 入门指南](#)
- [使用 Kudu 开发应用程序](#)

# Kudu 集成 Apache Impala

原文链接：[http://kudu.apache.org/docs/kudu\\_impala\\_integration.html](http://kudu.apache.org/docs/kudu_impala_integration.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813620>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## 使用 Apache Kudu 与 Apache Impala（孵化）

**Kudu** 与 **Apache Impala**（孵化）紧密集成，允许您使用 **Impala** 使用 **Impala** 的 **SQL** 语法从 **Kudu tablets** 插入，查询，更新和删除数据，作为使用 **Kudu API** 构建自定义 **Kudu** 应用程序的替代方法。此外，您可以使用 **JDBC** 或 **ODBC** 将使用任何语言，框架或商业智能工具编写的现有或新应用程序与使用 **Impala** 作为代理的 **Kudu** 数据进行连接。

## 要求

- 此文档特定于 **Impala** 的某些版本。描述的语法将仅在以下版本中运行：
  - \* **CDH 5.10** 附带的 **Impala 2.7.0** 版本。 **SELECT VERSION()** 将报告 **impalad version 2.7.0-cdh5.10.0**。
  - \* 从源代码编译的 **Apache Impala 2.8.0** 版本。 **SELECT VERSION()** 将报告 **impalad version 2.8.0**。较早版本的 **Impala 2.7**（包括以前提供的特殊 **IMPALA\_KUDU** 版本）具有不兼容的语法。未来版本可能与此语法兼容，但我们建议您检查这是与您安装的相应版本相对应的最新可用文档。
- 本文档不描述 **Impala** 安装过程。请参阅 **Impala** 文档，并确保您可

以在继续之前对 **HDFS** 上的 **Impala** 表运行简单查询。

## 配置

**Kudu** 内不需要进行配置更改，从而可以访问 **Impala** 。

虽然不是绝对必要的，但建议您配置 **Impala** 与 **Kudu Master servers** 的位置：

- 在 **Impala** 服务配置中设置 **--kudu\_master\_hosts = <master1> [: port]**，**<master2> [: port]**，**<master3> [: port]** 标志。如果您正在使用 **Cloudera Manager**，请参阅相应的 **Cloudera Manager** 文档。

如果在 **Impala** 服务中未设置此标志，则每次创建表格时都必须手动提供此配置，方法是在 **TBLPROPERTIES** 子句中指定 **kudu\_master\_addresses** 属性。

本指南的其余部分假设配置已设置。

## 使用 **Impala Shell**

注意

这只是 **Impala Shell** 功能的一小部分。有关详细信息，请参阅 **Impala Shell** 文档。

- 使用 **impala-shell** 命令启动 **Impala Shell**。默认情况下，**impala-shell** 尝试连接到端口 **21000** 上的 **localhost** 上的 **Impala** 守护程序。要连接到其他主机，请使用 **-i <host:port>** 选项。要自动连接到特定的 **Impala** 数据库，请使用 **-d <database>** 选项。例如，如果您的所有 **Kudu** 表都位于 **Impala\_kudu** 数据库中的 **Impala** 中，

请使用 **-d impala\_kudu** 来使用此数据库。

- 要退出 **Impala Shell**，请使用以下命令：**quit**;

## 内部和外部 **Impala** 表

使用 **Impala** 创建新的 **Kudu** 表时，可以将表创建为内部表或外部表。

### **Internal** ( 内部 )

内部表由 **Impala** 管理，当您从 **Impala** 中删除时，数据和表确实被删除。当您使用 **Impala** 创建新表时，通常是内部表。

### **External** ( 外部 )

外部表（由 **CREATE EXTERNAL TABLE** 创建）不受 **Impala** 管理，并且删除此表不会将表从其源位置（此处为 **Kudu**）丢弃。相反，它只会去除 **Impala** 和 **Kudu** 之间的映射。这是 **Kudu** 提供的用于将现有表映射到 **Impala** 的语法。

有关内部和外部表的更多信息，请参阅 [Impala文档](#)。

## 查询 **Impala** 中现有的 **Kudu** 表

通过 **Kudu API** 或其他集成（如 **Apache Spark**）创建的表不会在 **Impala** 中自动显示。要查询它们，您必须先在 **Impala** 中创建外部表以将 **Kudu** 表映射到 **Impala** 数据库中：

```
CREATE EXTERNAL TABLE my_mapping_table
STORED AS KUDU
TBLPROPERTIES (
  'kudu.table_name' = 'my_kudu_table'
);
```

## 从 **Impala** 创建一个新的 **Kudu** 表

从 **Impala** 在 **Kudu** 中创建一个新表类似于将现有的 **Kudu** 表映射到 **Impala** 表，但您需要自己指定模式和分区信息。

使用以下示例作为指导。 **Impala** 首先创建表，然后创建映射。

```
CREATE TABLE my_first_table
(
  id BIGINT,
  name STRING,
  PRIMARY KEY(id)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

在 **CREATE TABLE** 语句中，必须首先列出构成主键的列。此外，主键列隐式标记为 **NOT NULL** 。

创建新的 **Kudu** 表时，需要指定一个分配方案。请参阅 [分区表](#)。为了简单起见，上面的表创建示例通过散列 **id** 列分成 16 个分区。有关分区的指导，请参阅 [Thumb](#) 的分区规则。

```
CREATE TABLE AS SELECT
```

您可以使用 **CREATE TABLE ... AS SELECT** 语句查询 **Impala** 中的任何其他表或表来创建表。以下示例将现有表 **old\_table** 中的所有行导入到 **Kudu** 表 **new\_table** 中。 **new\_table** 中的列的名称和类型将根据 **SELECT** 语句的结果集中的列确定。请注意，您必须另外指定主键和分区。

```
CREATE TABLE new_table
PRIMARY KEY (ts, name)
PARTITION BY HASH(name) PARTITIONS 8
STORED AS KUDU
AS SELECT ts, name, value FROM old_table;
```

## 指定 **Tablet Partitioning** ( **Tablet** 分区 )

表分为每个由一个或多个 **tablet servers** 提供的 **tablets** 。理想情况下，**tablets** 应该相对平等地拆分表的数据。 **Kudu** 目前没有自动（或手动）拆分预先存在的 **tablets** 的机制。在实现此功能之前，您必须在创建表时指定分区。在设计表格架构时，请考虑使用主键，您可以将表拆分成以类似速度增长的分区。使用 **Impala** 创建表时，可以使用 **PARTITION BY** 子句指定分区：

注意

**Impala** 关键字（如 **group** ）在关键字意义上不被使用时，由背面的字符包围。

```
CREATE TABLE cust_behavior (
  _id BIGINT PRIMARY KEY,
  salary STRING,
  edu_level INT,
  usergender STRING,
  `group` STRING,
  city STRING,
  postcode STRING,
  last_purchase_price FLOAT,
  last_purchase_date BIGINT,
  category STRING,
  sku STRING,
  rating INT,
  fulfilled_date BIGINT
)
PARTITION BY RANGE (_id)
(
  PARTITION VALUES < 1439560049342,
  PARTITION 1439560049342 <= VALUES < 1439566253755,
  PARTITION 1439566253755 <= VALUES < 1439572458168,
  PARTITION 1439572458168 <= VALUES < 1439578662581,
  PARTITION 1439578662581 <= VALUES < 1439584866994,
  PARTITION 1439584866994 <= VALUES < 1439591071407,
  PARTITION 1439591071407 <= VALUES
```

```
STORED AS KUDU;
```

如果您有多个主键列，则可以使用元组语法指定分区边界：（'va', 1），（'ab', 2）。该表达式必须是有效的 **JSON**。

## Impala 数据库和 Kudu

每个 **Impala** 表都包含在称为数据库的命名空间中。默认数据库称为默认数据库，用户可根据需要创建和删除其他数据库。

当从 **Impala** 中创建一个受管 **Kudu** 表时，相应的 **Kudu** 表将被命名为 **my\_database :: table\_name**。

不支持 **Kudu** 表的 **Impala** 关键字

创建 **Kudu** 表时不支持以下 **Impala** 关键字： - **PARTITIONED - LOCATION - ROWFORMAT**

优化评估 **SQL** 谓词的性能

如果您的查询的 **WHERE** 子句包含与 **operators =**，**<=**，**\**，**\**，**>=**，**BETWEEN** 或 **IN** 的比较，则 **Kudu** 直接评估该条件，只返回相关结果。这提供了最佳性能，因为 **Kudu** 只将相关结果返回给 **Impala**。对于谓词 **!=**，**LIKE** 或 **Impala** 支持的任何其他谓词类型，**Kudu** 不会直接评估谓词，而是将所有结果返回给 **Impala**，并依赖于 **Impala** 来评估剩余的谓词并相应地过滤结果。这可能会导致性能差异，这取决于评估 **WHERE** 子句之前和之后的结果集的增量。

分区表

根据主键列上的分区模式将表格划分为 **tablets**。每个 **tablet** 由至少一台 **tablet server** 提供。理想情况下，一张表应该分成多个 **tablets** 中



分布的 **tablet servers**，以最大化并行操作。您使用的分区模式的详细信息将完全取决于您存储的数据类型和访问方式。关于 **Kudu** 模式设计的全面讨论，请参阅 [Schema Design](#)。

**Kudu** 目前没有在创建表之后拆分或合并 **tablets** 的机制。创建表时，必须为表提供分区模式。在设计表格时，请考虑使用主键，这样您就可以将表格分为以相同速率增长的 **tablets**。

您可以使用 **Impala** 的 **PARTITION BY** 关键字对表进行分区，该关键字支持 **RANGE** 或 **HASH** 分发。分区方案可以包含零个或多个 **HASH** 定义，后面是可选的 **RANGE** 定义。**RANGE** 定义可以引用一个或多个主键列。[基本](#) 和 [高级分区](#) 的示例如下所示。

## 基本分区

### **PARTITION BY RANGE** ( 按范围划分 )

您可以为一个或多个主键列指定范围分区。**Kudu** 中的范围分区允许根据所选分区键的特定值或值的范围拆分表。这样可以平衡并行写入与扫描效率。

假设您有一个具有列 **state**，**name** 和 **purchase\_count** 的表。以下示例创建 50 个 **tablets**，每个 **US state**。

## 注意

### 单调增加 **Values**

如果您在其值单调递增的列上按范围进行分区，则最后一个 **tablet** 的增长将远大于其他平台。此外，插入的所有数据将一次写入单个 **tablet**，限制了数据摄取的可扩展性。在这种情况下，请考虑通过

**HASH** 分配，而不是或除 **RANGE** 之外。

```
CREATE TABLE customers (  
    state STRING,  
    name STRING,  
    purchase_count int,  
    PRIMARY KEY (state, name)  
)  
PARTITION BY RANGE (state)  
(  
    PARTITION VALUE = 'al',  
    PARTITION VALUE = 'ak',  
    PARTITION VALUE = 'ar',  
    -- ... etc ...  
    PARTITION VALUE = 'wv',  
    PARTITION VALUE = 'wy'  
)  
STORED AS KUDU;
```

## PARTITION BY HASH ( 哈希分区 )

您可以通过散列分发到特定数量的 “**buckets**”，而不是通过显式范围进行分发，或与范围分布组合。您指定要分区的主键列，以及要使用的存储桶数。通过散列指定的键列来分配行。假设散列的值本身不会表现出显着的偏差，这将有助于将数据均匀地分布在数据桶之间。

您可以指定多个定义，您可以指定使用复合主键的定义。但是，在多个散列定义中不能提及一列。考虑两列a和b： $\sqrt{* \text{HASH}(a), \text{HASH}(b)* \sqrt{\text{HASH}(a, b)*} \times \text{HASH}(a), \text{HASH}(a, b)}$

注意

没有指定列的 **HASH** 的 **PARTITION BY HASH** 是通过散列所有主键列来创建所需数量的桶的快捷方式。

如果主键值均匀分布在其域中，并且数据偏移不明显，例如

**timestamps** ( 时间戳 ) 或 **IDs** ( 序列号 ), 则哈希分区是合理的方法。

以下示例通过散列 **id** 和 **sku** 列创建 16 个 **tablets** 。这传播了所有 16 个 **tablets** 的写作。在这个例子中, 对一系列 **sku** 值的查询可能需要读取所有 16 个 **tablets** , 因此这可能不是此表的最佳模式。有关扩展示例, 请参阅 [高级分区](#)。

```
CREATE TABLE cust_behavior (  
  id BIGINT,  
  sku STRING,  
  salary STRING,  
  edu_level INT,  
  usergender STRING,  
  `group` STRING,  
  city STRING,  
  postcode STRING,  
  last_purchase_price FLOAT,  
  last_purchase_date BIGINT,  
  category STRING,  
  rating INT,  
  fulfilled_date BIGINT,  
  PRIMARY KEY (id, sku)  
)  
PARTITION BY HASH PARTITIONS 16  
STORED AS KUDU;
```

## 高级分区

您可以组合 **HASH** 和 **RANGE** 分区来创建更复杂的分区模式。您可以指定零个或多个 **HASH** 定义, 后跟零个或一个 **RANGE** 定义。每个定义可以包含一个或多个列。虽然枚举每个可能的分发模式都超出了本文档的范围, 但是几个例子说明了一些可能性。

## PARTITION BY HASH and RANGE

考虑上面的 [简单哈希](#) 示例, 如果您经常查询一系列 **sku** 值, 可以通过将哈希分区与范围分区相结合来优化示例。

以下示例仍然创建了16个 **tablets**，首先将 **id** 列分为 **4** 个存储区，然后根据 **sku** 字符串的值应用范围划分将每个存储区分为四个数据块。至少四片（最多可达**16**张）。当您查询相邻范围的 **sku** 值时，您很有可能只需从四分之一的 **tablets** 中读取即可完成查询。

## 注意

默认情况下，使用 **PARTITION BY HASH** 时，整个主键是散列的。要只对主键进行散列，可以使用像 **PARTITION BY HASH(id, sku)** 这样的语法来指定它。

```
CREATE TABLE cust_behavior (  
  id BIGINT,  
  sku STRING,  
  salary STRING,  
  edu_level INT,  
  usergender STRING,  
  `group` STRING,  
  city STRING,  
  postcode STRING,  
  last_purchase_price FLOAT,  
  last_purchase_date BIGINT,  
  category STRING,  
  rating INT,  
  fulfilled_date BIGINT,  
  PRIMARY KEY (id, sku)  
)  
PARTITION BY HASH (id) PARTITIONS 4,  
RANGE (sku)  
(  
  PARTITION VALUES < 'g',  
  PARTITION 'g' <= VALUES < 'o',  
  PARTITION 'o' <= VALUES < 'u',  
  PARTITION 'u' <= VALUES  
)  
STORED AS KUDU;
```

## Multiple **PARTITION BY HASH** Definitions ( 多重划分由 **HASH** 定义 )

再次扩展上述示例，假设查询模式将是不可预测的，但您希望确保写入分布在大量 **tablets** 上您可以通过在主键列上进行散列来实现整个主键的最大分配。

```
CREATE TABLE cust_behavior (  
  id BIGINT,  
  sku STRING,  
  salary STRING,  
  edu_level INT,  
  usergender STRING,  
  `group` STRING,  
  city STRING,  
  postcode STRING,  
  last_purchase_price FLOAT,  
  last_purchase_date BIGINT,  
  category STRING,  
  rating INT,  
  fulfilled_date BIGINT,  
  PRIMARY KEY (id, sku)  
)  
PARTITION BY HASH (id) PARTITIONS 4,  
              HASH (sku) PARTITIONS 4  
STORED AS KUDU;
```

该示例创建**16**个分区。您也可以使用 **HASH(id,sku) PARTITIONS 16**。但是，对于 **sku** 值的扫描几乎总是会影响所有**16**个分区，而不是可能限制为 **4**。

## Non-Covering Range Partitions ( 不覆盖分区 )

**Kudu 1.0** 及更高版本支持使用非覆盖范围分区，其解决方案如下：

- 没有未覆盖的范围分区，在需要考虑不断增加的主键的时间序列数据或其他模式的情况下，服务于旧数据的 **tablet** 的大小相对固定，而接收新数据的 **tablets** 将不受限制地增长。
- 在您希望根据其类别（如销售区域或产品类型）对数据进行分区的

情况下，无需覆盖范围分区，则必须提前了解所有分区，或者如果需要添加或删除分区，请手动重新创建表。例如引入或消除产品类型。

非覆盖范围分区有一些注意事项。请务必阅读链接：[/docs/schema\\_design.html](/docs/schema_design.html) [Schema Design guide]。

此示例每年创建一个 **tablet**（共5个 **tablets**），用于存储日志数据。该表仅接受 **2012** 年至 **2016** 年的数据。这些范围之外的键将被拒绝。

```
CREATE TABLE sales_by_year (  
  year INT, sale_id INT, amount INT,  
  PRIMARY KEY (sale_id, year)  
)  
PARTITION BY RANGE (year) (  
  PARTITION VALUE = 2012,  
  PARTITION VALUE = 2013,  
  PARTITION VALUE = 2014,  
  PARTITION VALUE = 2015,  
  PARTITION VALUE = 2016  
)  
STORED AS KUDU;
```

当记录开始进入 **2017** 年时，他们将被拒绝。在这一点上，**2017** 年的范围应该如下：

```
ALTER TABLE sales_by_year ADD RANGE PARTITION VALUE = 2017;
```

在需要数据保留的滚动窗口的情况下，范围分区也可能会丢弃。例如，如果不再保留 **2012** 年的数据，则可能会批量删除数据：

```
ALTER TABLE sales_by_year DROP RANGE PARTITION VALUE = 2012;
```

请注意，就像删除表一样，这不可逆转地删除存储在丢弃的分区中的所有数据。

## Partitioning Rules of Thumb ( 拇指分区规则 )

- 对于大型表格，如事实表，目标是在集群中拥有核心数量的 **tablets** 。
- 对于小型表格（如维度表），目标是足够数量的 **tablets**，每个 **tablets** 的大小至少为 **1 GB** 。

一般来说，请注意，在当前的实现中，**tablets** 的数量限制了读取的并行性。增加 **tablets** 数量超过核心数量可能会有减少的回报。

将数据插入 **Kudu** 表

**Impala** 允许您使用标准 **SQL** 语句将数据插入 **Kudu** 。

插入单个值

此示例插入单个行。

```
INSERT INTO my_first_table VALUES (99, "sarah");
```

此示例使用单个语句插入三行。

```
INSERT INTO my_first_table VALUES (1, "john"), (2, "jane"), (3, "jane");
```

批量插入

批量插入时，至少有三种常用选择。每个可能有优点和缺点，具体取决于您的数据和情况。

### Multiple single **INSERT** statements ( 多个单独的 **INSERT** 语句 )

这种方法具有易于理解和实现的优点。这种方法可能是低效的，因为 **Impala** 与 **Kudu** 的插入性能相比具有很高的查询启动成本。这将导致

相对较高的延迟和较差的吞吐量。

### Single INSERT statement with multiple VALUES ( 具有多个 VALUES 的单个 INSERT 语句 )

如果包含超过 1024 个 **VALUES** 语句，则在将请求发送到 **Kudu** 之前，**Impala** 将其分组为 1024（或 **batch\_size** 的值）。通过在 **Impala** 方面摊销查询启动处罚，此方法可能会执行比多个顺序 **INSERT** 语句略好。要设置当前 **Impala Shell** 会话的批量大小，请使用以下语法：**set batch\_size = 10000;**

注意

增加 **Impala** 批量大小会导致 **Impala** 使用更多的内存。您应该验证对群集的影响并进行相应调整。

### Batch Insert ( 批量插入 )

从 **Impala** 和 **Kudu** 的角度来看，通常表现最好的方法通常是使用 **Impala** 中的 **SELECT FROM** 语句导入数据。

1. 如果您的数据尚未在 **Impala** 中，则一种策略是 [从文本文件](#)（如 **TSV** 或 **CSV** 文件）导入。
2. [创建 Kudu 表](#)，注意指定为主键的列不能为空值。
3. 通过查询包含原始数据的表将值插入到 **Kudu** 表中，如以下示例所示：

```
INSERT INTO my_kudu_table
SELECT * FROM legacy_data_import_table;
```

### Ingest using the C++ or Java API ( 采用 C ++ 或 Java API )



在许多情况下，适当的摄取路径是使用 **C ++** 或 **Java API** 直接插入到 **Kudu** 表中。与其他 **Impala** 表不同，通过 **API** 插入到 **Kudu** 表中的数据可用于 **Impala** 中的查询，而不需要任何 **INVALIDATE METADATA** 语句或其他 **Impala** 存储类型所需的其他语句。

## **INSERT** and Primary Key Uniqueness Violations ( 插入和主键唯一性违规 )

在大多数关系数据库中，如果您尝试插入已插入的行，则插入将失败，因为主键将被重复。请参阅 **INSERT**，**UPDATE** 和 **DELETE** 操作中的故障。然而，**Impala** 不会失败查询。相反，它会生成一个警告，但是继续执行 **insert** 语句的其余部分。

如果插入的行意图替换现有行，则可以使用 **UPSERT** 而不是 **INSERT**。

```
INSERT INTO my_first_table VALUES (99, "sarah");
UPSERT INTO my_first_table VALUES (99, "zoe");
-- the current value of the row is 'zoe'
```

## 更新行

```
UPDATE my_first_table SET name="bob" where id = 3;
```

## 注意

当目标表在 **Kudu** 时，**UPDATE** 语句仅适用于 **Impala**。

## 批量更新

您可以使用批量插入中相同的方法 [批量更新](#)。

```
UPDATE my_first_table SET name="bob" where age > 10;
```

## 删除行

```
DELETE FROM my_first_table WHERE id < 3;
```

您也可以使用更复杂的语法进行删除。**FROM** 子句中的逗号是 **Impala** 指定连接查询的一种方法。有关 **Impala** 连接的更多信息，请参阅 <http://www.cloudera.com/content/cloudera/en/documentation/core/latest>

```
DELETE c FROM my_second_table c, stock_symbols s WHERE c.name = s.
```

## 注意

当目标表在 **Kudu** 时，**DELETE** 语句仅适用于 **Impala** 。

## 批量删除

您可以使用“[插入批量](#)”中概述的相同方法 批量删除 。

```
DELETE FROM my_first_table WHERE id < 3;
```

## **INSERT**，**UPDATE** 和 **DELETE** 操作期间的故障

**INSERT**，**UPDATE** 和 **DELETE** 语句不能被视为整体事务。如果这些操作中的一个无法部分通过，则可能已经创建了密钥（在 **INSERT** 的情况下），或者记录可能已被其他进程修改或删除（在 **UPDATE** 或 **DELETE** 的情况下）。您应该设计您的应用程序。

## 更改表属性

您可以通过更改表的属性来更改 **Impala** 与给定 **Kudu** 表相关的元数据。这些属性包括表名，**Kudu** 主地址列表，以及表是否由 **Impala**（内部）或外部管理。

## Rename an Impala Mapping Table ( 重命名 Impala 映射表 )

```
ALTER TABLE my_table RENAME TO my_new_table;
```

### 注意

使用 **ALTER TABLE ... RENAME** 语句重命名表仅重命名 **Impala** 映射表，无论该表是内部还是外部表。这样可以避免可能访问基础的 **Kudu** 表的其他应用程序的中断。

## Rename the underlying Kudu table for an internal table ( 重新命名内部表的基础 Kudu 表 )

如果表是内部表，则可以通过更改 **kudu.table\_name** 属性重命名底层的 **Kudu** 表：

```
ALTER TABLE my_internal_table  
SET TBLPROPERTIES('kudu.table_name' = 'new_name')
```

## Remapping an external table to a different Kudu table ( 将外部表重新映射到不同的 Kudu 表 )

如果另一个应用程序在 **Impala** 下重命名了 **Kudu** 表，则可以重新映射外部表以指向不同的 **Kudu** 表名称。

```
ALTER TABLE my_external_table_  
SET TBLPROPERTIES('kudu.table_name' = 'some_other_kudu_table')
```

## Change the Kudu Master Address ( 更改 Kudu Master 地址 )

```
ALTER TABLE my_table  
SET TBLPROPERTIES('kudu.master_addresses' = 'kudu-new-master.example');
```

## Change an Internally-Managed Table to External ( 将内部管理的表更改为外部 )

```
ALTER TABLE my_table SET TBLPROPERTIES('EXTERNAL' = 'TRUE');
```

使用 **Impala** 删除 **Kudu** 表

如果表是使用 **Impala** 中的内部表创建的，则使用 **CREATE TABLE**，标准 **DROP TABLE** 语法会删除底层的 **Kudu** 表及其所有数据。如果表被创建为一个外部表，使用 **CREATE EXTERNAL TABLE**，**Impala** 和 **Kudu** 之间的映射被删除，但 **Kudu** 表保持原样，并包含其所有数据。

```
DROP TABLE my_first_table;
```

## 下一步做什么？

上面的例子只是探索了 **Impala Shell** 所能做的一小部分。

- 了解 [Impala 项目](#)
- 阅读 [Impala 文档](#)
- 查看 [Impala SQL 参考](#)
- 阅读 **Impala** 内部部分或了解如何在 [Impala Wiki](#) 上为 **Impala** 做出贡献。
- 阅读原生的 [Kudu API](#)。

已知问题和限制

- 在使用 **Impala** 中的外部表格时，必须为具有大写字母或非 **ASCII** 字符的名称的 **Kudu** 表分配备用名称。
- 具有包含大写字母或非 **ASCII** 字符的列名称的 **Kudu** 表可能不会用作 **Impala** 中的外部表。专栏可能在 **Kudu** 更名为解决这个问题。
- 创建 **Kudu** 表时，**CREATE TABLE** 语句必须在主键顺序中包含其他列之间的主键列。
- 包含 **UNIXTIME\_MICROS** 类型列的库杜表可能不会用作 **Impala** 中的外部表。
- **Impala** 不能使用 **TIMESTAMP** , **DECIMAL** , **VARCHAR** 或嵌套类型的列创建 **Kudu** 表。
- **Impala** 无法更新主键列中的值。
- **NULL** , **NOT NULL** , **!=** 和 **LIKE** 谓词不会被推送到 **Kudu** , 而是会被 **Impala** 扫描节点评估。这可能会降低相对于其他类型谓词的性能。
- 通过 **Impala** 的更新, 插入和删除是非事务性的。如果查询失败的一部分途径, 其部分效果将不会回滚。
- 单个查询的最大并行度仅限于表中的 **tablets** 数量。为了获得良好的分析性能, 针对每个主机10个或更多个 **tablets** 或使用大型表。

# 管理 Kudu

原文链接：<http://kudu.apache.org/docs/administration.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813623>

贡献者：小瑶 ApacheCN Apache中文网

## Apache Kudu Administration ( 管理 Apache Kudu )

注意

**Kudu** 与 **Cloudera Manager** 相比在独立安装中更容易管理。有关使用 **Kudu** 与 **Cloudera Manager** 的更多详细信息，请参阅 **Cloudera Kudu** 文档。

## 启动和停止 Kudu 进程

重要

仅当使用操作系统软件包（例如 **rpm** 或 **deb**）安装 **Kudu** 时，这些说明才是相关的。

1. 使用以下命令启动 **Kudu** 服务：

```
$ sudo service kudu-master start
$ sudo service kudu-tserver start
```

2. 要停止 **Kudu** 服务，请使用以下命令：

```
$ sudo service kudu-master stop
```

```
$ sudo service kudu-tserver stop
```

## Kudu Web 界面

**Kudu tablet servers** 和 **masters** 在内置的 **Web** 界面上显示有用的操作信息，

### Kudu Master Web Interface

**Kudu** 主进程在 **8051** 端口上为其 **Web** 界面提供服务。该界面暴露了几个页面，其中包含有关群集状态的信息：

- **tablet servers** 列表，其 **host names** 和上次心跳时间。
- 表格列表，包括每个表的 **schema** 和 **tablet location information** 信息。
- 您可以将其粘贴到 **Impala Shell** 中以将现有表添加到 **Impala** 的已知数据源列表中的 **SQL** 代码。

### Kudu Tablets Server Web 界面

每个 **tablet server** 都提供端口 **8050** 上的 **Web** 界面。该界面显示有关服务器上托管的每个 **tablet** 的信息，其当前状态以及有关维护后台操作的调试信息。

常见的 **Web** 界面页面

**Kudu masters** 和 **tablet servers** 通过其 **Web** 界面暴露了一组通用的信息：

- HTTP 访问服务器日志。
- 一个 **/rpcz** 端点，通过 **JSON** 列出当前运行的 **RPC**。
- 页面提供了有关进程不同组件的内存使用情况的概述和详细信息。

- 关于当前配置标志的信息。
- 关于当前正在运行的线程及其资源消耗的信息。
- 一个 **JSON** 端点显示有关服务器的指标。
- 有关守护程序部署版本号的信息。

这些界面是从每个守护进程的 **Web UI** 的 **landing page** 链接的。

## Kudu Metrics ( 指标 )

**Kudu** 守护进程公开了大量的指标。一些指标与整个服务器进程相关联，而其他指标与特定 **tablet** 副本相关联。

列出可用的指标

**Kudu** 服务器的全套可用度量值可以通过特殊的命令行标志进行转储：

```
$ kudu-tserver --dump_metrics_json  
$ kudu-master --dump_metrics_json
```

这将输出一个大的 **JSON** 文档。每个指标都表示其名称，标签，描述，单位和类型。由于输出是 **JSON** 格式的，所以可以轻松地将这些信息进行解析，并将其馈送到从 **Kudu** 服务器收集指标的其他工具中。

通过**HTTP**收集指标

可以通过访问/指标通过其 **HTTP** 接口从服务器进程收集度量。此页面的输出是 **JSON** ，可以通过监视服务轻松解析。此端点在其查询字符串中接受几个 **GET** 参数：

- **/metrics?metrics=<substring1>,<substring2>,...** - 将返回的指标限制为至少包含一个提供的子字符串的指标。子串也匹配实体名称，因此可用于收集特定 **tablet** 的指标。



- **/metrics?include\_schema=1** - 包括 JSON 输出中的度量架构信息，如单位，描述和标签。通常这些信息可以节省空间。
- **/metrics?compact=1** - 从结果 JSON 中消除不必要的空格，从远程主机获取此页面时可以减少带宽。
- **/metrics?include\_raw\_histograms=1** - 包括直方图指标的原始存储桶和值，可实现随时间和跨主机的百分位数度量的准确聚合。

例如：

```
$ curl -s 'http://example-ts:8050/metrics?include_schema=1&metrics'
```

```
[
  {
    "type": "server",
    "id": "kudu.tabletserver",
    "attributes": {},
    "metrics": [
      {
        "name": "rpc_connections_accepted",
        "label": "RPC Connections Accepted",
        "type": "counter",
        "unit": "connections",
        "description": "Number of incoming TCP connections",
        "value": 92
      }
    ]
  }
]
```

```
$ curl -s 'http://example-ts:8050/metrics?metrics=log_append_later'
```

```
[
  {
    "type": "tablet",
    "id": "c0ebf9fef1b847e2a83c7bd35c2056b1",
    "attributes": {
```

```

        "table_name": "lineitem",
        "partition": "hash buckets: (55), range: [(<start>), (
        "table_id": ""
    },
    "metrics": [
        {
            "name": "log_append_latency",
            "total_count": 7498,
            "min": 4,
            "mean": 69.3649,
            "percentile_75": 29,
            "percentile_95": 38,
            "percentile_99": 45,
            "percentile_99_9": 95,
            "percentile_99_99": 167,
            "max": 367244,
            "total_sum": 520098
        }
    ]
}
]

```

## 注意

所有直方图和计数器都是从服务器开始时间开始测量的，收集后不会重置。

## 收集指标到日志

**Kudu** 可以配置为使用 **--metrics\_log\_interval\_ms** 标志将其所有度量值定期转储到本地日志文件。将此标志设置为将度量值写入日志文件的时间间隔。

度量日志将与其他 **Kudu** 日志文件一样写入与其相同的目录，具有相同的命名格式。任何指标日志文件达到 **64MB** 未压缩后，日志将被滚动，上一个文件将被 **gzip** 压缩。

生成的日志文件有三个空格分隔的字段。第一个字段是单词指标。第二个字段是自 **Unix** 纪元以来的微秒的当前时间戳。第三个是使用紧凑的 **JSON** 编码在服务器上的所有指标的当前值。编码与通过上述 **HTTP** 获取的指标相同。

## 注意

虽然度量记录自动滚动并压缩以前的日志文件，但它不会删除旧的日志文件。由于度量记录可以使用大量的磁盘空间，因此请考虑设置系统实用程序来监视日志目录中的空间并归档或删除旧段。

## 常见的 **Kudu** 工作流程

### 迁移到多个 **Kudu Master**

为了获得高可用性并避免出现单点故障，**Kudu** 集群应该由多个主人创建。许多 **Kudu** 集群是由一个单一的主人创建的，无论是简单的还是由于 **Kudu** 多主人的支持在当时仍然是实验性的。此工作流演示如何迁移到多主配置。

## 注意

将工作流添加到现有的多主配置中是不安全的。不要为此目的使用它。

## 注意

该工作流程至少基本上熟悉 **Kudu** 配置管理。如果使用 **Cloudera Manager (CM)**，工作流程也预先假定它是熟悉的。

## 注意

以下所有命令行步骤都应该像 **Kudu UNIX** 用户一般执行。

## 准备迁移

1. 建立维护窗口（一小时应该足够）。在此期间，**Kudu** 集群将不可用。
2. 决定使用多少 **masters**。**masters** 的数量应该是奇数。建议使用三或五个节点主配置;他们可以容忍一两个故障。
3. 对现有 **master** 执行以下准备步骤：
  - 识别和记录主数据所在的目录。如果使用 **Kudu** 系统软件包，则默认值为 **/var/lib/kudu/master**，但可以通过 **fs\_wal\_dir** 和 **fs\_data\_dirs** 配置参数进行自定义。请注意，如果您将 **fs\_data\_dirs** 设置为除 **fs\_wal\_dir** 值之外的某些目录，则应将其明确包含在其中还包含 **fs\_wal\_dir** 的每个命令中。
  - 识别和记录 **master** 正在为 **RPC** 使用的端口。默认端口值为 **7051**，但可能使用 **rpc\_bind\_addresses** 配置参数进行了自定义。
  - 识别 **master** 的 **UUID**。可以使用以下命令获取它：

```
$ kudu fs dump uuid --fs_wal_dir=<master_data_dir> 2>/dev/n
```

### **master\_data\_dir**

现有 **master** 以前记录的数据目录

例子

```
$ kudu fs dump uuid --fs_wal_dir=/var/lib/kudu/master 2>/de  
4aab798a69e94fab8d77069edff28ce0
```

- 可选：配置 **master** 的 **DNS** 别名。别名可能是 **DNS cname**（如果机器已经在 **DNS** 中有 **A** 记录），**A** 记录（如果机器仅

由其 **IP** 地址知道）或 **/etc/hosts** 中的别名。别名应该是 **master** 的抽象表示（例如，**master - 1**）。

注意

没有 **DNS** 别名，不可能从永久性主机故障中恢复，因此强烈建议。

4. 为每个新的主设备执行以下准备步骤：

- 在集群中选择一个未使用的机器。主机生成的负载很小，因此它可以与其他数据服务或负载生成过程共同配置，尽管与其他 **Kudu** 主机不同，配置相同。
- 确保 **Kudu** 通过系统包装（在这种情况下应安装 **kudu** 和 **kudu-master** 包装）或通过其他方式安装在机器上。
- 选择并记录主数据将存在的目录。
- 选择并记录主机应用于 **RPC** 的端口。
- 可选：配置主服务器的 **DNS** 别名（例如，**master-2**，**master-3** 等）。

执行迁移

1. 停止整个集群中的所有 **Kudu** 进程。
2. 格式化每个新 **master** 上的数据目录，并记录生成的 **UUID**。使用以下命令序列：

```
$ kudu fs format --fs_wal_dir=<master_data_dir>  
$ kudu fs dump uuid --fs_wal_dir=<master_data_dir> 2>/dev/null
```

**master\_data\_dir**

新 **master** 以前录制的数据目录。

示例

```
$ kudu fs format --fs_wal_dir=/var/lib/kudu/master  
$ kudu fs dump uuid --fs_wal_dir=/var/lib/kudu/master 2>/dev/n  
f5624e05f40649b79a757629a69d061e
```

3. 如果使用 **CM**，现在添加新的 **Kudu master roles**，但不要启动它们。
  - 如果使用 **DNS** 别名，请使用该主机的别名覆盖每个角色（包括现有主角色）的“**Master Address**”参数的空值。
  - 如果使用非默认 **RPC** 端口值，请添加端口号（以冒号分隔）。
4. 使用以下命令重写 **master** 的 **Raft** 配置，在现有主机上执行：

```
$ kudu local_replica cmeta rewrite_raft_config --fs_wal_dir=<ma
```

## **master\_data\_dir**

现有 **master** 以前记录的数据目录

**tablet\_id**必须是字符串

00000000000000000000000000000000**all\_masters**空格分隔的

**master** 的列表，新的和现在的。列表中的每个条目必须是 <uuid>:

<hostname>: <port> 格式的字符串 **UUID**

master 以前记录的 **UUID**

## **hostname**

master 以前记录的 **hostname** 或 别名

## **port**

master 以前记录的 **RPC** 的端口号

示例

```
$ kudu local_replica cmeta rewrite_raft_config --fs_wal_dir=/v
```

5. 修改现有 **master** 和新 **masters** 的 **master\_addresses** 配置参数的值。新值必须是所有主节点的逗号分隔列表。每个条目是 **<hostname>: <port>** 形式的字符串

**hostname**

master 以前记录的 hostname 和 别名

**port**

master 以前记录的 RPC 端口号

6. 启动现有的 **master**
7. 使用以下命令将 **master** 数据复制到每个新 **master**，在每台新 **master** 上执行：

```
$ kudu local_replica copy_from_remote --fs_wal_dir=<master_data
```

**master\_data\_dir**

新 master 以前记录的数据目录

**tablet\_id**

必须是字符串 000000000000000000000000000000000000

**existing\_master**

现有 master 的 RPC 地址必须是 **<hostname>: <port>** 形式的字符串

## hostname

现有 master 以前记录的 hostname 或别名

## port

现有 master 以前记录的 RPC 端口号

示例

```
$ kudu local_replica copy_from_remote --fs_wal_dir=/var/lib/kud
```

## 8. 启动所有的新的 master

注意

如果使用 **CM**，请跳过下一步。

## 9. 修改每个 tablet server 的 tserver\_master\_addrs 配置参数的值。

新值必须是以逗号分隔的主文件列表，其中每个条目是

**<hostname>** 形式的字符串：**<port>**

## hostname

master 以前记录的 hostname 和别名

## port

master 以前记录的 RPC 端口号

## 10. 启动所有的 tablet servers

恭喜，集群已经迁移到多个 **masters** 了！要验证所有 **master** 是否正常工作，请考虑执行以下健康性检查：

- 使用浏览器访问每个 **master** 的 **Web UI**。看看 **/master page**。所有的 **master** 都应该被列在那里，一个主人在 **LEADER** 角色，其他的在 **FOLLOWER** 角色。每个 **master** 的 **master** 的内容应该是一



样的。

- 使用 **kudu** 命令行工具在集群上运行 **Kudu** 系统检查（**ksck**）。有关详细信息，请参阅 [使用 \*\*ksck\*\* 检查群集运行状况](#)。

## 在多 **Master** 部署中从死亡的 **Kudu Master** 恢复

**Kudu multi-master** 部署功能通常在主机丢失的情况下。但是，重要的是更换死主;否则第二次故障可能会导致可用性的丢失，具体取决于可用 **master** 的数量。此工作流程描述如何更换 **dead master**。

由于 [KUDU-1620](#)，无需重新启动现场主机即可执行此工作流程。因此，工作流需要一个维护窗口，尽管主要通常是快速重启。

### 注意

**Kudu** 还不支持主人的 **Raft** 配置更改。因此，如果部署是使用 **DNS** 别名创建的，则只能替换主服务器。有关详细信息，请参阅 [multi-master 移动工作流程](#)。

### 注意

该工作流程至少基本上熟悉 **Kudu** 配置管理。如果使用 **Cloudera Manager**（**CM**），工作流程也预先假定它是熟悉的。

### 注意

以下所有命令行步骤都应该像 **Kudu UNIX** 用户一般执行。

## 为恢复做准备

1. 确保 **dead master** 机器正常并且真正死亡。采取一切必要步骤，防

止意外重启;这对于集群后恢复来说可能是相当危险的。

2. 选择剩下的仍然活着的 **master** 之一作为恢复的基础。这个工作流程的其余部分将把这个主机称为“**reference**”主机。
3. 选择 **new master** 所在的群集中未使用的机器。**master** 生成的负载很小，因此它可以与其他数据服务或负载生成过程共同配置，尽管与其他 **Kudu** 主机不同，配置相同。这个工作流程的其余部分将把这个主机称为“**replacement**”主机。
4. 为 **replacement master** 执行以下准备步骤：
  - 确保 **Kudu** 通过系统包装（在这种情况下应安装 **kudu** 和 **kudu-master** 包装）或通过其他方式安装在机器上。
  - 选择并记录 **master** 数据将存在的目录。
5. 为每个现在活着的 **master** 执行以下步骤：
  - 识别和记录主数据所在的目录。如果使用 **Kudu** 系统软件包，则默认值为 **/var/lib/kudu/master**，但可以通过 **fs\_wal\_dir** 和 **fs\_data\_dirs** 配置参数进行自定义。请注意，如果您将 **fs\_data\_dirs** 设置为除 **fs\_wal\_dir** 值之外的某些目录，则应将其明确包含在其中还包含 **fs\_wal\_dir** 的每个命令中。
  - 识别和记录 **master** 的 **UUID**。可以使用以下命令获取它：

```
$ kudu fs dump uuid --fs_wal_dir=<master_data_dir> 2>/dev/n
```

**master\_data\_dir**

活着 master 以前记录的数据目录

示例

```
$ kudu fs dump uuid --fs_wal_dir=/var/lib/kudu/master 2>/dev/n  
80a82c4b8a9f4c819bab744927ad765c
```

## 6. 对 **reference master** 执行以下准备步骤：

- 识别和记录主数据所在的目录。如果使用 **Kudu** 系统软件包，则默认值为 **/var/lib/kudu/master**，但可以通过 **fs\_wal\_dir** 和 **fs\_data\_dirs** 配置参数进行自定义。请注意，如果您将 **fs\_data\_dirs** 设置为除 **fs\_wal\_dir** 值之外的某些目录，则应将其明确包含在其中还包含 **fs\_wal\_dir** 的每个命令中。
- 使用以下命令识别并记录集群中每个 **master** 的 **UUID**：

```
$ kudu local_replica cmeta print_replica_uuids --fs_wal_dir
```

**master\_data\_dir**

reference master 以前记录的数据目录

**tablet\_id**

必须是字符串 000000000000000000000000000000000000

示例

```
$ kudu local_replica cmeta print_replica_uuids --fs_wal_dir  
80a82c4b8a9f4c819bab744927ad765c 2a73eeee5d47413981d9a1c637
```

## 7. 使用以前记录的两个 **UUID** 列表（一个用于所有 **live master**，一个用于所有 **master**），确定并记录（通过消除处理）**dead master** 的 **UUID**。

执行恢复

### 1. 使用以前记录的 **dead master** 主机的 **UUID** 格式化 **replacement**

**master** 上的数据目录。使用以下命令序列：

```
$ kudu fs format --fs_wal_dir=<master_data_dir> --uuid=<uuid>
```

**master\_data\_dir**

replacement master 以前记录的数据目录

**uuid**

dead master 以前记录的 UUID

例子

```
$ kudu fs format --fs_wal_dir=/var/lib/kudu/master --uuid=80a8:
```

2. 使用以下命令将 **master** 数据复制到 **replacement master** :

```
$ kudu local_replica copy_from_remote --fs_wal_dir=<master_data_dir>
```

**master\_data\_dir**

replacement master 以前记录的数据目录

**tablet\_id**

必须是字符串 00000000000000000000000000000000

**reference\_master**

reference\_master 的 RPC 地址必须是 <hostname>: <port> 形式的字符串

## hostname

reference master 以前记录的 hostname 或别名

## port

reference master 之前记录的 RPC 的端口号

示例

```
$ kudu local_replica copy_from_remote --fs_wal_dir=/var/lib/kud
```

3. 如果使用 **CM**，请立即添加替换的 **Kudu master role**，但不要启动它。
  - 使用 **replacement master** 的别名覆盖 **new role** 的 “**Master Address**” 参数的空值。
  - 如果使用非默认 **RPC** 端口值，请添加端口号（以冒号分隔）。
4. 重新配置 **dead master** 的 **DNS** 别名以指向 **replacement master**。
5. 启动 **replacement master**。
6. 重新启动现有的 **live master**。这导致短暂的可用性中断，但是只有 **masters** 回来才能持续。

恭喜，**dead master** 已经被替换了！要验证所有 **master** 是否正常工作，请考虑执行以下健康性检查：

- 使用浏览器访问每个 **master** 的 **Web UI**。看看 **/master page**。所有的主人都应该被列在那里，一个 **master** 在 **LEADER** 角色，其他的在 **FOLLOWER** 角色。每个 **master** 的 **master** 的内容应该是一样的。
- 使用 **kudu** 命令行工具在集群上运行 **Kudu** 系统检查（**ksck**）。有关详细信息，请参阅 [使用 ksck 检查群集运行状况](#)。

使用 **ksck** 检查集群运行状况

**kudu CLI** 包括一个名为 **ksck** 的工具，可用于检查群集运行状况和数据完整性。**ksck** 会发现问题，如副本不足的 **tablet**，无法连接的 **tablet server** 或没有 **leader** 的 **tablet**。

应从命令行运行 **ksck**，并要求指定 **master address** 的完整列表：

```
$ kudu cluster ksck master-01.example.com,master-02.example.com,ma
```

要查看 **ksck** 可用选项的完整列表，请使用 **--help** 标志。如果集群是健康的，**ksck** 将打印成功消息，并返回零（成功）退出状态。

```
Connected to the Master
Fetched info from all 1 Tablet Servers
Table IntegrationTestBigLinkedList is HEALTHY (1 tablet(s) checked)

The metadata for 1 table(s) is HEALTHY
OK
```

如果集群不健康，例如，如果 **tablet server** 进程已停止，则 **ksck** 将报告问题并返回非零退出状态：

```
Connected to the Master
WARNING: Unable to connect to Tablet Server 8a0b66a756014def82760a09946d1fce (tserver-01.example.com:7050): Network error: could not send Ping
WARNING: Fetched info from 0 Tablet Servers, 1 weren't reachable
Tablet ce3c2d27010d4253949a989b9d9bf43c of table 'IntegrationTestBigLinkedList' is unavailable: 1 replica(s) not RUNNING
      8a0b66a756014def82760a09946d1fce (tserver-01.example.com:7050):

Table IntegrationTestBigLinkedList has 1 unavailable tablet(s)

WARNING: 1 out of 1 table(s) are not in a healthy state
=====
Errors:
=====
```

```
error fetching info from tablet servers: Network error: Not all  
table consistency check error: Corruption: 1 table(s) are bad
```

```
FAILED
```

```
Runtime error: ksck discovered errors
```

为了验证数据完整性，可以设置可选的 **--checksum\_scan** 标志，通过扫描每个 **tablet** 副本并比较结果，可以确保集群具有一致的数据。 **--tables** 或 **--tablets** 标志可用于将校验和扫描的范围分别限制在特定的表格或 **tablet** 上。例如，可以使用以下命令对 **IntegrationTestBigLinkedList** 表上的数据完整性进行检查：

```
$ kudu cluster ksck --checksum_scan --tables IntegrationTestBigLinkedList
```

### 从磁盘故障恢复

**Kudu tablet servers** 无法恢复磁盘故障。当包含数据目录或预写日志（**WAL**）的磁盘死机时，必须重建整个 **tablet server**。在 **tablet server** 发生故障后，**Kudu** 会在其他服务器上自动重新复制 **tablet**，但需要手动干预才能将失败的 **tablet server** 还原到运行状态。

在磁盘故障后恢复 **tablet servers** 的第一步是更换发生故障的磁盘，或从数据目录 **and/or WAL** 配置中删除出现故障的磁盘。接下来，必须删除数据目录和 **WAL** 目录的内容。例如，如果 **tablet servers** 配置了 **--fs\_wal\_dir = /data/0/kudu-tserver-wal** 和 **--fs\_data\_dirs = /data/1/kudu-tserver/data/2/kudu-tserver**，则以下命令将删除数据目录和 **WAL** 目录内容：

```
$ rm -rf /data/0/kudu-tserver-wal/* /data/1/kudu-tserver/* /data/2/kudu-tserver/*
```

在 **WAL** 和数据目录被清空之后，可以启动 **tablet servers** 进程。当使用系统包安装 **Kudu** 时，通常使用服务：

```
$ sudo service kudu-tserver start
```

一旦 **tablet servers** 再次运行，就会根据需要在其上创建新的 **tablet** 副本。



# Kudu 故障排除

原文链接：<http://kudu.apache.org/docs/troubleshooting.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813626>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## 启动错误

### Errors During Hole Punching Test ( 穿透测试中的错误 )

**Kudu** 需要 **hole punching capabilities** ( 穿透能力 ) 才能有效。穿透支持取决于您的操作系统内核版本和本地文件系统实现。

- **RHEL** 或 **CentOS 6.4** 或更高版本，修补到 **2.6.32-358** 或更高版本的内核版本。未配置的 **RHEL** 或 **CentOS 6.4** 不包括支持穿透的内核。
- **Ubuntu 14.04** 包括 **Linux** 内核的 **3.13** 版本，它支持打孔。
- 较新版本的 **EXT4** 或 **XFS** 文件系统支持穿透，但 **EXT3** 不支持。旧版本的 **XFS** 不支持穿透返回 **EOPNOTSUPP** ( 操作不支持 ) 错误。不支持穿透的 **EXT4** 或 **XFS** 的较旧版本会导致 **Kudu** 发出以下错误消息，并且无法启动：

穿透试验时出错日志块管理器需要一个具有穿透支持的文件系统，如 **ext4** 或 **xfs** 。在 **el6** 上，内核版本 **2.6.32-358** 或更新版本是必需的。不使用穿透时 ( 以某种效率和可扩展性为代价 ) ，重新配置 **Kudu** 与 **--block\_manager = file** 。有关更多信息，请参阅 **Kudu** 文档细节。原始

错误消息如下。

没有穿透支持，日志块管理器不安全使用。它不会删除块，并将消耗更多的磁盘空间。

即使你的环境中无法使用穿透，你仍然可以尝试 **Kudu** 。通过将 **--block\_manager = file** 标志添加到用于启动主服务器和 **tablet server** 的命令，启用文件块管理器而不是日志块管理器。文件块管理器不会与日志块管理器一样缩放。

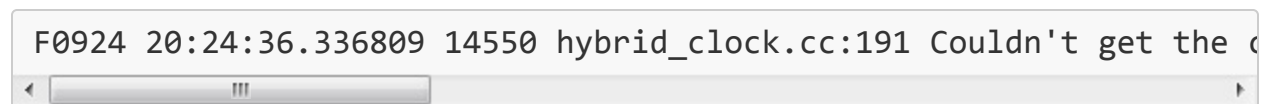
注意

由于文件块管理器的规模和性能不佳，只能用于小规模的评价和开发。

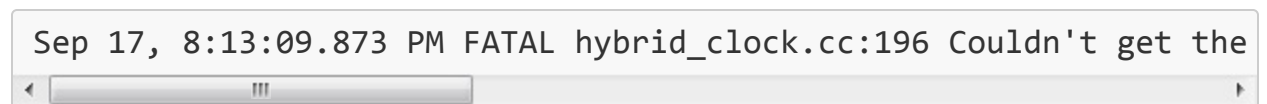
## NTP Clock Synchronization ( NTP 时钟同步 )

对于主服务器和平板电脑服务器守护程序，必须使用 **NTP** 同步服务器的时钟。另外，最大时钟误差（不要误估为误差）低于可配置的阈值。默认值为 **10** 秒，但可以使用标志 **--max\_clock\_sync\_error\_usec** 设置。

如果未安装 **NTP** ，或者如果时钟被报告为不同步，则 **Kudu** 将不会启动，并会发出如下消息：

A terminal window screenshot showing a log entry: "F0924 20:24:36.336809 14550 hybrid\_clock.cc:191 Couldn't get the c". The text is truncated at the end. The terminal has a scrollbar on the right.

如果 **NTP** 安装并同步，但最大时钟错误太高，用户将看到如下消息：

A terminal window screenshot showing a log entry: "Sep 17, 8:13:09.873 PM FATAL hybrid\_clock.cc:196 Couldn't get the". The text is truncated at the end. The terminal has a scrollbar on the right.

或者



重要

如果已安装 **NTP**，用户可以通过运行 **ntptime** 来监视同步状态。相关值是报告的最大误差。

要安装 **NTP**，请对您的操作系统使用相应的命令：

OS ( 操作系统 )	Command ( 命令 )
Debian/Ubuntu	sudo apt-get install ntp
RHEL/CentOS	sudo yum install ntp

如果 **NTP** 已安装但未运行，请使用以下命令之一启动它：

OS ( 操作系统 )	Command ( 命令 )
Debian/Ubuntu	sudo service ntp restart
RHEL/CentOS	sudo /etc/init.d/ntpd restart

重要

**NTP** 需要网络连接，可能需要几分钟才能同步时钟。在某些情况下，有点网络连接可能使 **NTP** 将时钟报告为不同步。一个常见的，虽然临时的解决方法是使用上述命令重新启动 **NTP**。

如果时钟被 **NTP** 报告为同步，但是最大误差太高，则用户可以通过设置上述标志来将阈值增加到更高的值。例如，将可能的最大误差增加到 **20** 秒，标志应该如下设置：**--max\_clock\_sync\_error\_usec =**

20000000

## 报告 Kudu 崩溃

**Kudu** 在 **Kudu** 遇到崩溃时使用 **Google Breakpad** 库来生成 **minidump**。这些 **minidumps** 的大小通常只有几 **MB**，即使禁用了核心转储生成也会生成这些。在这个时候，只能在 **Kudu** 的 **Linux** 上生成 **minidumps**。

**minidump** 文件包含有关崩溃的进程的重要调试信息，包括加载的共享库及其版本，崩溃时运行的线程列表，处理器寄存器的状态和每个线程的堆栈内存副本，以及 **CPU** 和操作系统版本信息。

也可以强制 **Kudu** 通过向 **kudu-tserver** 或 **kudu-master** 进程发送 **USR1** 信号来创建一个 **minidump**，而不会杀死进程。例如：

```
sudo pkill -USR1 kudu-tserver
```

默认情况下，**Kudu** 将其 **minidump** 存储在其配置的名为 **minidumps** 的 **glog** 目录的子目录中。可以通过设置 **--minidump\_path** 标志来定制该位置。在删除最旧的之前，**Kudu** 将只保留一定数量的 **minidumps**，以避免用 **minidump** 文件填满磁盘。可以通过设置 **--max\_minidumps gflag** 来控制将保留的最小数量。

**Minidumps** 包含特定于创建它们的二进制文件的信息，因此在不访问崩溃的确切二进制文件的情况下不可用，或者非常相似的二进制文件。

重要

**Minitump** 可以通过电子邮件发送给 **Kudu** 开发人员或附加到 **JIRA**，以帮助 **Kudu** 开发人员调试崩溃。为了使其有用，开发人员将需要知道

**Kudu** 的确切版本和发生崩溃的操作系统。请注意，虽然 **minidump** 不包含堆内存转储，但它确实包含堆栈内存，因此可以将应用程序数据显示在**minidump** 中。如果机密或个人信息存储在群集上，请不要共享 **minidump** 文件。

## 性能故障排除

### Kudu追踪

**kudu-master** 和 **kudu-tserver** 守护进程包括基于开源 **Chromium** 跟踪框架的内置跟踪支持。您可以使用跟踪来帮助诊断延迟问题或 **Kudu** 服务器上的其他问题。

#### 访问跟踪界面

每个 **Kudu** 守护进程中，跟踪界面是通过 **Web** 浏览器访问嵌入式 **Web** 服务器的一部分。

表 1. 跟踪界面 **URL**

Daemon ( 守护进程 )	URL
Tablet Server	<a href="http://tablet-server-1.example.com:8050/tracing.html">http://tablet-server-1.example.com:8050/tracing.html</a>
Master	<a href="http://master-1.example.com:8051/tracing.html">http://master-1.example.com:8051/tracing.html</a>

#### 注意

已知跟踪界面适用于最新版本的 **Google Chrome** 。其他浏览器可能无法正常工作。

## Collecting a trace ( 收集痕迹 )

导航到跟踪界面后，单击屏幕左上角的记录按钮。当开始诊断问题时，首先选择所有类别。单击记录开始记录跟踪。

在跟踪收集期间，事件被收集到内存中的环形缓冲区中。该环形缓冲器的大小固定，最终可以达到100%。但是，在删除旧事件的同时，仍然收集新的事件。记录跟踪时，触发您有兴趣探索的行为或工作负载。

收集数秒后，单击停止。收集的踪迹将被下载并显示。使用 ? 键显示有关使用跟踪界面探索跟踪的帮助文本。

## Saving a trace ( 保存痕迹 )

您可以将收集的痕迹保存为 **JSON** 文件，以便以后分析，然后在收集跟踪后单击“保存”。要加载和分析保存的 **JSON** 文件，请单击加载并选择文件。

## RPC Timeout Traces ( RPC超时跟踪 )

如果客户端应用程序遇到 **RPC** 超时，**Kudu tablet servers**

**WARNING** 级别日志应包含包含 **RPC** 级别跟踪的日志条目。例如：

```
W0922 00:56:52.313848 10858 inbound_call.cc:193] Call kudu.consensus
from 192.168.1.102:43499 (request call id 3555909) took 1464ms (c
W0922 00:56:52.314888 10858 inbound_call.cc:197] Trace:
0922 00:56:50.849505 (+      0us) service_pool.cc:97] Inserting ont
0922 00:56:50.849527 (+     22us) service_pool.cc:158] Handling cal
0922 00:56:50.849574 (+     47us) raft_consensus.cc:1008] Updating
0922 00:56:50.849628 (+     54us) raft_consensus.cc:1050] Early mar
0922 00:56:50.849968 (+    340us) raft_consensus.cc:1056] Triggerir
0922 00:56:50.850119 (+    151us) log.cc:420] Serialized 1555 byte
0922 00:56:50.850213 (+     94us) raft_consensus.cc:1131] Marking c
0922 00:56:50.850218 (+      5us) raft_consensus.cc:1148] Updating
0922 00:56:50.850219 (+      1us) raft_consensus.cc:1195] Filling c
0922 00:56:50.850221 (+      2us) raft_consensus.cc:1169] Waiting c
```

```
0922 00:56:52.313763 (+1463542us) raft_consensus.cc:1182] finished
0922 00:56:52.313764 (+      1us) raft_consensus.cc:1190] UpdateRep
0922 00:56:52.313788 (+     24us) inbound_call.cc:114] Queueing suc
```

这些跟踪可以指示请求的哪个部分缓慢。请将它们包含在与 **RPC** 延迟异常值相关的错误报告中。

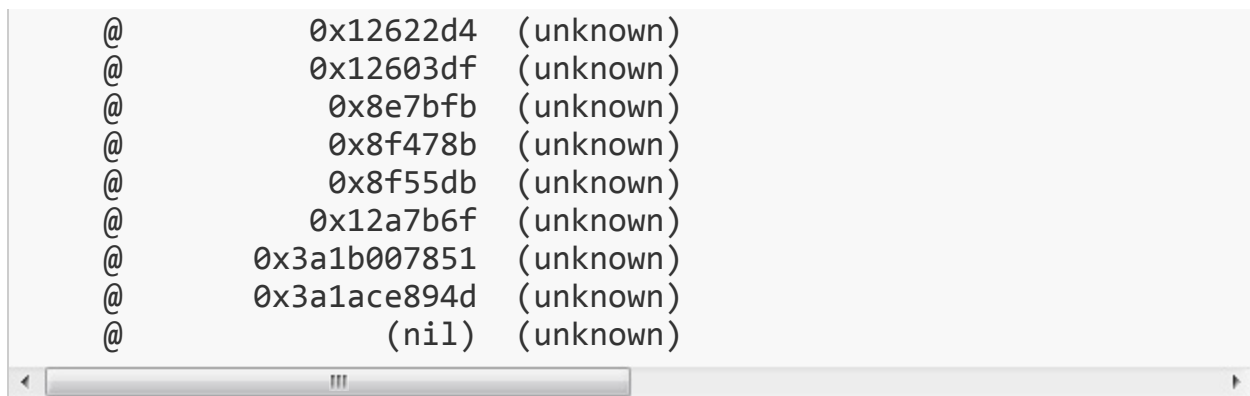
## Kernel Stack Watchdog Traces ( 内核堆栈看门狗跟踪 )

每个 **Kudu** 服务器进程都有一个称为 **Stack Watchdog** 的后台线程，它监视服务器中的其他线程，以防它们被阻塞超过预期的时间段。这些跟踪可以指示操作系统问题或瓶颈存储。

当看门狗线程识别线程阻塞的情况时，它会在 **WARNING** 日志中记录一个条目，如下所示：

```
W0921 23:51:54.306350 10912 kernel_stack_watchdog.cc:111] Thread 1
Kernel stack:
[<fffffffffa00b209d>] do_get_write_access+0x29d/0x520 [jbd2]
[<fffffffffa00b2471>] jbd2_journal_get_write_access+0x31/0x50 [jbd2]
[<fffffffffa00fe6d8>] __ext4_journal_get_write_access+0x38/0x80 [ext4]
[<fffffffffa00d9b23>] ext4_reserve_inode_write+0x73/0xa0 [ext4]
[<fffffffffa00d9b9c>] ext4_mark_inode_dirty+0x4c/0x1d0 [ext4]
[<fffffffffa00d9e90>] ext4_dirty_inode+0x40/0x60 [ext4]
[<fffffffff811ac48b>] __mark_inode_dirty+0x3b/0x160
[<fffffffff8119c742>] file_update_time+0xf2/0x170
[<fffffffff8111c1e0>] __generic_file_aio_write+0x230/0x490
[<fffffffff8111c4c8>] generic_file_aio_write+0x88/0x100
[<fffffffffa00d3fb1>] ext4_file_write+0x61/0x1e0 [ext4]
[<fffffffff81180f5b>] do_sync_readv_writev+0xfb/0x140
[<fffffffff81181ee6>] do_readv_writev+0xd6/0x1f0
[<fffffffff81182046>] vfs_writev+0x46/0x60
[<fffffffff81182102>] sys_pwritev+0xa2/0xc0
[<fffffffff8100b072>] system_call_fastpath+0x16/0x1b
[<ffffffffffffffff>] 0xffffffffffffffff

User stack:
  @      0x3a1ace10c4 (unknown)
  @      0x1262103 (unknown)
```



@	0x12622d4	(unknown)
@	0x12603df	(unknown)
@	0x8e7bfb	(unknown)
@	0x8f478b	(unknown)
@	0x8f55db	(unknown)
@	0x12a7b6f	(unknown)
@	0x3a1b007851	(unknown)
@	0x3a1ace894d	(unknown)
@	(nil)	(unknown)

这些跟踪可用于诊断根源于延迟问题（由 **Kudu** 以下的系统引起），如磁盘控制器或文件系统。

## 使用 **Kudu** 的问题

### **ClassNotFoundException:**

#### **com.cloudera.kudu.hive.KuduStorageHandler**

尝试通过 **Hive** 使用 **Kudu** 表时，用户将遇到此异常。这不是一个丢失的 **jar** 的情况，而只是 **Impala** 将 **Hive** 中的 **Kudu** 元数据存储在其他工具（包括 **Hive** 本身和 **Spark**）中无法读取的格式。 **Hive** 用户没有解决方法。 **Spark** 用户需要创建临时表。



# 使用 **Kudu** 开发应用程序

原文链接：<http://kudu.apache.org/docs/developing.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813629>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## 使用 **Kudu** 开发应用程序

**Kudu** 提供 **C ++**，**Java** 和 **Python** 客户端 **API** 以及参考示例来说明它们的用途。

注意

不支持使用服务器端或专用接口，不属于公共 **API** 的接口没有稳定性保证。

## 查看 **API** 文档

### **C++ API** 文档

您可以在线查看 **C ++** 客户端 **API** 文档。或者，在从源代码构建 **Kudu** 后，您还可以另外构建 **doxygen** 目标（例如，如果使用 **make**，则运行 **make doxygen**），并通过在您最喜欢的 **Web** 中打开 **docs/doxygen/client\_api/html/index.html** 文件来使用本地生成的 **API** 文档浏览器。

重要

为了构建 **doxygen** 目标，有必要在您的构建机器上安装带有 **Dot**（

**graphviz** ) 支持的 **doxygen** 。如果您从源代码构建 **Kudu** 后安装了 **doxygen** ，则需要再次运行 **cmake** 以获取 **doxygen** 位置并生成适当的目标。

## Java API 文档

您可以在线查看 [Java API 文档](#) 。或者，在构建 **Java** 客户端之后，**Java API** 文档可以在 **java/kudu-client/target/apidocs/index.html** 中找到。

## 工作实例

**kudu 示例 Github** 仓库中提供了几个示例应用程序。每个示例包括一个自述文件，显示如何编译和运行它。这些例子说明了 **Kudu API** 的正确使用，以及如何设置虚拟机来运行 **Kudu** 。以下列表包括今天可用的一些示例。检查存储库本身，以防此列表过期。

### java/java-example

连接到 **Kudu** 实例的简单 **Java** 应用程序创建一个表，向其中写入数据，然后丢弃该表。

### java/collectl

侦听 **TCP** 套接字的小型 **Java** 应用程序，用于与 **Collectl** 有线协议相对应的时间序列数据。常用的 **collectl** 工具可用于将示例数据发送到服务器。

### java/insert-loadgen

生成随机插入负载的 **Java** 应用程序。

## python/dstat-kudu

一个示例程序，显示如何使用 **Kudu Python API** 将数据加载到由外部程序生成的新的/现有的 **Kudu** 表中，**dstat** 在这种情况下。

## python/graphite-kudu

使用 **graphite-web** ( 石墨网 ) 与 **Kudu** 作为后端的实验插件。

## demo-vm-setup

脚本下载并运行带有 **Kudu** 的 **VirtualBox** 虚拟机已安装。有关详细信息，请参阅 [快速入门](#) 。

这些例子应该是您自己的 **Kudu** 应用程序和集成的有用起点。

## Maven Artifacts ( Maven 工件 )

以下 **Maven <dependency>** 元素对于 **Apache Kudu** 公开版本（从 **1.0.0** 开始）是有效的：

```
<dependency>
  <groupId>org.apache.kudu</groupId>
  <artifactId>kudu-client</artifactId>
  <version>1.1.0</version>
</dependency>
```

**Java** 客户端和各种 **Java** 集成（例如 **Spark** ， **Flume** ）的便利二进制文件现在也可以通过 [ASF Maven 存储库](#) 和 [Maven Central 存储库](#) 获得。

## Impala命令使用 Kudu 的例子

请参阅 [使用 Impala 与 Kudu](#) 进行有关使用 **Kudu** 安装和使用 **Impala** 的指导，其中包括几个 **impala-shell** 示例。

## Kudu 与 Spark 集成

**Kudu** 从 **1.0.0** 版开始，通过 **Data Source API** 与 **Spark** 集成。使用 **-packages** 选项包括 **kudu-spark** 依赖关系：

如果使用 **Spark** 与 **Scala 2.10**，请使用 **kudu-spark\_2.10** 插件：

```
spark-shell --packages org.apache.kudu:kudu-spark_2.10:1.1.0
```

如果在 **Scala 2.11** 中使用 **Spark 2**，请使用 **kudu-spark2\_2.11** 插件：

```
spark-shell --packages org.apache.kudu:kudu-spark2_2.11:1.1.0
```

然后导入 **kudu-spark** 并创建一个数据框：

```
import org.apache.kudu.spark.kudu._
import org.apache.kudu.client._
import collection.JavaConverters._

// Read a table from Kudu
val df = sqlContext.read.options(Map("kudu.master" -> "kudu.master"))

// Query using the Spark API...
df.select("id").filter("id" >= 5).show()

// ...or register a temporary table and use SQL
df.registerTempTable("kudu_table")
val filteredDF = sqlContext.sql("select id from kudu_table where id >= 5")

// Use KuduContext to create, delete, or write to Kudu tables
val kuduContext = new KuduContext("kudu.master:7051", sqlContext.sparkContext)

// Create a new Kudu table from a dataframe schema
// NB: No rows from the dataframe are inserted into the table
kuduContext.createTable(
  "test_table", df.schema, Seq("key"),
  new CreateTableOptions()
    .setNumReplicas(1)
```

```

        .addHashPartitions(List("key").asJava, 3))

// Insert data
kuduContext.insertRows(df, "test_table")

// Delete data
kuduContext.deleteRows(filteredDF, "test_table")

// Upsert data
kuduContext.upsertRows(df, "test_table")

// Update data
val alteredDF = df.select("id", $"count" + 1)
kuduContext.updateRows(filteredRows, "test_table")

// Data can also be inserted into the Kudu table using the data source
// NB: The default is to upsert rows; to perform standard inserts
// NB: Only mode Append is supported
df.write.options(Map("kudu.master" -> "kudu.master:7051", "kudu.table" -> "test_table"))

// Check for the existence of a Kudu table
kuduContext.tableExists("another_table")

// Delete a Kudu table
kuduContext.deleteTable("unwanted_table")

<> and OR predicates are not pushed to Kudu, and instead will be evaluated in Spark

```

## Spark 集成已知问题和限制

- 注册为临时表时，必须为具有大写字母或非 **ASCII** 字符的名称的 **Kudu** 表分配备用名称。
- 具有包含大写字母或非 **ASCII** 字符的列名称的 **Kudu** 表可能不与 **SparkSQL** 一起使用。 **Columns** 可能在 **Kudu** 更名为解决这个问题。
- **<>** 和 **OR** 谓词不被推送到 **Kudu**，而是将在 **Spark** 的 **evaluate** 过程中执行，只有具有后缀通配符的 **LIKE** 谓词才被推送到 **Kudu** 执行，这意味着 **LIKE“FOO%”** 语句被下推到 **Kudu**，而 **“FOO**

%**BAR**”这样的语句不会。

- **Kudu** 不支持 **Spark SQL** 支持的所有类型，例如 **Date**，**Decimal** 和复杂类型。
- **Kudu** 表只能在 **SparkSQL** 中注册为临时表。可能不会使用 **HiveContext** 查询 **Kudu** 表。

## Kudu Python 客户端

**Kudu Python** 客户端为 **C ++** 客户端 API 提供了一个 **Python** 友好的界面。下面的示例演示了如何使用部分 **Python** 客户端。

```
import kudu
from kudu.client import Partitioning
from datetime import datetime

# Connect to Kudu master server
client = kudu.connect(host='kudu.master', port=7051)

# Define a schema for a new table
builder = kudu.schema_builder()
builder.add_column('key').type(kudu.int64).nullable(False).primary
builder.add_column('ts_val', type_=kudu.unixtime_micros, nullable=
schema = builder.build()

# Define partitioning schema
partitioning = Partitioning().add_hash_partitions(column_names=['k

# Create new table
client.create_table('python-example', schema, partitioning)

# Open a table
table = client.table('python-example')

# Create a new session so that we can apply write operations
session = client.new_session()

# Insert a row
op = table.new_insert({'key': 1, 'ts_val': datetime.utcnow()})
session.apply(op)
```

```
# Upsert a row
op = table.new_upsert({'key': 2, 'ts_val': "2016-01-01T00:00:00.000000"})
session.apply(op)

# Updating a row
op = table.new_update({'key': 1, 'ts_val': ("2017-01-01", "%Y-%m-%d %H:%M:%S")})
session.apply(op)

# Delete a row
op = table.new_delete({'key': 2})
session.apply(op)

# Flush write operations, if failures occur, capture print them.
try:
    session.flush()
except kudu.KuduBadStatus as e:
    print(session.get_pending_errors())

# Create a scanner and add a predicate
scanner = table.scanner()
scanner.add_predicate(table['ts_val'] == datetime(2017, 1, 1))

# Open Scanner and read all tuples
# Note: This doesn't scale for large scans
result = scanner.open().read_all_tuples()
```

## 与 MapReduce ， YARN 和其他框架集成

**Kudu** 旨在与 **Hadoop** 生态系统中的 **MapReduce** ， **YARN** ， **Spark** 和其他框架集成。 请参阅 [RowCounter.java](#) 和 [ImportCsv.java](#) ， 了解可以对自己的集成进行建模的示例。 请继续关注未来使用 **YARN** 和 **Spark** 的更多示例。

# Kudu Schema Design ( 模式设计 )

原文链接：[http://kudu.apache.org/docs/schema\\_design.html](http://kudu.apache.org/docs/schema_design.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813632>

贡献者：小瑶 ApacheCN Apache中文网

## Apache Kudu Schema Design ( Apache Kudu 模式设计 )

**Kudu** 表具有与传统 **RDBMS** 中的表类似的结构化数据模型。模式设计对于实现 **Kudu** 的最佳性能和运行稳定性至关重要。每个工作负载都是独一无二的，没有一个最合适每个表的单一模式设计。本文档概述了 **Kudu** 的有效模式设计理念，特别注意与传统 **RDBMS** 模式所采用的方法不同的地方。

在高层次上，创建 **Kudu** 表有三个问题：[列设计](#)，[主键设计](#)和[分区设计](#)。其中只有分区对于熟悉传统的非分布式关系数据库的人来说将是一个新的概念。最后一节讨论 [改变现有表的模式](#)，以及关于模式设计的[已知限制](#)。

## The Perfect Schema ( 完美的模式 )

完美的模式将完成以下工作：

- 数据将以这样的方式进行分发，即读取和写入在 **tablet servers** 上均匀分布。这受到分区的影响。
- **Tablets** 将以均匀，可预测的速度增长，**tablets** 的负载将随着时间



的推移而保持稳定。这最受影响分区。

- 扫描将读取完成查询所需的最少数据量。这主要受到主键设计的影响，但分区也通过分区修剪来起作用。

完美的模式取决于数据的特性，需要做的事情以及集群的拓扑。模式设计是您最大限度地发挥 **Kudu** 集群性能的最重要的一件事。

## Column Design ( 列设计 )

**Kudu** 表由一个列或多个列组成，每个列都有一个定义的类型。不属于主键的列可能为空。支持的列类型包括：

- boolean
- 8-bit signed integer
- 16-bit signed integer
- 32-bit signed integer
- 64-bit signed integer
- unixtime\_micros (从 UNIX 时代起，64 位微秒)
- single-precision (32-bit) IEEE-754 floating-point number ( 单精度，32位，IEEE-754 浮点数 )
- double-precision (64-bit) IEEE-754 floating-point number ( 双精度，64位，IEEE-754 浮点数 )
- UTF-8 encoded string (up to 64KB uncompressed) ( UTF-8 编码字符串（高达 64KB 未压缩） )
- binary (up to 64KB uncompressed) ( 二进制（高达 64KB 未压缩） )

**Kudu** 利用强类型的列和柱状磁盘存储格式来提供高效的编码和序列化。为了充分利用这些功能，列应该被指定为适当的类型，而不是使用

字符串或二进制列来模拟 '**schemaless**' 表，否则可能会进行结构化。除了编码之外，Kudu 允许按照每列进行压缩。

## Column Encoding ( 列编码 )

可以根据列的类型，使用编码创建 **Kudu** 表中的每个列。

表 1. **Encoding type** ( 编码类型 )

Column Type ( 列类型 )	Encoding ( 编码 )	Default ( 默认 )
int8, int16, int32	plain, bitshuffle, run length	bitshuffle
int64, unixtime_micros	plain, bitshuffle, run length	bitshuffle
float, double	plain, bitshuffle	bitshuffle
bool	plain, run length	run length
string, binary	plain, prefix, dictionary	dictionary

## plain Encoding ( 普通编码 )

数据以其自然格式存储。例如，**int32 values** 作为固定大小的 **32 位 little-endian integers** 存储。

## Bitshuffle Encoding

重新排列一组值以存储每个值的最高有效位，其次是每个值的第二个最高有效位，依此类推。最后，结果是 **LZ4** 压缩。**Bitshuffle** 编码对于具有许多重复值的列或按主键排序时少量更改的列是不错的

选择。 **bithuffle** 项目对性能和用例有很好的概述。

## Run Length Encoding ( 运行长度编码 )

通过仅存储值和计数，在列中压缩运行（连续重复值）。当按主键排序时，运行长度编码对于具有许多连续重复值的列是有效的。

## Dictionary Encoding ( 字典编码 )

构建了唯一值的字典，并且每个列值都被编码为字典中的相应索引。字典编码对于基数较低的列是有效的。如果给定行集的列值由于唯一值的数量太高而无法压缩，则 **Kudu** 将透明地回退到该行集的纯编码。这在 **flush** 过程中进行评估。

## Prefix Encoding ( 前缀编码 )

公共前缀在连续列值中进行压缩。前缀编码对于共享公共前缀的值或主键的第一列可能有效，因为行按 **tablet** 内的主键排序。

## Column Compression ( 列压缩 )

**Kudu** 允许使用 **LZ4**，**Snappy** 或 **zlib** 压缩编解码器进行每列压缩。默认情况下，列未压缩存储。如果减少存储空间比原始扫描性能更重要，请考虑使用压缩。

## Primary Key Design ( 主键设计 )

每个 **Kudu** 表必须声明一个主键索引由一个或多个列组成。主键列必须不可为空，并且可能不是布尔值或浮点型。在表创建期间设置后，主键中的列集合可能不会更改。像 **RDBMS** 主键一样，**Kudu** 主键强制执行唯一性约束;尝试插入与现有行具有相同主键值的行将导致重复的键错误。

与 **RDBMS** 不同，**Kudu** 不提供自动递增列功能，因此应用程序必须始终在插入期间提供完整的主键。行删除和更新操作还必须指定要更改的行的完整主键；**Kudu** 本身不支持范围删除或更新。在插入行之后，列的主键值可能不会被更新；但是，可以删除该行并重新插入更新的值。

## Primary Key Index ( 主键索引 )

与许多传统关系数据库一样，**Kudu** 的主键是 **clustered index** ( 聚集索引 )。 **tablet** 中的所有行都保持主键排序顺序。在主键上指定相等或范围约束的 **Kudu** 扫描将自动跳过不能满足谓词的行。这允许通过在主键列上指定相等约束来有效地找到各行。

注意

主键索引优化适用于单个 **tablet** 上的扫描。有关扫描如何使用谓词跳过整个 **tablet** 的详细信息，请参阅 [分区修剪](#) 部分。

## Partitioning ( 分区 )

为了提供可扩展性，**Kudu** 表被划分为称为 **tablets** 的单元，并分布在许多 **tablet servers** 上。行总是属于单个 **tablet** 。将行分配给 **tablet** 的方法由在表创建期间设置的表的分区决定。

选择分区策略需要了解数据模型和表的预期工作负载。对于写入繁重的工作负载，重要的是设计分区，以使写入分散在 **tablet** 上，以避免单个 **tablet** 的超载。对于涉及许多短扫描的工作负载，如果所有扫描的数据位于同一个 **tablet** 中，那么远程服务器的开销占主导地位，可以提高性能。了解这些基本权衡是设计有效的分区模式的核心。

重要

没有默认分区

**Kudu** 在创建表时不提供默认分区策略。建议预期具有较大读写工作负荷的新表至少与 **tablet servers** 一样多的 **tablets** 。

**Kudu** 提供了两种类型的分区：**range partitioning (范围分区)** 和 **hash partitioning (哈希分区)**。表也可能具有 **多级分区**，其结合范围和哈希分区，或多个散列分区实例。

## Range Partitioning (范围分区)

范围分区使用完全有序的分区键分配行。每个分区分配了范围分区密钥空间的连续段。密钥必须由主键列的子集组成。如果范围分区列与主键列匹配，则行的范围分区键将等于其主键。在没有哈希分区的范围分区表中，每个范围分区将对应于一个 **tablet** 。

在创建表时，将初始的范围分区集合指定为一组分区边界和拆分行。对于每个绑定，将在表中创建一个范围分区。每个分割将两个范围分割。如果没有指定分区界限，则表将默认为覆盖整个密钥空间的单个分区（下面和下面无界限）。范围分区必须始终不重叠，拆分行必须在范围分区内。

重要

请参阅 [范围分区示例](#)，以进一步讨论范围分区。

## Range Partition Management (范围分区管理)

**Kudu** 允许在运行时从表中动态添加和删除范围分区，而不影响其他分区的可用性。删除分区将删除属于分区的 **tablet** 以及其中包含的数据。随后插入到丢弃的分区将失败。可以添加新分区，但不能与任何现有的

分区重叠。 **Kudu** 允许在单个事务性更改表操作中删除并添加任意数量的范围分区。

动态添加和删除范围分区对于时间序列使用情况尤其有用。随着时间的推移，可以添加范围分区以覆盖即将到来的时间范围。例如，存储事件日志的表可以在每月开始之前添加一个月份的分区，以便保存即将到来的事件。必要时可以删除旧范围分区，以便有效地删除历史数据。

## Hash Partitioning ( 哈希分区 )

哈希分区通过哈希值将行分配到许多 **buckets** ( 存储桶 )之一。在 **single-level hash partitioned tables** ( 单级散列分区表 )中，每个 **bucket** ( 存储桶 )将对应于一个 **tablet**。在创建表时设置桶数。通常，主键列用作散列的列，但与范围分区一样，可以使用主键列的任何子集。

哈希分区是一种有效的策略，当不需要对表进行有序访问时。哈希分区对于在 **tablet** 之间随机散布这些功能是有用的，这有助于减轻热点和 **tablet** 大小不均匀。

注意

有关哈希分区的进一步讨论，请参阅 [哈希分区示例](#)。

## Multilevel Partitioning ( 多级分区 )

**Kudu** 允许一个表在单个表上组合多级分区。零个或多个哈希分区级别可以与可选的范围分区级别组合。除了单个分区类型的约束之外，多级分区的唯一附加约束是多个级别的哈希分区不能对相同的列进行哈希。

当正确使用时，多级分区可以保留各个分区类型的优点，同时减少每个分区的缺点。多级分区表中的 **tablet** 总数是每个级别中分区数量的乘积。

注意

请参阅 [哈希和范围分区示例](#) 以及 [哈希和哈希分区示例](#)，以进一步讨论多级分区。

## Partition Pruning ( 分区修剪 )

当可以确定分区可以被扫描谓词完全过滤时，**Kudu** 扫描将自动跳过扫描整个分区。要修剪哈希分区，扫描必须在每个散列列上包含相等谓词。要修剪范围分区，扫描必须在范围分区列上包含相等或范围谓词。对多级分区表的扫描可以独立地利用任何级别的分区修剪。

## Partitioning Examples ( 分区示例 )

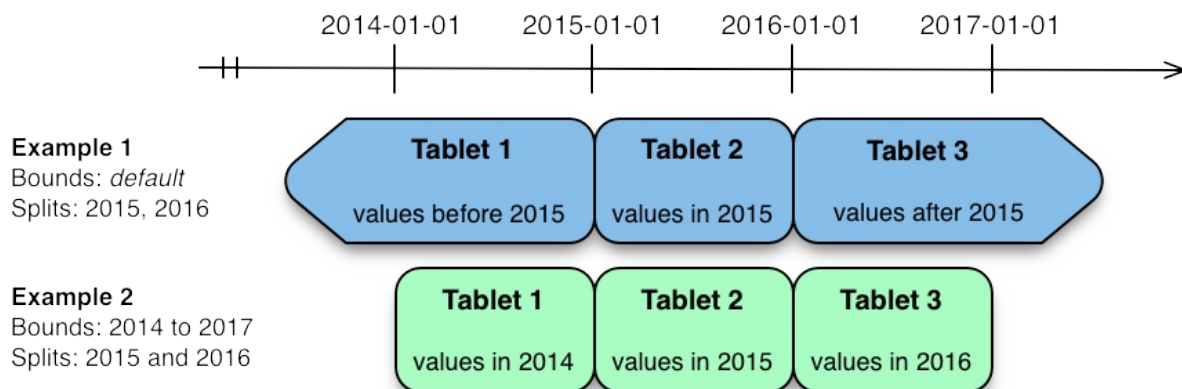
为了说明与为表设计分区策略相关的因素和权衡，我们将通过一些不同的分区方案。考虑存储机器度量数据的下表模式（为了清楚起见，使用 **SQL** 语法和日期格式的时间戳）：

```
CREATE TABLE metrics (  
    host STRING NOT NULL,  
    metric STRING NOT NULL,  
    time INT64 NOT NULL,  
    value DOUBLE NOT NULL,  
    PRIMARY KEY (host, metric, time),  
);
```

## Range Partitioning Example ( 范围分区示例 )

分割度量表的一种自然方式是在时间列上对范围进行分区。假设我们想要每年都有一个分区，该表将保存 **2014** 年，**2015** 年和 **2016** 年的数

据。表格至少有两种方式可以被分割：具有无界范围的分区，也可以是有限范围的分区。



上图显示了度量表可以在时间列上进行范围分区的两种方式。在第一个示例（蓝色）中，使用默认范围分区边界，并在 **2015-01-01** 和 **2014** 年 **1** 月分隔。这导致三个 **tablet**：**2015** 年之前的第一个值，**2015** 年的第二个值，以及 **2016** 年以后的第三个值。第二个例子（绿色）使用了 **[(2014-01-01), (2017-01-01)]**，并分期于 **2015-01-01** 和 **2016-01-01**。第二个例子可以等价地通过 **[(2014-01-01), (2015-01-01)]**，**[(2015-01-01), (2016-01-01)]** 和 **[(2016-01-01), (2017-01-01)]**，没有分裂。第一个例子有无界的下限和上限分区，而第二个例子包括边界。

上述范围分区示例中的每一个都允许有时限的扫描来修剪掉在扫描时间限制之外的分区。当有很多分区时，这可以大大提高性能。写作时，两个例子都有潜在的热点问题。由于度量趋向于始终在当前时间写入，大多数写入将进入单个范围分区。

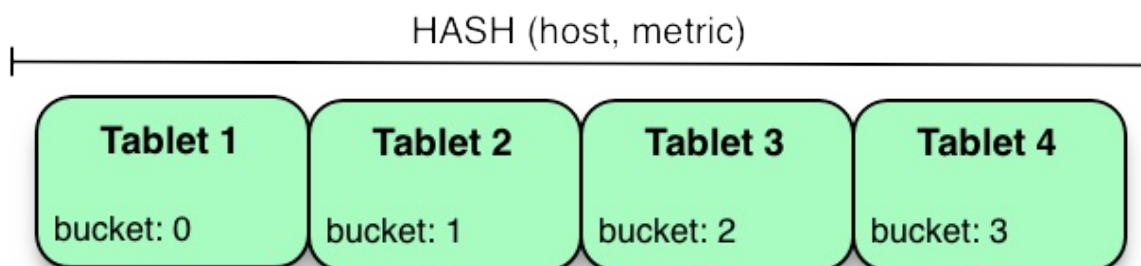
第二个例子比第一个例子更灵活，因为它允许未来几年的范围分区添加到表中。在第一个示例中，在 **2016-01-01** 之后的时间的所有写入将落入最后一个分区，因此分区可能最终变得太大，以致于单个 **tablet**



**servers** 无法处理。

## Hash Partitioning Example ( 哈希分区示例 )

分割 **metrics table** ( 度量表 ) 的另一种方法是在 **host** 和 **metric columns** ( 度量列 ) 上哈希分区。



在上面的示例中，**metrics table** 是将 **host** 和 **metric columns** 上的哈希分区分分为四个存储桶。与前面的范围分区示例不同，此分区策略将均匀地在表中的所有 **tablets** 上传播写入，这有助于总体写入吞吐量。通过指定相等谓词，扫描特定的 **host** 和 **metric** 可以利用分区修剪，将扫描的 **tablet** 数量减少到一个。使用纯哈希分区策略要小心的一个问题是，随着越来越多的数据被插入到表中，**tablet** 可以无限期地增长。最终 **tablet** 将变得太大，无法让个人 **tablet servers** 持有。

注意

虽然这些示例编号为 **tablet**，但实际上 **tablet** 只提供 **UUID** 标识符。哈希分区表中的 **tablet** 之间没有自然排序。

## Hash and Range Partitioning Example ( 哈希和范围分区示例 )

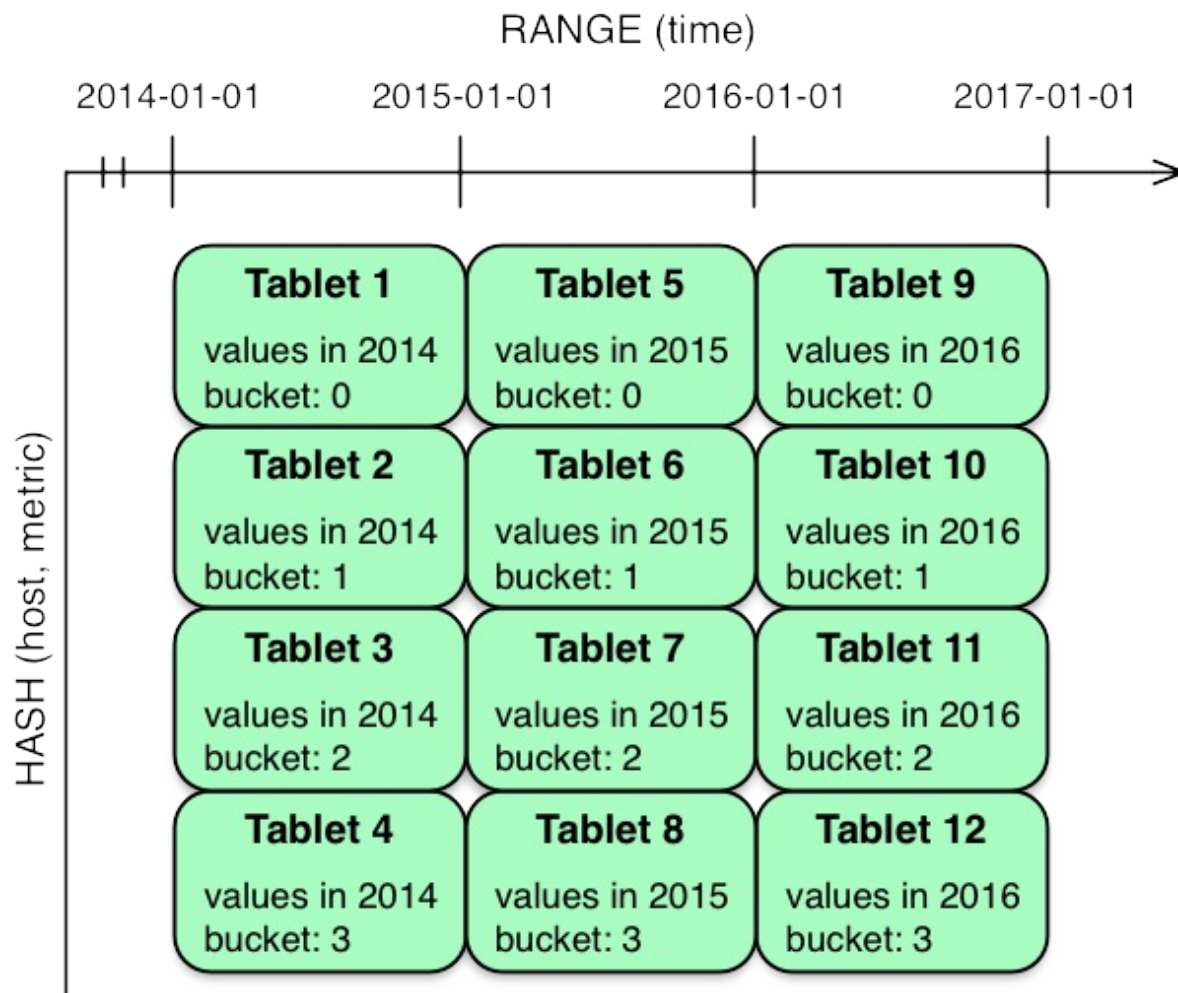
以前的示例显示了 **metrics table** 如何在 **time column** ( 时间列 ) 上进行范围分区，或者在 **host** 和 **metrics column** 上进行哈希分区。这些

策略具有相关的实力和弱点：

表 2. **Partitioning Strategies** ( 分区策略 )

Strategy ( 策略 )	Writes ( 写入 )	Reads ( 读取 )	Tablet Growth ( tablet 增长 )
range(time)	✗ - 所有写入到最新分区	✓ - 可以修剪与时间绑定的 <b>scan</b>	✓ - 可以在未来的时间段添加新的 <b>tablets</b>
hash(host, metric)	✓ - 在 <b>tablets</b> 上均匀分布	✓ - 可以修剪对特定 <b>hosts</b> 和 <b>metrics</b> 的 <b>scan</b>	✗ - <b>tablets</b> 可以增长到很大

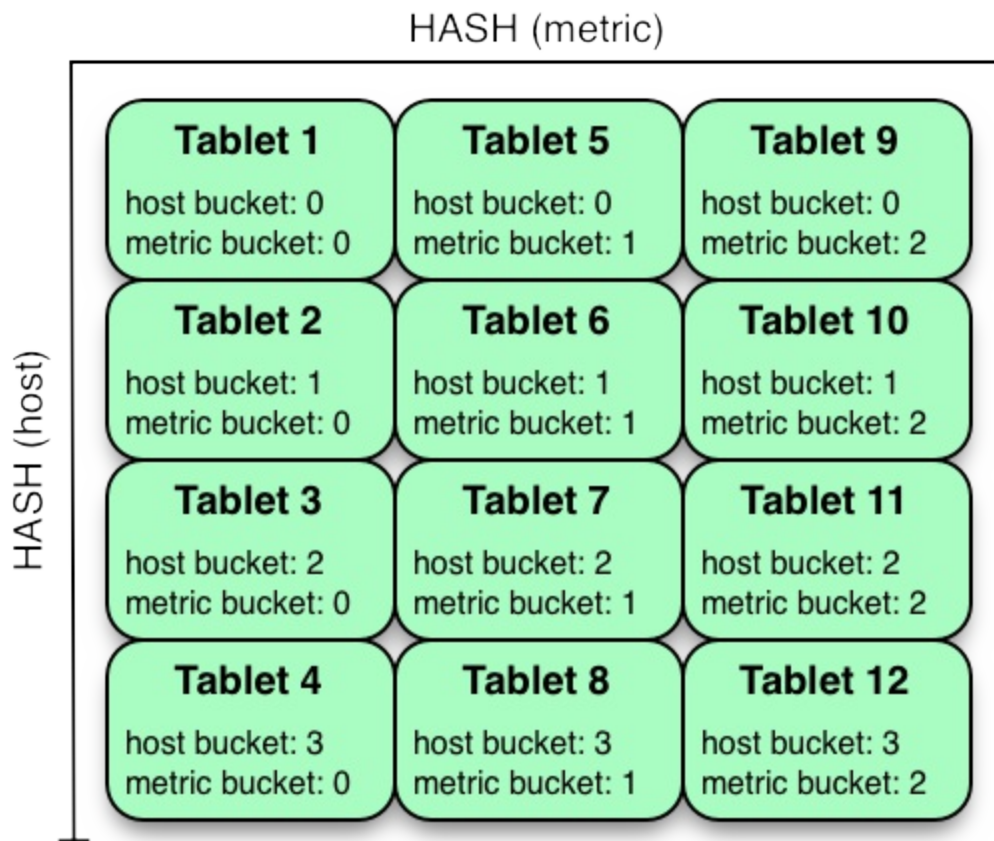
哈希分区有利于最大限度地提高写入吞吐量，而范围分区可避免 **tablet** 无限增长的问题。这两种策略都可以利用分区修剪来优化不同场景中的扫描。使用多级分区，可以组合这两种策略，以获得两者的优点，同时最大限度地减少每个策略的缺点。



在上面的示例中，**time column** 上的范围分区与 **host** 和 **metric columns** 上的哈希分区相结合。这个策略可以被认为具有二维划分：一个用于哈希级别，一个用于范围级别。在当前时刻写入此表将并行化到哈希桶的数量，在 **4** 这种情况下。读取可以利用时间限制和特定的 **host** 和 **metric** 谓词来修剪分区。可以添加新的范围分区，这将导致创建 **4** 个额外的 **tablet**（好像新列已添加到图表中）。

### Hash and Hash Partitioning Example ( 哈希和哈希示例 )

**Kudu** 可以在同一个表中支持任意数量的哈希分区级别，只要这些级别没有共同的 **hashed columns** ( 散列列 )。



在上面的示例中，表是主机上的哈希分区，分为 **4** 个桶，散列在公制中分区为 **3** 个桶，从而产生12个 **tablet**。尽管在使用此策略时，所有 **tablets** 中的写入将趋向于传播，但是比单独的 **host** 或 **metric** 的所有值始终属于单个 **tablet** 时，与多个独立列的哈希分区相比，它更容易受到热点查找。扫描可以分别利用 **host** 和 **metric columns** 上的等式谓词来修剪分区。

多级哈希分区也可以与范围分区相结合，逻辑上增加了分区的另一个维度。

## Schema Alterations ( 模式变更 )

您可以通过以下方式更改表的模式：

- 重命名表
- 重命名主键列
- 重命名，添加或删除非主键列
- 添加和删除范围分区

可以在单个事务操作中组合多个更改步骤。

注意

重命名主键列

**KUDU-1626** : **Kudu** 还不支持重命名主键列。

## Known Limitations ( 已知限制 )

**Kudu** 目前有一些已知的限制，可能会影响模式设计。

### Number of Columns ( 列数 )

默认情况下，**Kudu** 不允许创建 **300** 列以上的表。我们建议使用较少列的模式设计来获得最佳性能。

### Size of Cells ( 单元格大小 )

在编码或压缩之前，没有单个单元格可能大于 **64KB** 。组合密钥的单元在 **Kudu** 完成的内部复合密钥编码之后，总共限制在 **16KB** 。插入不符合这些限制的行将导致错误返回给客户端。

### Size of Rows ( 行大小 )

尽管单个单元格可能高达 **64KB** ，而 **Kudu** 最多支持 **300** 列，但建议不要单行大于几百 **KB** 。

## **Valid Identifiers ( 有效标识符 )**

诸如表和列名称的标识符必须是有效的 **UTF-8** 序列，不超过 **256** 个字节。

## **Immutable Primary Keys ( 不可变的主键 )**

**Kudu** 不允许您更新一行的主键列。

## **Non-alterable Primary Key ( 不可更改的主键 )**

表创建后，**Kudu** 不允许您更改主键列。

## **Non-alterable Partitioning ( 不可更改的分区 )**

**Kudu** 不允许您更改创建后如何分区表，但添加或删除范围分区除外。

## **Non-alterable Column Types ( 不可更改的列类型 )**

**Kudu** 不允许更改列的类型。

## **Partition Splitting ( 分区切分 )**

表创建后，分区不能拆分或合并。

# Kudu Security ( 安全 )

原文链接：<http://kudu.apache.org/docs/security.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pageId=10813635>

贡献者：小瑶，[ApacheCN](#)，[Apache中文网](#)

**Kudu** 包括安全功能，它可以让集群更安全以防止未授权的用户访问。本指南介绍了 **Kudu** 提供的安全功能，列出了必要的配置选项。[Known Limitations](#) 包含了 **Kudu** 中安全兼容性方面不足的列表。

## Authentication (认证)

**Kudu** 可以配置为在服务器之间和客户端和服务端之间实施安全认证。身份验证可防止不受信任的行为者访问 **Kudu**，并安全地识别连接的用户或服务以进行授权检查。**Kudu** 的认证旨在通过使用 **Kerberos** 与其他安全的 **Hadoop** 组件进行互操作。

可以在 **Kudu servers** 上使用 `--rpc-authentication` 标记来配置认证，它可以被设置为 `required`，`optional`，或 `disabled`。默认情况下，该标记设置为 `optional`。当设置为 `required` 时，**Kudu** 将拒绝客户端和缺少身份验证凭证的服务器的连接。当设置为 `optional` 时，**Kudu** 将尝试使用强身份验证。当配置为 `disabled` 或针对 “`optional`” 的强认证失败时，默认情况下，**Kudu** 将只允许来自受信任子网的未经身份验证的连接，它们是非私有网络

(`127.0.0.0/8`, `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`, `169.254.0.0/16`) 和本地网络接口的本地子网。来自可公开路由的 **IP** 的未认证连接

将被拒绝。

可信任的子网可以使用 `--trusted_subnets` 标记进行配置，该标记可以设置为以逗号分隔的 **CIDR** 表示法的 **IP** 块。将其设置为“**0.0.0.0/0**”以允许来自所有远程 **IP** 地址的未经身份验证的连接。但是，如果网络访问不受防火墙的限制，恶意用户可能会获得未经授权的访问。如果认证被配置为 **required**，这可以减少这样的行为。

警告：

当 `--rpc-authentication` 标记设置为 **optional** 时，该集群不会阻止来自未授权用户的访问请求。要保护集群，请使用 `--rpc-authentication=required`。

## Internal PKI

**Kudu** 使用内部 **PKI** 系统向集群中的服务器发出 **X.509** 证书。具有获得证书的对等体之间的连接将使用 **TLS** 进行身份验证，这不需要联系 **Kerberos KDC**。这些证书 **only**（仅仅）用于 **Kudu** 服务器之间以及 **Kudu** 客户端和服务器之间的内部通信。这些证书永远不会在面向公众的协议中呈现。

通过使用内部颁发的证书，**Kudu** 提供强大的身份验证功能，可扩展到巨大的群集，并允许使用 **TLS** 加密，而无需在每个节点上手动部署证书。

## Authentication Tokens（认证令牌）

在对安全的群集进行身份验证后，**Kudu** 客户端将自动向 **Kudu**主机请求一个身份验证令牌。认证令牌封装认证用户的身份，并携带 **master** 的 **RSA** 签名，以便验证其真实性。



该令牌将用于验证后续连接。默认情况下，身份验证令牌仅在七天内有效，因此即使令牌遭到入侵，也不能无限期地使用令牌。在大多数情况下，身份验证令牌应对用户完全透明。通过使用身份验证令牌，**Kudu** 可以利用强大的身份验证功能，而无需支付与每个连接的中央机构进行通信的可扩展性成本。

当与分布式计算框架（如 **Spark**）一起使用时，身份验证令牌可以简化配置并提高安全性。例如，**Kudu Spark connector**（连接器）将在规划阶段自动检索认证令牌，并将令牌分发到任务。这样，**Spark** 可以对受保护的 **Kudu** 集群工作，只有 **planner node** 具有 **Kerberos** 凭据。

## Scalability（可扩展性）

**Kudu** 认证旨在扩展到数千个节点，这需要避免与中央认证机构（如 **Kerberos KDC**）的不必要的协调。相反，**Kudu** 服务器和客户端将使用 **Kerberos** 与 **Kudu** 主机建立初始信任，然后使用备用凭据进行后续连接。特别是，**master** 将向服务器发出内部 **X.509** 证书，向客户端发出临时认证令牌。

## Encryption（加密）

**Kudu** 允许使用 **TLS** 对服务器之间以及客户端和服务器之间的所有通信进行加密。

可以在 **Kudu servers** 上使用 `--rpc-encryption` 标记来配置加密，它可以被设置为 `required`，`optional` 或 `disabled`。默认情况下，该标志设置为 `optional`。当配置为 `required` 时，**Kudu** 将拒绝未加密的连接。当配置为 `optional` 时，**Kudu** 将尝试使用加密。与认证相同，当配置为 `disabled` 或针对 `optional` 的情况下加密失败，**Kudu** 将只允许来自受信任的子网的未加密的连接，并拒绝公开可路由的 IP 的任何未加密的连接。要保护群集，请使用 `--rpc-encryption=required`。

警告：

**Kudu** 将自动关闭本地环回连接上的加密，因为来自这些连接的流量不会从外部暴露出来。这允许像 **Spark** 和 **Impala** 这样的位置感知计算框架避免加密开销，同时确保数据的机密性。

## Coarse-Grained Authorization（粗粒度授权）

**Kudu** 支持基于认证的客户端 **Kerberos** 主体（即用户或服务）对客户端请求进行粗粒度授权。可以配置的两个级别的访问是：

- **Superuser**（超级用户）- 被授权为超级用户的 **principals** 能够执行某些管理功能，例如使用 `kudu` 命令行工具诊断或修复群集问题。
- **User**（用户）- 授权为用户 **principals** 都能够在 **Kudu** 集群中访问和修改所有数据。这包括 **create**（创建），**drop**（删除）和 **alter**（更改）表以及 **read**（读取），**insert**（插入），**update**（更新）和 **delete**（删除）数据的功能。

警告：

在内部，**Kudu** 针对守护进程本身具有第三的访问级别。这样可以确保用户无法连接到集群，并可以作为 **tablet server**。

使用白名单样式的访问控制列表（**ACL**）授予访问级别，对于两个级别中的每一级都可以访问。每个访问控制列表指定以逗号分隔的用户列表，或者可以设置 `*` 为表示所有经过身份验证的用户能够以指定级别获得访问权限。有关示例，请参阅下面的 [配置安全的 Kudu 群集](#)。

警告：

用户 **ACL** 的默认值是 `*` 允许所有用户访问集群。但是，如果启用了身份验证，这仍然限制对只能通过 **Kerberos** 成功验证的用户的访问。与 **Kudu server** 在同一网络上的未认证用户将无法访问集群。

## Web UI Encryption（加密）

可以通过为每个服务器提供 **TLS** 证书，将 **Kudu Web UI** 配置为使用安全的 **HTTPS** 加密。有关 **Web UI HTTPS** 配置的更多信息，请参阅 [配置安全的 Kudu 集群](#)。

## Web UI Redaction

为了防止在 **Web UI** 中暴露敏感数据，所有行数据都被修改。表元数据，如表名，列名和分区信息不会被修改。可以通过在 **Kudu server** 上设置 `--webserver-enabled=false` 标记来完全禁用 **Web UI**。

警告：

禁用 **Web UI** 也将禁用 **REST endpoints**，例如 `/metrics`。监控系统依靠这些 **endpoints** 来收集 **metrics** 指标数据。

## Log Security（日志安全）

为了防止敏感数据包含在 **Kudu** 服务器日志中，默认情况下会修改所有行数据。该功能可以配置 `--redact` 标记来关闭。

## Configuring a Secure Kudu Cluster（配置安全的 Kudu 集群）

应在所有服务器（**master** 和 **tablet server**）上设置以下配置参数，以

确保 **Kudu** 集群安全：

```
# Connection Security
#-----
--rpc-authentication=required
--rpc-encryption=required
--keytab-file=<path-to-kerberos-keytab>

# Web UI Security
#-----
--webserver-certificate-file=<path-to-cert-pem>
--webserver-private-key-file=<path-to-key-pem>
# optional
--webserver-private-key-password-cmd=<password-cmd>

# If you prefer to disable the web UI entirely:
--webserver-enabled=false

# Coarse-grained authorization
#-----

# This example ACL setup allows the 'impala' user as well as the
# 'nightly_etl_service_account' principal access to all data in the
# Kudu cluster. The 'hadoopadmin' user is allowed to use administrative
# tooling. Note that, by granting access to 'impala', other users
# may access data in Kudu via the Impala service subject to its own
# authorization rules.
--user-acl=impala,nightly_etl_service_account
--superuser-acl=hadoopadmin
```

有关这些标记的更多信息，请参见配置标记指南。

## Known Limitations（已知限制）

**Kudu** 有一些已知的安全限制：

### Long-lived Tokens（常用的令牌）

**Java Kudu** 客户端在初始 **tokens**（令牌）到期后不会自动请求新的 **authn tokens**（令牌），因此不支持长期存在的安全集群中的

**Java** 客户端。然而，针对 **C** 的 **Kudu** 客户端会自动请求新的 **authn tokens** 令牌，因此支持安全集群中长久的 **C** 客户端（即超出 **authn token** 令牌生存期）。

### **Custom Kerberos Principal**（自定义 kerberos 主体）

**Kudu** 不支持为 **Kudu** 进程设置自定义服务 **principal**。**principal** 必须是 '**kudu**'。

### **External PKI**

**Kudu** 不支持外部颁发的内部线路加密证书（服务器到服务器和客户端到服务器）。细粒度授权 **Kudu** 无法根据操作类型或目标（表，列等）限制访问。**ACL** 目前不支持基于组中成员资格的授权。

### **On-disk Encryption**（磁盘加密）

**Kudu** 没有内置的磁盘加密。然而，**Kudu** 可以使用全盘加密工具，如 **dm-crypt**。

### **Web UI Authentication**（认证）

**Kudu Web UI** 缺少基于 **Kerberos** 的身份验证（**SPNEGO**），因此不能基于 **Kerberos** 的 **principals** 进行访问限制。

### **Flume Integration**（**Flume** 集成）

需要验证或加密的安全 **Kudu** 集群不支持 **Flume** 集成。

# Kudu Transaction Semantics ( 事务语义 )

原文链接 : [http://kudu.apache.org/docs/transaction\\_semantics.html](http://kudu.apache.org/docs/transaction_semantics.html)

译文链接 : <http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813638>

贡献者 : 小瑶 [ApacheCN](#) [Apache中文网](#)

## Apache Kudu 的 Transaction Semantics ( 事务语义 )

这是 **Kudu** 的交易和一致性语义的简要介绍。对于这里提到的大部分内容的深入技术论述，为什么是正确的，请参阅技术报告[1]。

**Kudu** 的 **transactional semantics** ( 事务语义 )和体系结构受到 **Spanner** [2] 和 **Calvin** [3] 等最先进的系统的启发。 **Kudu** 建立在数十年的数据库研究基础之上。核心理念是通过提供简单，强大的语义的事务，使开发人员的生活更轻松，而不会牺牲性能或调整到不同需求的能力。

**Kudu** 旨在最终完全获得 **ACID** ，但是多 **tablet** 事务尚未实施。因此，本次讨论的重点是单片写入操作，只是简单地触摸多 **tablets** 读取。最终 **Kudu** 将支持完全严格的可序列化语义。事实上，它已经在有限的范围内，但并不是所有的角落都被覆盖，因为这仍然是一个正在进行的工作。

**Kudu** 目前允许以下操作：

- **Write operations** ( 写入操作 ) 是在具有多个副本的单个 **tablet** 中在存储引擎中插入，更新或删除的 **sets of rows** ( 行集 )。写操作没有单独的 “**read sets**”，即它们在执行写入之前不扫描现有数据。每个写入仅涉及即将更改的行的先前状态。用户明确不写 “**committed**”。相反，它们将在系统完成后自动提交。
- **Scans** ( 扫描 ) 是可以遍历多个 **tablets** 并读取信息的读取操作，具有一致性或正确性保证。扫描可以执行时间行进读取，即用户能够设置过去的扫描时间戳，并返回在该时间点反映存储引擎的状态的结果。

重要

开始之前

- 多次提到术语时间戳来说明功能，但是时间戳是用户最不可见的内部概念，除了在 **KuduScanner** 上设置时间戳。
- 我们通常指的是 **C++ client** 的方法和类。虽然 **Java** 客户端大多具有类似的方法和类，但是 **API** 的确切名称可能不同。

## Single tablet write operations ( 单 tablet 写入操作 )

**Kudu** 采用 **Multiversion** 并发控制 ( **MVCC** ) 和 **Raft** 一致性算法 [4]。  
• **Kudu** 的每个写入操作必须经过 **tablet** 的领导。

1. **leader** 将为其将更改的行获取所有锁。
2. **leader** 在提交写入复制之前分配写入时间戳。这个时间戳是 **MVCC** 中的 **write** 的 “**tag**”。
3. 在大多数副本确认更改后，实际的行将被更改。
4. 更改完成后，它们可以以原子方式同时进行写入和读取。

**tablets** 的所有副本都遵循相同的操作顺序，如果为写入操作分配时间戳  $n$  并更改行  $x$ ，则保证时间戳  $m > n$  的第二次写入操作可以看到  $x$  的新值。

锁定采集和时间戳分配的这种严格排序通过协商一致的方式强制执行 **tablets** 的所有副本。因此，相对于同一个平板电脑中的其他写入，关于时钟分配的时间戳，写入操作是完全排序的。换句话说，写入具有严格的可序列化语义，尽管在一个公认的有限的上下文中。有关这些语义是什么意思，请参阅这篇 [博客文章](#)。

虽然在 **ACID** 意义上的隔离和耐用性，多行写入操作尚未完全原子化。批量操作中单次写入的故障不会回滚操作，而是产生每行错误。

## Writing to multiple tablets ( 写入多个 tablets )

**Kudu** 还不支持跨多个 **tablets** 的事务。但是，一致的快照读取是可能的（如当前实现中的警告），如下所述。

**Kudu** 客户端的写入可选地缓冲在内存中，直到它们被刷新并发送到服务器。当客户端的会话刷新时，每个 **tablet** 的行被批处理在一起，并发送到托管 **tablet** 的 **leader replica** 的 **tablet servers**。由于没有 **inter-tablet** 事务，这些批次中的每一个代表具有自己的时间戳的单独的独立写入操作。然而，客户端 **API** 提供了对分配的时间戳施加一些约束的选项，以及客户端如何观察对不同 **tablet** 的写入。

**Kudu** 像 **Spanner** 一样被设计为外部一致 [5]，即使在跨多个平板电脑甚至多个数据中心的情况下也能保持一致性。在实践中，这意味着如果写入操作在平板电脑 **A** 上更改项目  $x$ ，并且以下写入操作在平板电脑 **B** 上更改项目  $y$ ，则可能需要强制执行，如果观察到  $y$  的更改，则  $x$  的更改也必须为观察到的。有很多例子可以说是重要的。例如，如果



**Kudu** 正在存储用于进一步分析的点击流，并且两次点击相互关联，但存储在不同的 **tablet** 中，则后续点击应该被分配后续时间戳，以便捕获它们之间的因果关系。

## **CLIENT\_PROPAGATED Consistency**

**Kudu** 的默认外部一致性模式称为 **CLIENT\_PROPAGATED**。有关如何工作的详细说明，请参阅 [1]。简而言之，此模式会导致单个客户端的写入自动在外部保持一致。在上述 **Clickstream** 场景中，如果两个点击是由不同的客户端实例提交的，则应用程序必须手动将时间戳从一个客户端传播到另一个客户端，以便捕获因果关系。

客户端 **a** 和 **b** 之间的时间戳可以传播如下：

### **Java Client**

调用 **AsyncKuduClient#getLastPropagatedTimestamp()** 在客户端 **a**，将时间戳传播到客户端 **b**，并在客户端 **b** 上调用 **AsyncKuduClient#setLastPropagatedTimestamp()**。

### **C++ Client**

在客户端 **a** 上调用 **KuduClient::GetLatestObservedTimestamp()**，将时间戳传播到客户端 **b**，并在客户端 **b** 上调用 **KuduClient::SetLatestObservedTimestamp()**。

## **COMMIT\_WAIT Consistency**

**Kudu** 还有一个在 **Google** 的 **Spanner** 中使用的外部一致性模型的实验实现，称为 **COMMIT\_WAIT**。**COMMIT\_WAIT** 通过在集群中的所有计算机上紧密同步时钟来起作用。然后，当发生写入时，分配时间

戳，并且在通过足够的时间之前写入的结果不可见，使得集群中的其他计算机不可能为下一次写入分配较低的时间戳。

当使用此模式时，写入的延迟与所有集群主机上的时钟精度紧密相关，并且使用松散时钟同步的此模式会导致写入需要很长时间才能完成甚至超时。请参阅 [已知问题和限制](#)。

**COMMIT\_WAIT**一致性模式可以如下选择：

### Java Client

调用

**KuduSession#setExternalConsistencyMode(ExternalConsister**

### C++ Client

调用 **KuduSession ::**

**SetExternalConsistencyMode(COMMIT\_WAIT)**

注意

**COMMIT\_WAIT** 一致性被认为是一个实验功能。它可能会返回不正确的结果，展示性能问题或对集群稳定性产生负面影响。不鼓励在生产环境中使用。

## Read Operations (Scans) ( 阅读操作(扫描) )

扫描是由可能跨越一个或多个 **tablets** 跨越一行或多行的客户端执行的读取操作。当服务器接收到扫描请求时，会收到 **MVCC** 状态的快照，然后根据用户选择的读取模式，以两种方式之一进行。该模式可以如下选择：

## Java Client

调用 **KuduScannerBuilder#setReadMode(...)**

## C++ Client

调用 **KuduScanner :: SetReadMode()**

以下模式在两个客户端都可用：

### READ\_LATEST

这是默认的读取模式。服务器拍摄 **MVCC** 状态的快照，并立即继续阅读。在此模式下读取只会产生 “**Read Committed**” 隔离。

### READ\_AT\_SNAPSHOT

在这种读取模式下，扫描是一致的和可重复的。快照的时间戳由服务器选择，或由用户通过 **KuduScanner :: SetSnapshotMicros()** 显式设置。建议明确设置时间戳；见 [Recommendations \( 建议书 \)](#)。服务器等待，直到这个时间戳为 “**safe**”（直到所有具有较低时间戳的写入操作都已完成并可见）。这种延迟加上外部一致性方法，最终将允许 **Kudu** 具有完全严格可序列化的读写语义。这仍然是一个正在进行的工作，一些异常仍然是可能的（请参阅 [已知问题和限制](#)）。只有在这种模式下进行扫描可以是容错的。

在读取模式之间进行选择需要平衡权衡并做出适合您工作负载的选择。例如，需要扫描整个数据库的报告应用程序可能需要执行仔细的计费操作，以便扫描可能需要容错，但可能不需要一个微微小的最新视图的数据库。在这种情况下，您可以选择 **READ\_AT\_SNAPSHOT**，并选择扫描开始时过去几秒的时间戳。另一方面，不吸收整个数据集并且

已经具有统计性质的机器学习工作负载可能不要求扫描是可重复的，因此您可以选择 **READ\_LATEST** 。

## Known Issues and Limitations ( 已知问题和限制 )

目前，在一些情况下，**Kudu** 已经完全严格序列化，存在几个缺陷和角落。以下是详细信息，接下来是一些建议。

### Reads (Scans) ( 阅读（扫描） )

- 对 **COMMIT\_WAIT** 的支持是实验性的，需要仔细调整时间同步协议，如 **NTP**（网络时间协议）。在生产环境中不鼓励使用它。

### Writes ( 写入 )

- 在 **leader change** 中，**READ\_AT\_SNAPSHOT** 扫描时间戳超出最后一次写入的快照，也可能会产生不可重复的读取（参见 [KUDU-1188](#)）。请参阅 有关解决方法的[建议](#)。
- **Impala** 扫描当前以 **READ\_LATEST** 的形式执行，并且没有一致性保证。
- 在 **AUTO\_BACKGROUND\_FLUSH** 模式下，或者使用“异步”刷新机制时，应用于单个客户端会话的写入可能由于将数据刷新到服务器的并发而重新排序。如果用不同的值连续快速更新单个行，这可能会特别明显。这种现象影响所有的客户端 **API** 实现。解决方法在 **API** 文档中介绍了 **FlushMode** 或 **AsyncKuduSession** 文档中各自的实现。见 [KUDU-1767](#)。

### Recommendations ( 建议 )

- 如果可重复的快照读取是必需的，请使用 **READ\_AT\_SNAPSHOT**

，其时间戳稍微在过去（在 **2-5** 秒之间，理想情况下）。这将绕过 **Reads**（**Scans**）中描述的异常。即使异常已被解决，回溯时间戳总是使扫描速度更快，因为它们不太可能被阻止。

- 如果需要外部一致性，并且您决定使用 **COMMIT\_WAIT**，则需要仔细调整时间同步协议。每个事务将在执行时等待 **2x** 最大时钟错误，通常为 **100** 毫秒至 **1** 秒范围与默认设置，也许更多。因此，交易将需要至少 **200** 毫秒至 **2** 秒使用默认设置完成甚至可能会超时。

\* 本地服务器应该用作时间服务器。我们使用 Google Compute Engine 数据中心提供的默认 **NTP** 时间源进行实验，并能够获得合理的最大误差范围，通常在 **12-17** 毫秒之间变化。

\* 应在 **/etc/ntp.conf** 中调整以下参数以收紧最大错误：

- 服务器 my\_server.org iburst minpoll 1 maxpoll 8
- tinker dispersion 500
- tinker allan 0

## 信息

上述参数以估计误差为代价最小化最大误差，后者可能高于“**normal**”值的数量级。这些参数也可能对时间服务器造成更大的负担，因为它们使得服务器的轮询频率更高。

## References ( 参考 )

- [1] David Alves, Todd Lipcon and Vijay Garg. Technical Report: HybridTime - Accessible Global Consistency with High Clock Uncertainty. April,

2014. <http://users.ece.utexas.edu/~garg/pdslab/david/hybrid-time-tech-report-01.pdf>

- [2] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's globally-distributed database. In Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12). USENIX Association, Berkeley, CA, USA, 251-264.
- [3] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 1-12.  
DOI=10.1145/2213836.2213838 <http://doi.acm.org/10.1145/221383>
- [4] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC'14), Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, Berkeley, CA, USA, 305-320.
- [5] Kwei-Jay Lin, "Consistency issues in real-time database systems," in System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International

Conference on , vol.2, no., pp.654-661 vol.2, 3-6 Jan 1989 doi:  
10.1109/HICSS.1989.48069

# 后台维护任务

原文链接：[http://kudu.apache.org/docs/background\\_tasks.html](http://kudu.apache.org/docs/background_tasks.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813641>

贡献者：小瑶 ApacheCN Apache中文网

## Apache Kudu 后台维护任务

**Kudu** 依赖于许多重要的自动维护活动运行后台任务。这些任务包括将数据从内存刷新到磁盘，压缩数据以提高性能，释放磁盘空间等。

### Maintenance manager ( 维护管理 )

**maintenance manager** 安排并运行后台任务。在任何给定的时间点，**maintenance manager** 根据当时所需的改进来确定下一个任务的优先级，例如减轻内存压力，提高读取性能或释放磁盘空间。通过设置 **--maintenance\_manager\_num\_threads** 可以控制专用于运行后台任务的工作线程数。

### Flushing data to disk ( 将数据刷新到磁盘 )

从内存到磁盘 **flush** 数据可以缓解内存压力，并可通过从 **MemRowSet** 中的写入优化的面向行的内存中格式切换到磁盘上读取优化的以列为单位的格式来提高读取性能。刷新数据的后台任务包括 **FlushMRSOp** 和 **FlushDeltaMemStoresOp**。

与这些操作相关联的 **metrics** 分别具有前缀 **flush\_mrs** 和 **flush\_dms**。



## Compacting on-disk data ( 压缩磁盘数据 )

**Kudu** 不断执行几种类型的压缩任务，以便在一段时间内保持一致的读写性能。将多个 **DiskRowSets** 组合到一个 **DiskRowSet** 中的合并压缩由 **CompactRowSetsOp** 运行。还可以运行两种类型的 **delta** 存储压缩操作：**MinorDeltaCompactionOp** 和 **MajorDeltaCompactionOp**。

有关这些不同类型的压实操作的更多信息，请参阅 [Kudu Tablet 设计文档](#)。

## Write-ahead log GC ( 预写日志 GC )

**Kudu** 每个 **tablet** 维护一个预先记录日志（**WAL**），分为固定大小的分段。当活动段达到配置大小（由 **--log\_segment\_size\_mb** 控制）时，**tablet** 会定期将 **WAL** 滚动到新的日志段。为了节省磁盘空间并减少启动时间，称为 **LogGCOp** 的后台任务尝试通过从本地节点不再需要它们的持久性而将其从磁盘中删除，从而对其进行垃圾收集（**GC**）旧的 **WAL** 段。

与此后台任务相关联的 **metrics** 具有前缀 **log\_gc**。

## Tablet history GC and the ancient history mark ( **tablet** 历史GC和古代历史记号 )

因为 **Kudu** 使用多重并发控制（**MVCC**）机制来确保快照扫描可以从新的更改隔离到一个表中，所以定期对旧的历史数据进行垃圾回收（删除）以释放磁盘空间。虽然 **Kudu** 不会删除在最新版本的数据中可见的行或数据，但 **Kudu** 会删除不再可见的旧更改的记录。

历史上的 **MVCC** 数据变得无法访问并被自由删除的时间点被称为古代

历史记录（ **AHM** ）。可以通过设置 **--tablet\_history\_max\_age\_sec** 来配置 **AHM** 。

有两个后台任务，**GC** 历史 **MVCC** 数据早于 **AHM**：运行合并压缩（称为 **CompactRowSetsOp**（见上））的后台任务，以及一个单独的后台任务，可以删除旧的撤消增量块，称为 **UndoDeltaBlockGCOp**。运行 **UndoDeltaBlockGCOp** 可减少所有工作负载中的磁盘空间使用情况，特别是在更新或升级版本较高的系统中。

与此后台任务相关联的 **metrics** 具有前缀 **undo\_delta\_block**。

# Kudu 配置参考

---

原文链接：[http://kudu.apache.org/docs/configuration\\_reference.html](http://kudu.apache.org/docs/configuration_reference.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813644>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## kudu-master Flags

### Stable Flags ( 稳定标志 )

标记为 **stable** ( 稳定 ) 而不是 **advanced** ( 高级 ) 的标志可以安全地用于常见配置任务。

#### **--block\_cache\_capacity\_mb**

块高速缓存容量 (MB)

<b>Type ( 类型 )</b>	int64
<b>Default ( 默认 )</b>	512
<b>Tags ( 标志 )</b>	stable

#### **--log\_force\_fsync\_all**

每次写入后，**Log/WAL** 是否应显式调用

<b>Type ( 类型 )</b>	bool
<b>Default ( 默认 )</b>	false
<b>Tags ( 标志 )</b>	stable

**--fs\_data\_dirs**

具有数据块的目录的逗号分隔列表。 如果没有指定，**fs\_wal\_dir** 将被用作唯一的数据块目录。

<b>Type ( 类型 )</b>	string
<b>Default ( 默认 )</b>	none
<b>Tags ( 标志 )</b>	stable

## 目录 2 - 标题 2

丢雷楼某、、、

## 目录 3 - 标题 2

丢雷楼某、、、

## 目录 3 - 标题 2

丢雷楼某、、、

# Kudu 命令行工具参考

---

原文链接：

[http://kudu.apache.org/docs/command\\_line\\_tools\\_reference.html](http://kudu.apache.org/docs/command_line_tools_reference.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813647>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## 目录 1 - 标题 2

丢雷楼某、、、

## 目录 2 - 标题 2

丢雷楼某、、、

## 目录 3 - 标题 2

丢雷楼某、、、

## 目录 3 - 标题 2

丢雷楼某、、、

# 已知的问题和限制

---

原文链接：[http://kudu.apache.org/docs/known\\_issues.html](http://kudu.apache.org/docs/known_issues.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813650>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## Schema ( 架构 )

### Primary keys ( 主键 )

- 创建表后，主键可能不会更改。您必须删除并重新创建表以选择新的主键。
- 构成主键的列必须首先列在模式中。
- 一行的主键可能不会使用 **UPDATE** 功能进行修改。要修改行的主键，必须删除该行并使用修改的密钥重新插入。这样的修改是非原子的。
- 不能使用 **DOUBLE**，**FLOAT** 或 **BOOL** 类型的列作为主键定义的一部分。另外，作为主键定义的一部分的所有列都必须为 **NULL**。
- 不支持自动生成的主键。
- 组合复合主键的单元在 **Kudu** 完成的内部复合密钥编码后，总共限制在16KB。

### Columns ( 列 )

- **DECIMAL**，**CHAR**，**VARCHAR**，**DATE** 和 **ARRAY** 等复杂类型不受支持。

- 不能通过更改表来更改现有列的类型和可空性。
- 表格最多可以有 **300** 列。

## Tables ( 表 )

- 表必须有奇数个副本，最多 **7** 个。
- 无法更改复制因子（在表创建时设置）。

## Cells (individual values) ( 单元格 ( 个体值 ))

- 在编码或压缩之前，单元格不能大于 **64KB** 。

## Other usage limitations ( 其他使用限制 )

- **Kudu** 主要用于分析用例。如果单行包含多个千字节的数据，则可能会遇到问题。
- 不支持辅助索引。
- 不支持多行事务。
- 不支持关系功能，如外键。
- 诸如列和表名称的标识符被限制为有效的 **UTF-8** 字符串。另外，执行最大长度为 **256** 个字符。
- 删除列不会立即回收空间。**Compaction** 必须先运行。
- 没有办法手动运行压缩，但删除表将立即回收空间。

## Partitioning Limitations ( 分区限制 )

- 表必须使用简单或复合主键手动预分割成 **tablets** 。自动拆分还不可能。创建表后，可能会添加或删除范围分区。有关详细信息，请参阅 [模式设计](#)。

- 现有表中的数据当前无法自动重新分区。作为解决方法，使用新的分区创建一个新表，并插入旧表的内容。
- 丢失大多数 **replicas** 的 **tablets** （例如 **3** 中剩下的 **1** 个）需要手动干预才能修复。

## Cluster management ( 集群管理 )

- **Rack awareness** ( 机架意识 ) 不受支持。
- 不支持 **Multi-datacenter** ( 多数据中心 )。
- 不支持 **Rolling restart** ( 滚动重新启动 )。

## Server management ( 服务器管理 )

- 生产部署应为 **tablet servers** 配置至少 **4GB** 的内存，理想情况下应为 **10GB** 以上。
- 写入日志（**WAL**）只能存储在一个磁盘上。
- 不能容忍磁盘故障，一旦检测到 **tablet server** 就会崩溃。
- 无法恢复数据的磁盘故障需要格式化该 **tablet server** 的所有 **Kudu** 数据，才能再次启动。
- 数据目录无法 添加/删除 ;所有这些都必须重新格式化以更改目录集。
- **tablet servers** 无法正常 **decommissioned** 。
- **tablet servers** 无法更改地址/端口。
- **Kudu** 对于最新的 **NTP** 具有严格的要求。 **Kudu**的**master** 和**tablet servers**在不同步时会崩溃。
- **Kudu** 版本只能使用 **NTP** 进行测试。其他时间同步提供商，如 **Chrony** 可能或可能不工作。

## Scale ( 规模 )



- 推荐的最大数量的 **tablet servers** 是**100**。
- 建议的最大 **masters** 数是 **3** 。
- 推荐的最大数量的存储数据，复制后和压缩后每个 **tablet server** 是 **4TB**。
- 每个 **tablet server** 的最大平均 **tablets** 推荐数量为 **1000** 次，**post-replication** 。
- 每个 **tablet server** 的每个表的最大 **tablet** 数是 **60**，复制后，创建表。

## Replication and Backup Limitations ( 复制和备份限制 )

- **Kudu** 当前不包括用于备份和还原的任何内置功能。鼓励用户使用 **Spark** 或 **Impala** 等工具，根据需要导出或导入表格。

## Security Limitations ( 安全限制 )

- 授权仅在全系统，粗粒度级别提供。表级，列级和行级授权功能不可用。
- 据报道，**Kudu** 已经报告在使用本地块设备加密（例如 **dmccrypt** ）的系统上正确运行 **Kudu** 。
- **Kudu** 服务器 **Kerberos principals** 必须遵循模式 **kudu/<HOST>@DEFAULT.REALM** 。不支持配置备用 **Kerberos principal** 。
- **Kudu** 与 **Apache Flume** 的集成不支持写入需要 **Kerberos** 认证的 **Kudu** 集群。
- 首次联系群集时，**Kudu** 客户端实例将检索身份验证令牌。这些令牌在一周后到期。使用单个 **Kudu** 客户端实例超过一周仅由 **C ++**

客户端支持，但不受 **Java** 客户端的支持。

## Other Known Issues ( 其他已知问题 )

以下是 **Kudu** 当前版本的已知错误和问题。他们将在以后的版本中解决。请注意，此列表并不详尽，仅用于沟通最重要的已知问题。

- 如果 **Kudu master** 配置了 **-log\_force\_fsync\_all** 选项，则 **tablet server** 和客户端将经历频繁的超时，并且集群可能无法使用。
- 如果 **tablet servers** 的 **tablet** 数量很多，可能需要几分钟才能启动。建议将每台服务器的 **tablet** 数量限制在 **100** 个以下。在分割表格时考虑这个限制。如果您注意到启动时间较慢，则可以监控 **Web UI** 中每个服务器的 **tablets** 数量。
- 在 **hostname** 中包含大写字母的 **hosts** 上，**Kerberos** 身份验证功能不正确。
- 如果在 **krb5.conf** 中配置了 **rdns = false**，则 **Kerberos** 身份验证功能不正常。

# 贡献于 Kudu

原文链接：<http://kudu.apache.org/docs/contributing.html>

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813653>

贡献者：小瑶 [ApacheCN](#) [Apache](#) 中文网

## Contributing Patches Using Gerrit ( 使用 Gerrit 贡献补丁 )

**Kudu** 团队使用 **Gerrit** 进行代码审查，而不是 **Github pull requests**。通常，您从 **Github pull**，但是 **push** 到 **Gerrit**，**Gerrit** 用于查看代码并将其合并到 **Github** 中。

有关使用 **Gerrit** 进行代码审查的概述，请参阅 [Gerrit 教程](#)。

### Gerrit 的初始设置

1. 使用您的 **Github** 用户名登录 **Gerrit**。
2. 前往 [Setting](#)。在 **Contact Information** 页面上更新您的姓名和电子邮件地址，并上传 **SSH** 公钥。如果您不更新您的姓名，它将在 **Gerrit** 评论中显示为 “**Anonymous Coward**”。
3. 如果还没有这样做，请 **clone the main Kudu repository**。默认情况下，**main remote** 称为 **origin**。当你 **fetch** 或 **pull**，你会从 **origin** 这样做。

```
git clone https://github.com/apache/kudu
```

4. 切换到新的 **kudu** 目录。
5. 添加 **gerrit remote** 。在以下命令中，用您的 **Github** 用户名替换 **<username>** 。

```
git remote add gerrit ssh://<username>@gerrit.cloudera.org:29418
```

6. 运行以下命令来安装 **Gerrit commit-msg hook** 。使用以下命令，用您的 **Github** 用户名替换 **<username>** 。

```
gitdir=$(git rev-parse --git-dir); scp -p -P 29418 <username>@gerrit.cloudera.org:~/.gerrit/commit-msg
```

7. 默认情况下，您已经设置了 **Kudu** 存储库使用 **pull -rebase** 。您可以使用以下两个命令，假设您至今已经检查过主机：

```
git config branch.autosetuprebase always
git config branch.master.rebase true
```

如果由于某种原因，您已经 **checked out branches** 而不是 **master** ，请在上面的第二个命令中替换 **master** 以获取 **other branch names**。

## Submitting Patches ( 提交补丁 )

要提交修补程序，首先提交您的更改（如果可能，使用描述性多行提交消息），然后将请求 **push** 到 **gerrit remote**。例如，要将更改推送到 **master** 分支：

```
git push gerrit HEAD:refs/for/master --no-thin
```

或者将更改 **push** 到 **gh-pages** 分支 ( 更新网页 )：

```
git push gerrit HEAD:refs/for/gh-pages --no-thin
```

## 注意

在准备一个修补程序进行审查时，最好遵循通用 **git** 提交准则和良好做法。

## 注意

**--no-thin** 参数是防止 **Gerrit** 中的错误的解决方法。请参阅 <https://code.google.com/p/gerrit/issues/detail?id=1582>。

## 注意

考虑为上述命令创建 **Git** 别名。 **Gerrit** 还包括一个名为 **git-review** 的命令行工具，您可能会发现有用。

**Gerrit** 会在您的提交消息中添加更改 **ID**，并创建一个 **Gerrit review**，其 **URL** 将作为推送回复的一部分发布。如果需要，您可以向 **kudu-dev** 邮件列表发送消息，解释补丁并请求 **review**。

获得反馈后，您可以更改或修改您的提交（例如，使用像 **git commit -amend** 这样的命令），同时保留更改 **ID**。将您的更改再次 **push** 给 **Gerrit**，这将在 **Gerrit** 中创建一个新的修补程序，并通知所有审阅者有关更改。

当您的代码经过审查并准备合并到 **Kudu** 代码库后，**Kudu** 提交者将使用 **Gerrit** 进行合并。你可以丢弃你的 **local branch**。

## Abandoning a Review ( 放弃 review )

如果您的补丁不被接受或您决定从考虑中提取补丁，则可以使用 **Gerrit UI** 放弃补丁。它仍将在 **Gerrit** 的历史上展示，但不会被列为待审查。

## Reviewing Patches In Gerrit ( 审查 Gerrit 中的补丁 )

您可以使用 **Web UI** 查看 **Gerrit** 中的统一或并行差异更改。要发表评论，请单击相关行号或突出显示该行的相关部分，然后键入 “**c**” 以显示注释框。要提交您的 **review** 和/或 您的 **review status** ，请转到评论的顶层，然后单击 **Reply**。您可以在此处添加其他顶级注释，然后提交。

要查看 **Gerrit review** 中的代码，请单击下载并将相关的 **Git** 命令粘贴到 **Git** 客户端。然后，您可以更新提交并推送 **Gerrit** 向审阅提交补丁，即使您不是原始审阅者。

**Gerrit** 允许您对 **review** 进行投票。在修补程序可以合并之前，需要至少提交一个提交者（除了提交者）之外的一个 **+2** 的投票。

## Code Style ( 代码风格 )

熟悉这些准则，以便您的贡献能够快速轻松地进行审查和整合。

一般来说，**Kudu** 遵循 **Google C++ Style Guide** ，但有以下例外：

### Notes on C++ 11 ( 关于 C++ 11 的注释 )

**Kudu** 使用 **C ++ 11** 。查看 **C ++ 11** 移动语义和 **rvalue** 引用的方便指南：<https://www.chromium.org/rvalue-references> 。

我们的目标是遵循大多数相同的指导原则，例如在可能的情况下迁移远离 **foo.Pass()** ，有利于 **std :: move(foo)**。

### Limitations on boost Use ( boost 使用限制 )

在 **kudu** 代码库中不存在合适的替换的情况下，可以使用仅来自标头库的 **boost** 类。然而：

- 不要对标准 **C ++** 库或 **src/kudu/gutil/** 中存在等效功能的 **boost** 类引入依赖关系。例如，喜欢来自 **gutil** 的 **strings :: Split()**，而不是 **boost :: split**。
- 喜欢使用 **boost** 的功能而不是重新实现相同的功能，除非使用 **boost** 功能需要过度使用我们的风格指南不允许的 **C ++** 功能。例如，**boost :: spirit** 非常基于模板元编程，不应该使用。
- 不要在 **Kudu C ++** 客户端的任何公用头文件中使用 **boost**，因为 **boost** 通常会破坏向后兼容性，并且在两个升级版本之间传递数据（一个由用户由 **Kudu** 导出）会导致严重的问题。

如果有任何提升功能引入新的依赖关系，最好发送电子邮件至 [dev@kudu.apache.org](mailto:dev@kudu.apache.org) 开始讨论。

## Line length ( 线长 )

**Kudu** 团队允许每行 **100** 个字符的行长度，而不是 **Google** 的 **80** 标准。尽可能保持在 **80** 以下，但如果需要，您可以溢出到 **100** 个左右。

## Pointers ( 指针 )

### Smart Pointers and Singly-Owned Pointers ( 智能指针和单独指针 )

通常，大多数对象应该有明确的 “**single-owner**” 语义。大多数时候，**singly-owned** 的对象可以包装在 **unique\_ptr <>** 中，确保在范围退出时删除，并防止意外复制。

如果对象是 **singly owned** 的，但是从多个位置引用，例如当已知指向对象至少与指针本身一样长时，将注释与将原始指针存储并存储的构造函数相关联，如在下面的例子中。

```
// 'blah' must remain valid for the lifetime of this class
```

```
MyClass(const Blah* blah) :  
    blah_(blah) {  
}
```

注意

**Kudu** 代码库的较旧部分使用 **gscoped\_ptr** 而不是 **unique\_ptr** 。这些都是在 **Kudu** 采用 **C ++ 11** 之前进行的。新代码不应该使用 **gscoped\_ptr** ，除非需要与现有代码进行接口。或者，考虑在您遇到这些问题时更新用法。

注意

严格禁止使用 **std :: auto\_ptr** ，因为它的难度大且易出错的语义。此外， **std :: auto\_ptr** 自 **C ++ 11** 被声明为不推荐使用。

**Smart Pointers for Multiply-Owned Pointers ( 多指针指针的智能指针 ):**

虽然 **single ownership** 是理想的，但有时候是不可能的，特别是当多个线程正在运行时，指针的生命周期没有明确定义。在这些情况下，您可以使用 **std :: shared\_ptr** 或 **Kudu** 自己的 **scoped\_refptr** 从 **gutil/ref\_counted.hpp** 。这些机制中的每一个依赖于引用计数，以便在没有更多指针保留时自动删除指示。这两种类型的指针之间的关键区别是 **scoped\_refptr** 要求对象扩展一个 **RefCounted** 基类，并将其引用计数存储在对象存储本身内，而 **shared\_ptr** 在堆上维护单独的引用计数。

利弊是：

**shared\_ptr**



- 可以与任何类型的对象一起使用，而不需要从特殊的基类派生对象
- 标准库的一部分，大多数 **C++** 开发人员熟悉
- 支持 **weak\_ptr** 的用例：
  - 当对象仅在存在的情况下需要被访问时才是临时所有权
  - 打破 **shared\_ptr** 的循环引用，如果由于聚合存在任何存在
- 您可以将 **shared\_ptr** 转换为 **weak\_ptr** 并返回
- 如果使用 **std::make\_shared <>()** 创建一个实例，则只能进行一次分配（因为 **C++ 11; Standard** 中的非绑定的要求）
- 如果使用 **shared\_ptr <T> p(new T)** 创建新对象需要两个分配（一个用于创建引用计数，另一个用于创建对象）
- 引用计数可能不在堆上的对象附近，因此在访问时可能会发生额外的高速缓存未命中
- **shared\_ptr** 实例本身需要 **16** 个字节（指向 **ref** 计数的指针和指向对象的指针）

## **scoped\_refptr**

- 只需要一个分配，并且 **ref** 计数与对象在同一个高速缓存行上
- 指针只需要 **8** 个字节（因为引用计数在对象内）
- 当需要更多控制时，您可以手动增加或减少参考计数
- 您可以将原始指针转换回 **scoped\_refptr**，而不必担心双重释放
- 由于我们控制实现，我们可以实现功能，例如调试构建，捕获每个对象的堆栈跟踪以帮助调试泄漏。
- 引用对象必须从 **RefCounted** 继承
- 不支持 **weak\_ptr** 的用例

由于 **scoped\_refptr** 通常越来越小，所以尝试在新代码中使用而不是

**shared\_ptr** 。现有代码在许多地方使用 **shared\_ptr** 。当与该代码连接时，可以继续使用 **shared\_ptr** 。

## Function Binding and Callbacks ( 函数绑定和回调 )

现有代码使用 **boost :: bind** 和 **boost :: function** 来进行函数绑定和回调。对于新代码，请使用 **gutil** 中的回调和绑定类。虽然功能较少（绑定不支持参数占位符，包装函数指针或函数对象），但它们通过参数生命周期管理的方式提供更多选项。例如，当 **Callback** 超出范围时，绑定参数的类扩展 **RefCounted** 将在绑定期间递增，并减少。

有关详细信息，请参阅 **gutil/callback.h** 中的大文件注释，**util/callback\_bind-test.cc** 作为示例。

## CMake Style Guide ( CMake 样式指南 )

**CMake** 允许以较低，上限或混合大小写的命令。要保持 **CMake** 文件一致，请使用以下准则：

- 小写的 **built-in commands**

```
add_subdirectory(some/path)
```

- 大写的 **built-in arguments**

```
message(STATUS "message goes here")
```

- 大写的 **custom commands or macros**

```
ADD_KUDU_TEST(some-test)
```

---

## GFlags

**Kudu** 使用 **gflags** 进行命令行和基于文件的配置。使用这些准则添加新的 **gflag** 。所有新 **gflags** 必须符合这些准则。现有的不符合要求的产品将及时符合规定。

### Name ( 名称 )

**gflag** 的名字传达了很多信息，所以选择一个好名字。该名称将传播到其他系统，如 [配置参考](#)。

- 多字名称的不同部分应以下划线分隔。例如， **fs\_data\_dirs** 。
- 该名称应以其影响的上下文为前缀。例如，  
**webserver\_num\_worker\_threads** 和 **cfile\_default\_block\_size** 。上下文可能很难定义，所以请记住，这个前缀将用于将类似的 **gflags** 组合在一起。如果 **gflag** 影响整个过程，那么它不应该是前缀。
- 如果 **gflag** 是一个数量，该名称应该后缀单位。例如，  
**tablet\_copy\_idle\_timeout\_ms** 。
- 如有可能，请使用短名称。这将为手动输入命令行选项的人节省时间。
- 这个名称是 **Kudu** 兼容性合同的一部分，不应该没有很好的理由来改变。

### Default value ( 默认值 )

选择默认值通常很简单，但像名称一样，它传播到其他系统中。

- 默认值是 **Kudu** 的 **compatibility contract** ( 兼容性合同 ) 的一部分，如果没有很好的理由，不应该改变。

## Description ( 描述 )

**gflag** 的描述应该补充名称并提供其他上下文和信息。与名称一样，说明传播到其他系统。

- 描述可以包括多个句子。每个都应该以一个大写字母开头，以一个句点结尾，并在之前的一个空格开始。
- 描述不应包含 **gflag** 的类型或默认值;它们是带外提供的。
- 描述应该在第三人称。不要使用像你这样的话。
- **gflag** 描述可以自由更改; **Kudu** 的发行预计不会保持不变。

## Tags ( 标志 )

**Kudu** 的 **gflag** 标记机制为每个 **gflag** 添加了机器可读上下文，用于消耗系统，如文档或管理工具。请参阅 **flag\_tags.h** 中的大块注释，以获取准则。

## Miscellaneous ( 杂 )

- 避免为同一个逻辑参数创建多个 **gflags** 。例如，许多 **Kudu** 二进制文件需要配置一个 **WAL** 目录。而不是创建 **foo\_wal\_dir** 和 **bar\_wal\_dir gflags** ，最好使用一个单一的 **kudu\_wal\_dir gflag** 来普遍使用。

## Testing ( 测试 )

**All new code should have tests. ( 所有新的代码都应该有测试。 )**

在现有文件中添加新的测试，或根据需要创建新的测试文件。

**All bug fixes should have tests. ( 所有错误修复都应该有测试。 )**

如果由现有测试用例触发，则可以修复错误而不添加新测试。例如，如果在 **20** 分钟左右之后运行多线程系统测试时出现了一个 **race**，那么值得尝试更有针对性的测试用例来触发该错误。但是如果这很难做，现有的系统测试就够了。

**Tests should run quickly (< 1s).** ( 测试应该很快运行 (<1s)。 )

如果要编写时间密集的测试，请使运行时依赖于通过 **KUDU\_ALLOW\_SLOW\_TESTS** 环境变量启用的 **KuduTest#AllowSlowTests**，并由 **Jenkins** 测试执行使用。

**Tests which run a number of iterations of some task should use a `gflags` command-line argument for the number of iterations.** ( 运行一些任务的迭代的测试应该使用 **gflags** 命令行参数来执行迭代次数。 )

这对于编写快速压力测试或性能测试非常方便。

**Commits which may affect performance should include before/after `perf-stat(1)` output.** ( 可能影响性能的提交应包括在 **perf-stat(1)** 输出 之前/之后。 )

这将显示性能提升或 **non-regression** ( 不回归 )。 **Performance-sensitive** ( 性能敏感 ) 代码应包括一些可用作目标基准测试用例。

# Export Control Notice ( 出口管制通知 )

原文链接：[http://kudu.apache.org/docs/export\\_control.html](http://kudu.apache.org/docs/export_control.html)

译文链接：<http://cwiki.apachecn.org/pages/viewpage.action?pagelId=10813657>

贡献者：小瑶 ApacheCN Apache中文网

## Export Control Notice ( 出口管制通知 )

该分发包括加密软件。您目前居住的国家可能对进口，拥有，使用，和/或再出口加密软件的其他国家有限制。在使用任何加密软件之前，请检查您所在国家有关进口，拥有或使用和再出口加密软件的法律，法规和政策，以查看是否允许。有关更多信息，请参阅 <http://www.wassenaar.org/>。

美国商务部，工业和安全局（**BIS**）已将该软件分类为出口商品管理编号（**ECCN**）**5D002.C.1**，其中包括使用或执行不对称算法的加密功能的信息安全软件。该 **Apache Software Foundation** 发行版的形式和方式使其符合许可证异常ENC技术软件无限制（**TSU**）出口（参见 **BIS** 出口管理规定，第 **740.13** 节），用于对象代码和源代码。

以下提供了有关加密软件的更多详细信息：

- 该软件使用 **OpenSSL** 启用 **TLS** 加密的连接，生成非对称加密的密钥，并使用这些密钥生成和验证签名。
- 该软件使用 **Java SE** 安全库，包括 **Java** 安全套接字扩展（**JSSE**），**Java** 通用安全服务（**JGSS**）和 **Java** 认证和授权 **API**（**JAAS**），以提供安全认证和 **TLS** 保护的传输。