时序数据库 CTSDB

CTSDB1.0版与2.0版

腾讯云

文档目录

CTSDB1.0版与2.0版
产品简介
产品概述

最近更新时间：2024-09-02 11:17:01

简介

时序数据库 CTSDB（TencentDB for CTSDB）是腾讯云推出的一款分布式、可扩展、支持近实时数据搜索与分析的时序数据库。该数据库为非关系型数据库，提供高效读写、低成本存储、强大的聚合分析能力、实例监控以及数据查询结果可视化等功能。整个系统采用多节点多副本的部署方式，有效保证了服务的高可用性和数据的高可靠性。

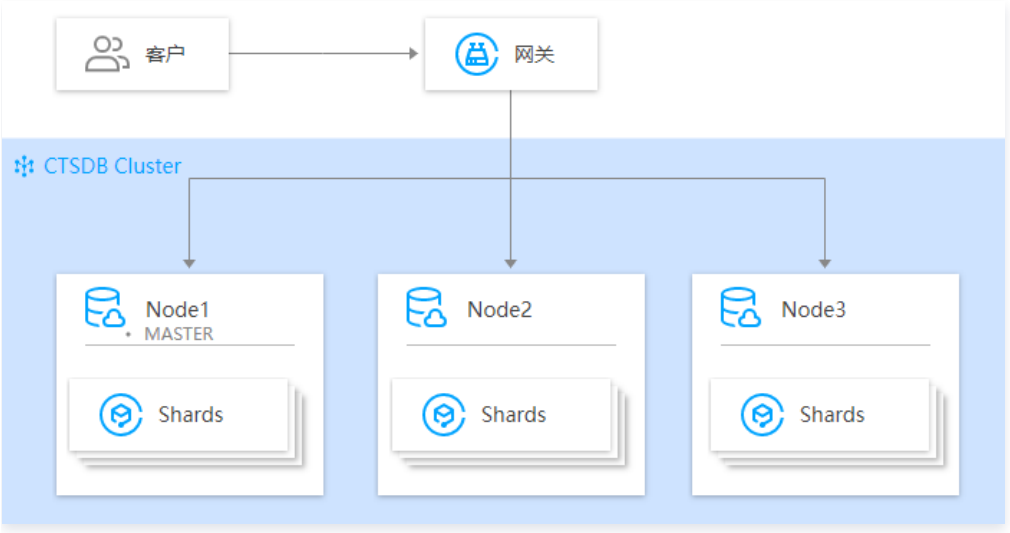
产品优势

CTSDB 在处理海量时序数据时优势如下：

- 高并发写入：数据先写入内存，再周期性的 Dump 为不可变的文件存储。且可以通过批量写入数据，降低网络开销。
- 低成本存储：通过数据上卷（Rollup），对历史数据做聚合，节省存储空间。同时利用合理的编码压缩算法，提高数据压缩比。
- 强大的聚合分析能力：支持丰富的聚合查询方式，不仅支持 avg、min、max 等常用的聚合方式，还支持 Group By、区间、Geo、嵌套等复杂聚合分析。

产品架构

通用集群架构

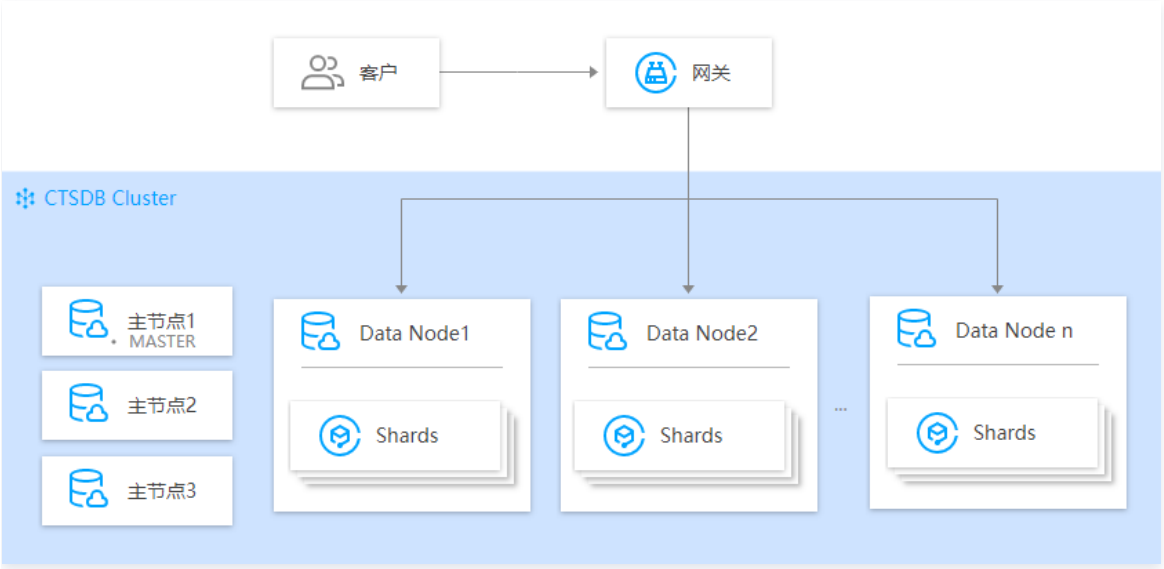


通用集群是由多个节点共同组成的分布式集群。

- 每个节点都对外接收请求，节点之间互通，彼此配合，提供数据存储和索引等服务（节点之间能够将客户端请求转向到合适的节点），均具有被选为 MASTER 节点的资格。
- CTSDB 通用集群的节点数量小于30个时，无需添加专有主节点，通用集群架构即可满足使用要求。

> **说明**
> CTSDB 集群可以通过添加专有主节点，将集群架构从通用集群架构优化升级为混合节点集群架构。

混合节点集群架构



混合节点集群是由一类具有被选为 MASTER 节点资格的专有主节点和数据节点组成的分布式集群。

- 专有主节点负责维护保障整个集群的健康状态和稳定性，不负责数据存储等服务；数据节点提供数据存储和索引等服务。
- 随着用户业务发展和数据量增长，节点数量超过30个时，建议添加专有主节点，将通用集群架构优化升级为混合节点集群架构，充分保证多节点超大集群的性能发挥。

> **说明**
> CTSDB 集群无法从混合节点集群架构变更为通用集群架构。

产品优势

最近更新时间：2024-08-30 16:37:35

高性能

支持批量写入、高并发查询，通过集群扩展，可线性提升系统性能。

易使用

丰富的数据类型，兼容 Elasticsearch 常用的 API 接口。控制台提供丰富的数据管理和运维功能，操作简单。

高可靠

支持多副本，分布式部署，数据自动均衡。

低成本

通过上卷表 Rollup 提高压缩比，降低存储成本。

强大的聚合分析能力

支持 max、min、avg、percentile、sum、count 等常用聚合。复杂的脚本聚合、时间区间聚合、GEO 聚合和嵌套聚合等。

地域和可用区

最近更新时间：2024-08-30 10:43:52

腾讯云数据库托管机房分布在全球多个位置，这些位置节点称为地域（Region），每个地域又由多个可用区（Zone）构成。
每个地域（Region）都是一个独立的地理区域。每个地域内都有多个相互隔离的位置，称为可用区（Zone）。每个可用区都是独立的，但同一地域下的可用区通过低时延的内网链路相连。腾讯云支持用户在不同位置分配云资源，建议用户在设计系统时考虑将资源放置在不同可用区以屏蔽单点故障导致的服务不可用状态。
地域、可用区名称是对机房覆盖范围最直接的体现，为便于客户理解，命名规则如下：

- 地域命名采取【覆盖范围＋机房所在城市】的结构，前半段表示该机房的覆盖能力，后半段表示该机房所在或临近的城市。
- 可用区命名采取【城市 + 编号】的结构。

地域

腾讯云不同地域之间隔离，保证不同地域间最大程度的稳定性和容错性。建议您选择最靠近您用户的地域，可降低访问时延、提高下载速度。用户启动实例、查看实例等操作都是区分地域属性的。
云产品内网通信的注意事项如下：

- 同地域下（保障同一账号，且同一个 VPC 内）的云资源之间可通过内网互通，可以直接使用 内网 IP 访问。
- 不同地域之间网络隔离，不同地域之间的云产品默认不能通过内网互通。
- 处于不同私有网络的云产品，可以通过 云联网 进行通信，此通信方式更较为高速、稳定。

可用区

可用区（Zone）是指腾讯云在同一地域内电力和网络互相独立的物理数据中心。目标是能够保证可用区间故障相互隔离（大型灾害或者大型电力故障除外），不出现故障扩散，使得用户的业务持续在线服务。通过启动独立可用区内的实例，用户可以保护应用程序不受单一位置故障的影响。

支持的地域和可用区

> **说明：**
> 不同地域可用区所开放的资源可能因资源售罄而缺少，之前已售罄的资源可能又得到了重新补给。资源的开放情况会根据实际业务使用情况会随时评估调整，请以控制台购买页所开放的资源为准。

中国

地域（region）	可用区（zone）
华南地区（广州） ap-guangzhou	广州四区 ap-guangzhou-4
华东地区（上海） ap-shanghai	上海二区 ap-shanghai-2
	上海三区 ap-shanghai-3
华北地区（北京） ap-beijing	北京一区 ap-beijing-1
西南地区（重庆） ap-chongqing	重庆一区 ap-chongqing-1
港澳台地区（中国香港） ap-hongkong	香港三区（中国香港节点可用于覆盖港澳台地区） ap-hongkong-3

其他国家和地区

地域（region）	可用区（zone）
亚太东南（新加坡） ap-singapore	新加坡二区（适合用于覆盖亚太东南地区） ap-singapore-2

亚太东南（曼谷） ap-bangkok	曼谷二区（曼谷节点可用于覆盖亚太东南地区） ap-bangkok-2
亚太南部（孟买） ap-mumbai	孟买二区（适合用于覆盖亚太南部地区） ap-mumbai-2
亚太东北（首尔） ap-seoul	首尔二区（首尔节点可用于覆盖亚太东北地区） ap-seoul-2
美国（硅谷） na-siliconvalley	硅谷二区（适合用于覆盖美国地区） na-siliconvalley-2
欧洲地区（法兰克福） eu-frankfurt	法兰克福二区（适合用于覆盖欧洲地区） eu-frankfurt-2

如何选择地域和可用区

购买云服务时建议选择最靠近您客户的地域，可降低访问时延、提高下载速度。

购买指南
计费概述

最近更新时间：2022-08-22 16:31:12

时序数据库 CTSDB 目前支持包年包月和按量计费两种模式（按量计费购买前需要实名认证，详见 实名认证指引 ）。包年包月是一种预付费计费模式，详情请参见 预付费计费说明 。按量计费是一种后付费计费模式，详情请参见 按量计费说明 。

按量计费阶梯价格

时序数据库 CTSDB 支持按量计费阶梯价，用得久更便宜。
按量计费根据使用时长不同，共分为三个阶梯：

- 0天 – 4天（4 * 24小时内），适用按量计费第一阶梯价格。
- 4天 – 15天（4 * 24小时起，至15 * 24小时内），适用按量计费第二阶梯价格。
- 15天以上（15 * 24小时起），适用按量计费第三阶梯价格。

具体的价格请参见 价格说明 。

实例续费管理

在续费管理页面提供实例的**批量续费**、**设为自动续费**、**统一到期日**等功能，详见 续费管理 。

> **注意**
> 为保障您业务正常进行，当硬盘空间即将满时，请及时升级数据库实例规格或者购买硬盘空间。详情请参见 调整数据库实例规格 。
> 实例存储数据量超过实例购买容量时会被锁住，仅能读取数据不能写入，需扩容或删除部分数据库表解除只读。

产品定价

最近更新时间：2024-01-18 10:32:32

售卖地域

腾讯云时序数据库 CTSDB 支持的售卖地域覆盖全球多个区域，具体请参见 地域和可用区 。CTSDB 支持内存和硬盘单独计价的售卖方式，为用户灵活搭配提供更多途径。

价格计算

实例价格计算公式

实例价格
= 内存规格费用 + 存储空间费用
= (总内存规格 * 单位内存价格) + (总存储空间 * 单位存储价格)
= (单节点内存数 * 节点数 * 单位内存价格) + (单节点存储空间 * 节点数 * 单位存储价格)

单节点内存价格

针对按量计费新增阶梯价格，用得久更加优惠。详情请参照价目表：

> **注意：**
> 在生产环境中，选择节点数量与节点规格，建议如下：
> - 节点数量：配置至少3个节点。仅有2个节点的实例，无法完全保证数据的高可靠性，仅适用于测试环境。
> - 节点规格：配置至少2核9GB。节点规格为1核2GB，内存过小无法保证高可用性，仅适用于测试环境。

CPU (核)	内存 (GB)	适用场景	推荐写入次数 (点/秒)	单位内存价格 包月优惠价 (元/GB/月)	单位内存价格 按量计费价第一阶梯 (0,96小时] (元/GB/小时)	单位内存价格 按量计费价第二阶梯 (96,360小时] (元/GB/小时)	单位内存价格 按量计费价第三阶梯(360小时以上) (元/GB/小时)
1	2	测试环境	12500	67	0.186	0.1396	0.09305
1	4	生产环境	25000	67	0.186	0.1396	0.09305
2	9	生产环境	50000	67	0.186	0.1396	0.09305
4	20	生产环境	100000	60.30	0.1675	0.1256	0.08375
8	40	生产环境	200000	56.95	0.1582	0.1186	0.0791
16	80	生产环境	400000	53.60	0.1489	0.1117	0.0744
28	128	生产环境	800000	50.25	0.13958	0.1047	0.06979

存储空间价格

数据库类型	单位存储价格 包月优惠价(元/GB/月)	单位存储价格 按量计费价格(元/GB/小时)
CTSDB 实例	0.36	0.0005

购买方式

最近更新时间：2023-08-15 14:49:02

本文为您介绍如何通过控制台购买时序数据库 CTSDB 实例。

前提条件

已 注册腾讯云账号，并 实名认证 成功。

操作步骤

1. 登录 CTSDB 购买页，选择数据库各项配置，确认无误后，单击**立即购买**。
- **计费模式**：支持包年包月和按量计费。
  - 若业务量有较稳定的长期需求，建议选择包年包月。
  - 若业务量有瞬间大幅波动场景，建议选择按量计费。
- **地域和可用区**：选择您业务需要部署的地域，请参见 地域和可用区 。
- **版本**：支持 CTSDB 1.0、CTSDB 2.0。
  - CTSDB 1.0 版本，兼容 Elasticsearch 6.8.2。
  - CTSDB 2.0 版本（推荐使用 ），兼容 Elasticsearch 7.10.1，支持类 SQL 查询语句，数据压缩算法、写入性能等多方面优化。
- **配置模式**：支持快速配置和自定义配置。
- **存储容量**：快速配置模式中，存储容量即单副本存储容量 = 集群硬盘总容量 / 副本数。
- **网络类型**：云数据库所属网络，建议您选择与云服务器同一个地域下的同一私有网络，否则无法通过内网连接云服务器和数据库。私有网络说明请参见 网络环境 。
- **端口**：自定义端口号需在1024到65535之间。
- **指定项目**：数据库实例所属的项目，缺省设置为默认项目。
- **标签**：给实例设定标签。您可以根据标签归类管理实例。单击**添加**，可以选择标签键与标签值。
- **实例名**：可选择创建后命名或立即命名。
- **密码**：root 账号的密码，8 - 64个字符，需包含英文字母、数字和 `~!@#$%^&*()_+-=|{}[];:<>,.?/` 字符中的三种。
- **费用**：详情请参见 价格说明 。
2. 购买完成后，返回实例列表，待实例状态变为**运行中**，即可进行后续操作。

相关文档

连接实例

续费说明

最近更新时间：2022-03-24 15:34:38

时序数据库 CTSDB 支持控制台续费和续费管理中心续费两种方式。

控制台续费

批量续费

1. 登录 CTSDB 控制台，在实例列表，选择一个或多个需要续费的实例，单击上方的**续费**。

2. 在弹出的对话框，选择续费时长，单击**确认**。
3. 支付成功后，返回实例列表，待实例状态变为**运行中**，即可正常使用。

自动续费

在实例列表，选择一个或多个需要续费的实例，单击上方的**自动续费**。

续费管理中心续费

在 续费管理页面 提供实例的**批量续费**、**设为自动续费**、**统一到期日**等功能，详见 续费管理 。

欠费说明

最近更新时间：2022-08-22 15:46:02

包年包月云数据库到期提醒

到期预警

包年包月的资源会在到期前七天开始，隔天向您推送到期预警，预警消息将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

欠费预警

包年包月的资源到期当天及每隔天向您推送欠费隔离预警，预警消息将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

回收机制

- 云服务资源到期前七天，系统会开始给您发送续费提醒通知。
- 到期后七天内云服务还可以继续使用，实例状态为**已过期**，需要尽快续费，系统将发送云服务到期提醒。
- 到期后第八天开始，此云数据库不可再使用，实例状态为**已隔离**。您可在控制台实例列表页面进行续费操作。
- 云数据库在到期十四天后若仍未进行续费操作，则资源将被系统回收，数据将被清除且不可恢复。
- 即到期后，云数据库仍有**七天可用时间**和**七天不可用时间**，您可在这十四天内对设备进行续费。余额充足的情况下，设置了自动续费的设备，自动续费也会照常执行。

按量计费云数据库到期提醒

余额预警

系统每天会根据您名下按量付费资源过去24小时的消费情况以及账户余额情况，预估余额可支撑的时间。若可支撑时间小于5天，我们将会向您推送余额预警。预警消息将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

欠费预警

按量计费资源每个整点进行扣费。在您的账户被扣为负值时，我们将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

欠费处理

从余额扣为负值时刻起：

- 24小时内，云数据库可继续使用且继续扣费。
- 24小时后，云数据库实例将**自动关机且停止扣费**。

自动关机之后：

- 关机后的3天内，若您的账户余额未充值到大于0，不可对其开机；若充值到余额大于0，计费将继续，可对其开机。
- 账户余额小于0达到3天，按量计费云数据库将回收，**所有数据将会被清理，且不可找回**。

云数据库回收时，我们将通过邮件及短信的方式通知到腾讯云账户的创建者以及所有协作者。

> ⚠ **注意**
> 按量计费资源不再使用时**请及时销毁**，以免继续扣费。
> 您的实际资源消耗可能不断变化，因此余额预警可能存在一定的误差。

退费说明

最近更新时间：2022-04-01 11:17:22

为了更加方便您使用云数据库，如果您在购买包年包月云数据库后有任何不满意，我们支持以下退货退款。

- 每个主体下，您可享受**1台**云数据库实例5天无理由退还，您支付的有效金额将返还到您的腾讯云账户。
- 除此之外，您还可享受普通退还，扣除您已使用的费用，退款金额将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

以上均需要通过 在线支持 申请。

退还说明

- 包年包月实例退还后，实例的状态一旦变为隔离中或已隔离时，就不再产生与该实例相关的费用。
- 包年包月实例彻底销毁后 IP 资源同时释放，实例无法访问。
- 包年包月实例退还后，实例状态标记为已销毁，此时实例无法访问。如您想恢复已经退还的包年包月实例，可以进行续费恢复。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。
- 某些活动资源不支持自助退还，具体以官网展示为准。

五天无理由自助退还

时序数据库 CTSDB 产品遵守腾讯云 云服务退货说明，如果您在购买时序数据库 CTSDB 后有任何不满意，我们支持五天内无理由自助退还，具体规则如下：

- 每个主体下，包年包月预付费时序数据库 CTSDB 自**新购**之日起五天之内（含五天），默认享受**1台**云数据库实例五天无理由退还。
- 计费模式由**按量计费**切换至**包年包月**的云数据库实例，不支持五天内无理由退还。
- 如出现疑似异常/恶意退货，腾讯云有权拒绝您的退货申请。

五天无理由自助退还规则

符合五天无理由退还场景的订单，退款金额为**购买时花费的全部消耗金额**，包括现金账户金额、收益转入账户金额以及赠送账户金额。具体退款规则请参见 五天内无理由全额退款 。

> ⚠ 注意
> - **抵扣的代金券不予以退还。**
> - **退还金额**将全部退还到**腾讯云账号余额**。

普通退还

如果您已经享用5天无理由退还，我们还支持您的**199台**包年包月云数据库实例可在任意时间内提交工单退还。
普通退还将扣除您已使用的费用，退款金额将按购买支付使用的现金和赠送金支付比例退还至您的腾讯云账户。

普通退还规则

退款金额 = 当前有效订单金额 + 未开始订单金额 − 资源已使用价值

- 当前有效订单金额：指生效中订单的付款金额，不包含折扣和代金券。
- 未开始订单金额：将来生效订单的付款金额，不包含代金券。
- 资源已使用价值按照如下策略计算：
- 已使用部分，发起退费当天已满整月按整月扣除，不满整月则按量计费扣除。
- 已使用部分精确到秒。
- 退款金额 ⩽ 0，按0计算并清退资源。

> ⚠ 注意
> - 抵扣或代金券不予以退还。
> - 退还金额将按购买使用的现金和赠送金支付比例返还到您的腾讯云账户。

快速入门
创建实例

最近更新时间：2024-05-14 11:15:31

本文为您介绍如何通过控制台购买时序数据库 CTSDB 实例。

前提条件

已 注册腾讯云账号，并 实名认证 成功。

操作步骤

1. 登录 CTSDB 购买页，选择数据库各项配置，确认无误后，单击**立即购买**。
   - **计费模式**：支持包年包月和按量计费。
     - 若业务量有较稳定的长期需求，建议选择包年包月。
     - 若业务量有瞬间大幅波动场景，建议选择按量计费。
   - **地域和可用区**：选择您业务需要部署的地域，请参见 地域和可用区 。
   - **版本**：支持 CTSDB 1.0、CTSDB 2.0。
     - CTSDB 1.0 版本，兼容 Elasticsearch 6.8.2。
     - CTSDB 2.0 版本（推荐使用 ），兼容 Elasticsearch 7.10.1，支持类 SQL 查询语句，数据压缩算法、写入性能等多方面优化。
   - **配置模式**：支持快速配置和自定义配置。
   - **存储容量**：快速配置模式中，存储容量即单副本存储容量 = 集群硬盘总容量 / 副本数。
   - **网络类型**：云数据库所属网络，建议您选择与云服务器同一个地域下的同一私有网络，否则无法通过内网连接云服务器和数据库。私有网络说明请参见 网络环境 。
   - **端口**：自定义端口号需在1024到65535之间。
   - **指定项目**：数据库实例所属的项目，缺省设置为默认项目。
   - **标签**：给实例设定标签。您可以根据标签归类管理实例。单击**添加**，可以选择标签键与标签值。
   - **实例名**：可选择创建后命名或立即命名。
   - **密码**：root 账号的密码，8 - 64个字符，需包含英文字母、数字和 `~!@#$%^&*()_+-=|{}[]:;<>,.?/` 字符中的三种。
   - **费用**：详情请参见 价格说明 。
2. 购买完成后，返回实例列表，待实例状态变为**运行中**，即可进行后续操作。

相关文档

连接实例

连接实例

最近更新时间：2024-05-16 17:00:42

CTSDB 实例目前仅提供 VPC 网络下的连接方式。您可以通过控制台连接实例，也可以通过 RESTful API 接口连接实例，通过 API 接口连接实例时需要提供 root 账号的密码，以确保安全性。
CURL 连接实例创建表的示例如下，其中 `${user:password}` 是实例的用户名和密码，`${vip}:${vport}` 是实例的 IP 和 Port，`${metric_name}` 是新建的表名称。参数介绍可参见 新建 metric 。

说明：

- 通过内网地址连接云数据库，云服务器 和数据库须是同一账号，且同一个 VPC 内（保障同一个地域），或同在基础网络内。
- 内网地址 IP 和 Port 可在 控制台 的实例列表查看。
- 如忘记账号密码，可参考 重置密码 修改账号密码。

```
curl -u ${user:password} -H 'Content-Type:application/json' -X PUT
${vip}:${vport}/_metric/${metric_name} -d'
    {
        "tags": {
            "region": "string",
            "set":  "long",
            "host": "string"
        },
        "time": {
             "name": "timestamp",
             "format": "epoch_second"
        },
        "fields": {
            "cpu_usage":  "float"
        },
        "options": {
            "expire_day": 7,
            "refresh_interval": "10s",
            "number_of_shards": 5,
            "number_of_replicas": 1,
            "rolling_period": 1
        }
    }'
```

操作指南
系统限制

最近更新时间：2024-08-30 16:32:39

CTSDB 支持 HTTP 协议进行数据写入和查询等操作。提供的 HTTP API 是 RESTful API，对资源的请求方式是通过向资源对应的 URI 发送标准的 HTTP 请求，例如 GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源等。用户通过 HTTP API 几乎可以实现所有的数据操作。CTSDB 通过提供 VPC 网络隔离和访问时提供用户名和密码的身份认证方式来保证数据的安全性，通过 JSON 格式的结构体进行数据交换，每个请求都会返回一个标准的 HTTP 响应状态码和响应内容。若操作失败，用户可以根据响应内容获取到具体错误信息。

命名规则

CTSDB 中 metric、tag、field 的命名应简明扼要。

- metric 命名规则：允许使用小写英文字母、数字、_（下划线）、-（短划线）、.（圆点）的任意组合，且不能以 .（圆点）、_（下划线）或 -（短划线）开头。长度为1到200个字符。
- metric 中 tags 和 fields 命名规则：允许大小写英文字母、数字、_（下划线）、-（短划线）、.（圆点）的任意组合，且不能以 .（圆点）开头。长度为1到255个字符。

系统限制

- metric 中 tags 和 fields 限制：metric中字段总数不超过 1000。
- 批量写入 metric 时写入点限制：建议单次 bulk 条数在 1000~5000 之间，物理大小在 1-15MB 大小之间。

系统默认规则

- **写入数据时新增字段处理**：如果您往 metric 表写入数据时有未定义的新增字段，CTSDB 默认会将新增字段存储为 tags。您也可通过修改 metric 的 options 选项的 default_type 字段来修改。修改 metric 请参考 更新 metric 。
- 若新增字段为整型，则 CTSDB 将其存储为 long 类型；
- 若新增字段为小数，则 CTSDB 将其存储为 float 类型；
- 若新增字段为字符串，则 CTSDB 将其存储为 string 类型，其长度为 max_string_length，超过部分会被系统丢弃，max_string_length 的长度可自定义，默认为256个字符，修改请参考 更新metric 。
- **日期与时间处理**：日期与时间在 CTSDB 中以 UTC 格式存储，因此，查询数据时涉及时间范围的查询请通过 time_zone 参数来指定时区，其格式为 ISO 8601 UTC 偏移（例如 +01:00 或者 -08:00），具体时区需根据实例所处的地域来确定，一般国内地域是东八区。具体请参考 常用查询示例 。

基本操作

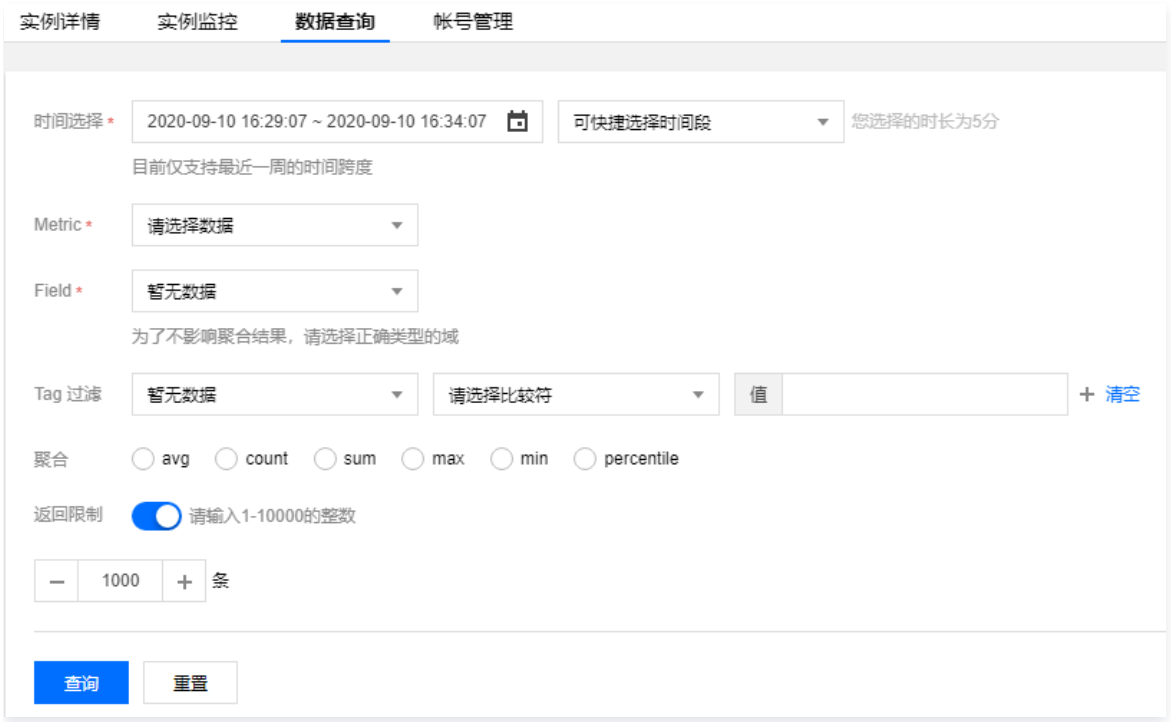
最近更新时间：2024-08-30 16:36:52

查看实例详情

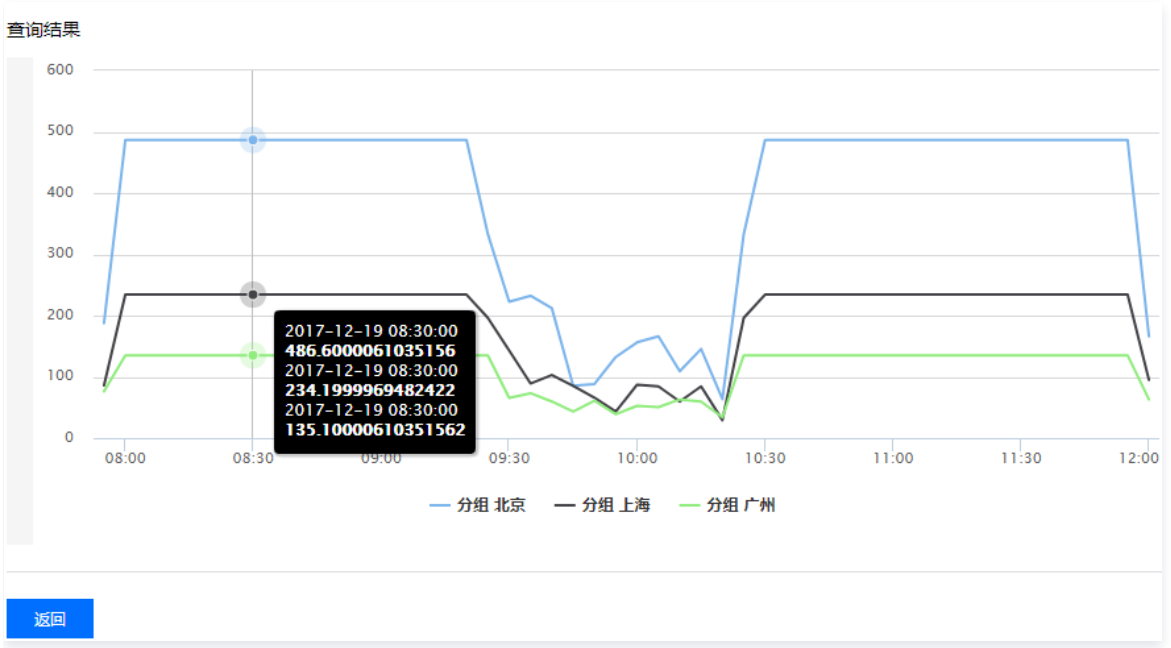
登录 CTSDB 控制台，在实例列表，单击实例 ID 或**操作**列的**管理**，进入实例详情页，即可查看实例详情信息。

查询数据

选择**数据查询**页，选择相应的查询条件。



单击**查询**，可生成图表。



查看监控

选择**实例监控**页，即可查看实例的各指标监控数据。

为实例指定项目

最近更新时间：2024-08-30 16:32:57

时序数据库 CTSDB 支持将实例分配至不同的项目进行管理。
需要注意的几点特性是：

- 数据库实例在项目间进行分配和移动，不会影响实例对外提供的服务。
- 用户须在新购实例时为实例指定所属的项目，缺省为**默认项目**。
- 已指定项目的实例，可通过 控制台 的实例详情页，更改项目重新指定到其他项目。

调整实例节点

最近更新时间：2024-08-30 16:32:18

时序数据库 CTSDB 支持灵活的实例节点调整操作。用户可根据其业务所处的实际情况（业务初期、业务快速发展期、业务高峰期、业务低谷期等）灵活的调整其实例的节点数量，从而更好满足用户的资源充分利用和成本实时优化等需求。

调整节点规则

- 时序数据库 CTSDB 实例处于正常状态下（运行中）并且当前没有任务执行时才能够发起调整节点操作。
- 调整节点过程中，不允许取消本次调整节点操作。
- 调整节点前后实例的名称、访问 IP、访问端口均不发生变化。
- 调整节点过程中，CTSDB 实例可正常访问，业务不受影响。
- 调整节点完毕时可能会涉及实例切换，建议程序有自动重连功能。
- 调整节点过程中，请尽量避免修改用户密码等操作。

调整节点方法

1. 登录 CTSDB 控制台，选择对应地域，在实例列表，单击实例 ID 或**操作**列的**管理**，进入实例详情页。
2. 在实例详情页的**节点数**处，单击**增加节点**。

3. 在弹出的对话框，选择需要增加的节点数，单击**升级**。

调整实例规格

最近更新时间：2024-08-30 11:09:42

时序数据库 CTSDB 支持灵活的实例规格调整操作。用户可根据其业务所处的实际情况（业务初期、业务快速发展期、业务高峰期、业务低谷期等）灵活的调整其实例规格，从而更好满足用户的资源充分利用和成本实时优化等需求。

调整配置规则

1. 时序数据库 CTSDB 实例处于正常状态下（运行中）、节点数至少3个节点，并且当前没有任务执行时才能够发起调整配置操作。
2. 调整配置过程中，不允许取消本次调整配置操作。
3. 调整配置前后实例的名称、访问IP、访问端口均不发生变化。
4. 调整配置过程中，可能会涉及到数据的搬迁，期间 CTSDB 实例可正常访问，业务不受影响。
5. 调整配置完毕时可能会涉及实例切换，建议程序有自动重连功能。
6. 调整配置过程中，请尽量避免修改用户密码等操作。

调整配置方法

1. 登录 CTSDB 控制台，选择对应地域，在实例列表，选择**操作**列的**更多** > **调整配置**。

2. 在弹出的对话框，选择升级后的规格，单击**提交**。
3. 返回实例列表，待实例状态变为**运行中**，即可正常使用。

费用计算

1. 对包年包月实例，用户自助升级数据库实例时，系统将计算实例规格间的差价，并从用户的账户中扣除差价，若账户余额不足则需要先进行充值。升级后将按照新的实例规格进行计费。
2. 对按量计费实例，升级后下一个计费周期将按照新的实例规格进行计费。

重置密码

最近更新时间：2024-05-16 17:00:42

操作场景

本文指导您在时序数据库 CTSDB 控制台中对数据库的每个账号进行密码重置操作。

> **注意：**
> - 数据库密码建议定期更换，最长间隔不超过3个月。
> - 数据库密码规格需要8个 - 64个字符，需包含英文字母、数字和~!@#$%^&*()_+-=|{}[]:;<>,.?/字符中的3种。

操作步骤

1. 登录 CTSDB 控制台，选择对应地域，在实例列表，单击**实例 ID**，进入数据库管理页面。
2. 在数据库管理页面，选择**账号管理**页，找到需要重置密码的账号，单击**重置密码**。

3. 在弹出的对话框，输入新密码和确认密码，单击**确定**。

销毁实例

最近更新时间：2024-08-28 16:31:38

操作场景

根据业务需求，您可以在控制台手动销毁按量计费和 提交工单 销毁包年包月实例。

- 按量计费先按需申请资源使用，在结算时会按您的实际资源使用量收取费用，如需退还实例，可对该实例进行销毁，实例销毁后，该实例将不再产生任何费用。
- 包年包月是一种预付费的形式，如果您在购买包年包月云数据库后有任何不满意，腾讯云支持退货退款。

> **注意**
> 为防止误操作，按量计费实例手动销毁后会保留24小时，此时实例会正常计费。

操作步骤

1. 登录 CTSDB 控制台，在实例列表，选择需要销毁的实例，在**操作**列选择**更多 > 销毁**。

2. 在弹出的对话框，勾选同意规则，单击**立即销毁**。

数据查询

最近更新时间：2024-09-09 14:14:21

时序数据库 CTSDB 提供如下两种数据查询方式：

控制台查询

请参见基本操作下的 查询数据 模块。

API 查询

请参见 HTTP API 参考下的 查询数据 模块。

监控指标

最近更新时间：2024-08-30 16:37:12

指标	标识	单位
写入速度	index_speed	个/s
平均磁盘使用率	disk_usage_avg	%
数据当前写入总次数	index_total	次
平均 JVM 内存使用率	jvm_mem_usage_avg	%
最大 JVM 内存使用率	jvm_mem_usage_max	%
平均 CPU 使用率	cpu_usage_avg	%
最大 CPU 使用率	cpu_usage_max	%
查询拒绝率	search_rejected_completed_percent	%
写入拒绝率	bulk_rejected_completed_percent	%

网络连接

最近更新时间：2024-08-30 11:21:31

时序数据库 CTSDB 目前只支持一种访问实例的网络模式：VPC 网络 。通过提供 VPC 网络隔离和访问时提供用户名和密码的身份认证方式来保证数据的安全性。
实例创建后，每个 CTSDB 实例拥有一个内网地址和端口。您初始化实例时，设置了 root 账号的密码后就可以通过 RESTful API 接口访问该实例。

访问管理
访问管理概述

最近更新时间：2023-10-12 15:12:31

存在问题

如果您在腾讯云中使用到了云服务器、私有网络、云数据库等多项服务，这些服务由不同的人管理，但都共享您的云账号密钥，将存在如下问题：

- 您的密钥由多人共享，泄密风险高。
- 您无法限制其它人的访问权限，易产生误操作造成安全风险。

解决方案

您可以通过子账号实现不同的人管理不同的服务来规避以上的问题。默认情况下，子账号没有使用云服务的权利或者相关资源的权限。因此，我们就需要创建策略来允许子账号使用他们所需要的资源或权限。

访问管理（Cloud Access Management，CAM）可以帮助您安全、便捷地管理对腾讯云服务和资源的访问。您可以使用 CAM 创建子用户、用户组和角色，并通过策略控制其访问范围。CAM 支持用户和角色 SSO 能力，您可以根据具体管理场景针对性设置企业内用户和腾讯云的互通能力。

您最初创建的腾讯云主账号，拥有整个账号全部腾讯云服务和资源的完全访问权限，建议您保护好主账号的凭证信息，日常使用子用户或角色进行访问，并开启多因素校验和定时轮换密钥。

当您使用 CAM 的时候，可以将策略与一个用户或一组用户关联起来，策略能够授权或者拒绝用户使用指定资源完成指定任务。有关 CAM 策略的更多相关基本信息，请参见 策略语法，更多使用信息，请参见 策略。

若您不需要对子账户进行云数据库相关资源的访问管理，您可以跳过此章节。跳过这些部分不会影响您对文档中其余部分的理解和使用。

快速入门

CAM 策略必须授权使用一个或多个 CTSDB 操作，或者必须拒绝使用一个或多个 CTSDB 操作，同时还必须指定可以用于操作的资源（可以是全部资源，某些操作也可以是部分资源），策略还可以包含操作资源所设置的条件。

> **说明**
> 建议用户使用 CAM 策略来管理 CTSDB 资源和授权 CTSDB 操作，对于存量分项目权限的用户体验不变，但不建议再继续使用分项目权限来管理资源与授权操作。

授权策略语法

最近更新时间：2024-05-16 17:00:42

CAM 策略语法

```
{
        "version":"2.0",
        "statement":
        [
           {
              "effect":"effect",
              "action":["action"],
              "resource":["resource"],
               "condition": {"key":{"value"}}
           }
       ]
}
```

- **版本 version**：必填项，目前仅允许值为"2.0"。
- **语句 statement**：用来描述一条或多条权限的详细信息。该元素包括 effect、action、resource、condition 等多个其他元素的权限或权限集合。一条策略有且仅有一个 statement 元素。
- **影响 effect**：必填项，描述声明产生的结果是“允许”还是“显式拒绝”。包括 allow（允许）和 deny（显式拒绝）两种情况。
- **操作 action**：必填项，用来描述允许或拒绝的操作。操作可以是 API 或者功能集（一组特定的 API，以 permid 前缀描述）。
- **资源 resource**：必填项，描述授权的具体数据。资源是用六段式描述，每款产品的资源定义详情会有所区别。
- **生效条件 condition**：必填项，描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、IP 地址等信息，有些服务允许您在条件中指定其他值。

CTSDB 的操作

在 CAM 策略语句中，您可以从支持 CAM 的任何服务中指定任意的 API 操作。对于 CTSDB，请使用以 name/ctsdb: 为前缀的 API 。如果您要在单个语句中指定多个操作的时候，请使用逗号将它们隔开，如下所示：

```
"action":["name/ctsdb:action1","name/ctsdb:action2"]
```

您也可以使用通配符指定多项操作。例如，您可以指定名字以单词 "Describe" 开头的所有操作，如下所示：

```
"action":["name/ctsdb:Describe*"]
```

如果您要指定 CTSDB 中所有操作，请使用 * 通配符，如下所示：

```
"action": ["name/ctsdb:*"]
```

CTSDB 的资源路径

每个 CAM 策略语句都有适用于自己的资源。
资源路径的一般形式如下：

```
qcs:project_id:service_type:region:account:resource
```

- **qcs**：qcloud service 的简称，表示是腾讯云的云资源。该字段是必填项。
- **project_id**：描述项目信息，仅为了兼容 CAM 早期逻辑，无需填写。
- **service_type**：产品简称，如 ctsdb 。
- **region**：地域信息，如 bj。

- **account**：资源拥有者的主账号信息，如 uin/12345678。
- **resource**：各产品的具体资源详情，如 instance/instance_id 或者 instance/*。

可授权的资源类型

最近更新时间：2024-08-28 11:09:43

时序数据库 CTSDB 支持资源级授权，您可以指定子账号拥有特定资源的接口权限。
支持资源级授权的接口列表如下：

> **说明**
> 表中未列出的云数据库 API 操作，即表示该 CTSDB API 操作不支持资源级权限。针对不支持资源级权限的 CTSDB API 操作，您仍可以向用户授予使用该操作的权限，但策略语句的资源元素必须指定为 *。

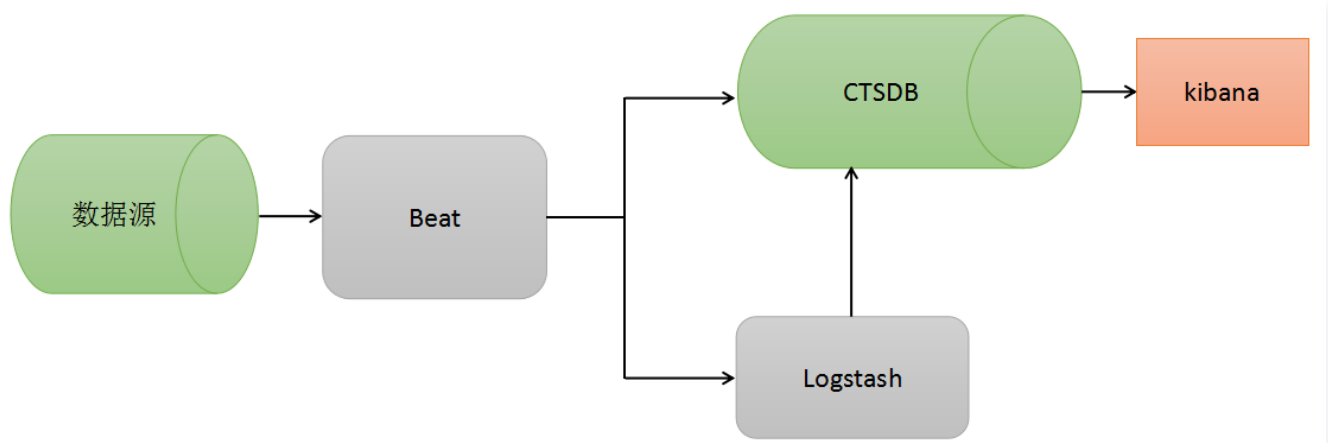
API 名	API 描述	资源六段式示例
ModifyDBInstanceUserPassword	修改数据库实例用户密码	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
DescribeDBInstances	查询实例列表	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
InitDBInstance	初始化数据库实例	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
ModifyDBInstanceName	修改数据库实例名称	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
ModifyDBInstanceProject	修改数据库实例项目	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
RecycleDBInstance	回收数据库实例	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
DescribeDBInstanceMetricInfo	查询数据库实例 Metric 信息	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
DescribeDBInstanceMetricList	查询数据库实例 Metric 列表	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`
DescribeDBInstanceMetricQuery	查询数据库实例 Metric 查询	`qcs::ctsdb:$region:$account:instanceId/*` `qcs::ctsdb:$region:$account:instanceId/$instanceId`

实践教程
CTSDB 对接 ELK 组件及 Grafana

最近更新时间：2024-05-15 11:23:16

概述

云数据库 CTSDB 是一款分布式、可扩展、支持近实时数据搜索与分析的时序数据库，且兼容 ELK 生态组件，您可以非常方便的使用 ELK 组件与 CTSDB 对接。
ELK 组件提供了丰富的数据处理功能，包括数据采集、数据清洗、可视化图形展示等。常用的 ELK 生态组件包括 Filebeat、Logstash、Kibana。同时，CTSDB 也支持 Grafana 作为可视化平台。常见架构图如下：



组件的使用

Filebeat

Filebeat 是一个轻量级开源日志文件数据搜集器，作为 agent 安装到服务器上，Filebeat 读取文件内容，发送到 Logstash 进行解析后进入 CTSDB，或直接发送到 CTSDB 进行集中式存储和分析。

Filebeat 的使用流程

1. **安装**
   Filebeat 安装介绍请参见 该地址 。
2. **配置**
   Filebeat 的配置采用 YAML 格式文件，主要配置为全局配置、输入配置、输出配置，下节会给出使用样例。
3. **启动**
   Filebeat 启动时可以指定配置文件路径，若不指定则默认使用 filebeat.yml。

Filebeat 使用示例

1. 首先，将 Filebeat 的安装包解压缩到某一目录，如下所示：

```
[user_02@TENCENT64 ~/filebeat]$ ls
NOTICE  README.md  data  filebeat  filebeat.full.yml  filebeat.template-es2x.json  filebeat.template-es6x.json  filebeat.template.json  filebeat.yml  logs  module  scripts
```

2. 然后，配置 filebeat.yml，配置参考如下：

```
filebeat.shutdown_timeout: 5 # How long filebeat waits on shutdown for the publisher to finish.
max_procs: 4 # 可同时执行的最大cpu数，默认为操作系统可用的逻辑cpu数
filebeat.spool_size: 102400
filebeat.idle_timeout: 2s
processors:
- drop_fields: # 需要drop掉的字段
  fields: ["beat","input_type","source","offset"]
filebeat.prospectors:
- paths: ["/data/log/filebeat-tutorial.log"]    # 样例数据所在的路径
```

```
  fields:
    metricname: metric1
  harvester_buffer_size: 1638400
  close_timeout: 0.5h
  scan_frequency: 2s
- paths: ["/mylog/*.log","/mylog1/*.log"]
  fields:
    metricname: table2
  harvester_buffer_size: 1638401
  close_timeout: 0.5h
  scan_frequency: 2s
output.elasticsearch:
  hosts: ["127.0.0.1:9200"]
  index: "%{[fields.indexname]}"  # 通配，可以达到不同类别的数据写入不同index的目的
  username: "root" # 对于有权限的CTSDB这里需要填用户名和密码
  password: "changeme"
  worker: 2 # 工作线程数
  loadbalance: true # 是否开启负载均衡
  bulk_max_size: 512 # 一次bulk的最大文档数
  flush_interval: 2s
  template:
    enabled: false  # 注意：Filebeat启动后会put一个默认的template，对接CTSDB时，需要禁用Filebeat的template
```

部分样例数据如下：

```
83.149.9.216 - - [04/Jan/2015:05:13:42 +0000] "GET /presentations/logstash-monitorama-
2013/images/kibana-search.png HTTP/1.1" 200 203023 "http://semicomplete.com/presentations/logstash-
monitorama-2013/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/32.0.1700.77 Safari/537.36"
83.149.9.216 - - [04/Jan/2015:05:13:42 +0000] "GET /presentations/logstash-monitorama-
2013/images/kibana-dashboard3.png HTTP/1.1" 200 171717 "http://semicomplete.com/presentations/logstash-
monitorama-2013/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/32.0.1700.77 Safari/537.36"
83.149.9.216 - - [04/Jan/2015:05:13:44 +0000] "GET /presentations/logstash-monitorama-
2013/plugin/highlight/highlight.js HTTP/1.1" 200 26185 "http://semicomplete.com/presentations/logstash-
monitorama-2013/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/32.0.1700.77 Safari/537.36"
```

3. **启动 Filebeat，并观察 CTSDB 中对应表的数据：**

```
nohup ./filebeat &
    less logs/filebeat # 查看部分日志，通过日志libbeat.es.published_and_acked_events=100可以看出我们的100条日志都成功写入到es中
    2018-05-25T14:32:24+08:00 INFO Non-zero metrics in the last 30s: filebeat.harvester.open_files=1
filebeat.harvester.running=1 filebeat.harvester.started=1 libbeat.es.call_count.PublishEvents=1
libbeat.es.publish.read_bytes=1535 libbeat.es.publish.write_bytes=40172
libbeat.es.published_and_acked_events=100 libbeat.publisher.published_events=100 publish.events=101
registrar.states.current=1 registrar.states.update=101 registrar.writes=2

    # 通过kibana或curl查看es中是否有数据写入到metric1
    # 命令：
    GET metric1/_search
    {
      "sort": [
    {
      "@timestamp": {
    "order": "desc"
      }
    }
```

```
        ],
        "docvalue_fields": ["@timestamp", "message"]
      }
      # 结果:
      {
        "took": 2,
        "timed_out": false,
        "_shards": {
      "total": 3,
      "successful": 3,
      "skipped": 0,
      "failed": 0
        },
        "hits": {
      "total": 100,
      "max_score": null,
      "hits": [
        {
      "_index": "metric1@1525536000000_30",
      "_type": "doc",
      "_id": "AWOV_o1wBzkw2jsSfrLN",
      "_score": null,
      "fields": {
        "@timestamp": [
      1527229914629
        ],
        "message": [
      "218.30.103.62 - - [04/Jan/2015:05:27:57 +0000] \"GET /blog/geekery/c-vs-python-bdb.html
HTTP/1.1\" 200 11388 \"-\" \"Sogou web spider/4.0(+http://www.sogou.com/docs/help/webmasters.htm#07)\""
        ]
      },
      "sort": [
        1527229914629
      ]
        },
      # 内容太多，这里省略，通过hits.total可以看出，查询命中了100条文档，证明100条log都成功写入CTSDB`
```

上述示例是直接通过 Filebeat 将原始日志数据写入到 CTSDB 中，并没有做字段的解析，下节将会介绍通过 Logstash 解析数据，然后写入 CTSDB。

Logstash

Logstash 是一款具有实时数据解析功能的开源数据收集引擎。Logstash 能够搜集各种数据源，并对数据进行过滤、分析、格式化等操作，然后存储到 CTSDB。

Logstash 使用流程

1. **安装**
   Logstash 安装请参见 该地址 。
2. **配置**
   Logstash 的主要配置包含三个模块，分别为数据源输入，数据解析规则，数据输出。下节会给出使用样例。
3. **启动**
   Logstash 启动时，可以指定配置文件，否则，默认使用 logstash.yml 作为配置，解析规则默认使用 pipelines.yml 中的配置。

Logstash使用示例

1. 首先，将 Logstash 的安装包解压缩到某一目录，如下所示：

```
[user_00@TENCENT64 ~/luckie/logstash-6.2.4]$ ls
bin  config  CONTRIBUTORS  data  first-pipeline.conf  Gemfile  Gemfile.lock  lib  LICENSE  logs  logstash-core  logstash-core-plugin-api  modules  nohup.out  NOTICE.TXT  tools  vendor
```

2. 然后，创建一个配置文件，当然也可以在 logstash.yml 和 pipelines.yml 中进行配置。这里创建一个配置文件名为 first-pipeline.conf，配置如下：

```
# 输入源
input {
 beats {
     port => "5044"
 }
}
# 解析过滤
filter {
 grok {
     match => {
         "message" => "%{COMBINEDAPACHELOG}"
     }
 }
}
# 输出
output {
 elasticsearch {
     action => "index"
     hosts => ["localhost:9200"]
     index => "logstash_metric" # CTSDB中创建的metric名
     document_type => "doc"
     user => "root" # 对于有权限的CTSDB需要指定用户名和密码
     password => "changeme"
 }
}
```

grok filter 插件在 Logstash 默认可用的，其能够将非结构化的数据解析为结构化的数据，具体使用参考 文档 。

3. 启动 Logstash、Filebeat，并观察 CTSDB 中对应表的数据：

```
  # 这里需要注意的是，Filebeat的输出是Logstash，因此Filebeat的输出项配置改为：
 output.logstash:
   hosts: ["localhost:5044"]
 # 清空Filebeat的data目录，启动Filebeat
 rm data/registry
 nohup ./filebeat &
 # 启动Logstash
 nohup bin/logstash -f first-pipeline.conf --config.reload.automatic &
 # 通过kibana或curl查看CTSDB中是否有数据写入到metric1
 # 命令：
 GET logstash_metric/_search
 {
   "sort": [
     {
       "@timestamp": {
         "order": "desc"
       }
     }
   ],
   "docvalue_fields": ["@timestamp","request", "response", "type", "bytes",  "verb", "agent",
"clientip"]
 }
# 结果：
 {
   "took": 0,
   "timed_out": false,
   "_shards": {
     "total": 0,
     "successful": 0,
     "skipped": 0,
```

```
      "failed": 0
    },
    "hits": {
      "total": 0,
      "max_score": 0,
      "hits": []
    }
  }
  # 这里发现结果是空的，说明没有数据没有写入到CTSDB，查看Logstash日志：
  [2018-05-25T21:00:07,081][ERROR][logstash.outputs.elasticsearch] Encountered a retryable error. Will
Retry with exponential backoff  {:code=>403, :url=>"http://127.0.0.1:9200/_bulk"}
  [2018-05-25T21:00:07,081][ERROR][logstash.outputs.elasticsearch] Encountered a retryable error. Will
Retry with exponential backoff  {:code=>403, :url=>"http://127.0.0.1:9200/_bulk"}
  # 发现bulk出错，并返回403权限错误，仔细验证 用户名和密码，发现并无问题，继续查看es日志：
  [2018-05-25T20:59:27,545][WARN ][o.e.p.o.OPackActionFilter] [1505480279000001609] process index
failed: Invalid format: "2018-05-25T12:51:18.905Z"
  [2018-05-25T20:59:27,547][WARN ][o.e.p.o.OPackActionFilter] [1505480279000001609] process index
failed: Invalid format: "2018-05-25T12:51:18.905Z"
  # 从es的日志可以看出 ，时间格式解析出错。出错的原因是，笔者建metric时没有指定时间字段的格式，那么CTSDB默认为
epoch_millis，因此需要修改下时间格式：
  POST /_metric/logstash_metric/update?pretty
  {
    "time": {
           "name": "@timestamp",
           "format": "strict_date_optional_time"
      }
  }
  # 重启Logstash、Filebeat重新写入数据，再查看CTSDB：
  # 结果
  {
    "took": 2,
    "timed_out": false,
    "_shards": {
      "total": 3,
      "successful": 3,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": 100,
      "max_score": null,
      "hits": [
        {
          "_index" : "logstash_metric@1527004800000_30",
          "_type" : "doc",
          "_id" : "AWOvG0YgQoCkIV2BLcov",
          "_score" : null,
          "fields" : {
            "request" : [
              "/blog/tags/puppet?flav=rss20"
            ],
            "agent" : [
              "\"UniversalFeedParser/4.2-pre-314-svn +http://feedparser.org/\""
            ],
            "@timestamp" : [
              1527651156725
            ],
            "response" : [
              "200"
            ],
            "bytes" : [
```

```
            14872
          ],
          "clientip" : [
            "46.105.14.53"
          ],
          "verb" : [
            "GET"
          ],
          "type" : [
            "log"
          ]
        },
        "sort" : [
          1527651156725
        ]
      },
    ... ...
    # 内容太多，这里省略，通过hits.total可以看出，查询命中了100条文档，证明100条log都成功写入CTSDB
```

从上述示例，我们可以看出，通过 Filebeat 采集数据到 Logstash，然后利用 Logstash 的数据解析功能，将日志解析为多个字段，然后写入 CTSDB。

Kibana

Kibana 是一个旨在为 Elasticsearch 设计的开源的分析和可视化平台。可以使用 Kibana 来搜索，查看存储在 CTSDB metric 中的数据并与其进行交互。可以利用 Kibana 中丰富的图表、表格、曲线等功能来可视化数据并进行数据分析。

Kibana 使用流程

1. **安装**
   下载与 Elasticseach 对应的 Kibana 版本，并解压到某一目录，兼容的具体版本可参考 查看实例详情 在控制台查看。
2. **配置**
   Kibana 的配置很简单，下节会给出样例。具体配置项含义参考 该地址 。
3. **运行**
   Kibana 运行时，默认使用 config/kibana.yml 作为配置。

Kibana使用示例

1. 首先，将 Kibana 的安装包解压到某一目录，如下所示

```
[user_00@TENCENT64 ~/repository/kibana]$ ls
bin  config  data  kibana.log  kibana.nohup  LICENSE.txt  logs  node  node_modules  nohup.out  optimize  package.json  plugins  plugins.bk  README.txt  src  webpackShims
```

2. 然后，修改 config 下的配置文件。主要配置如下：

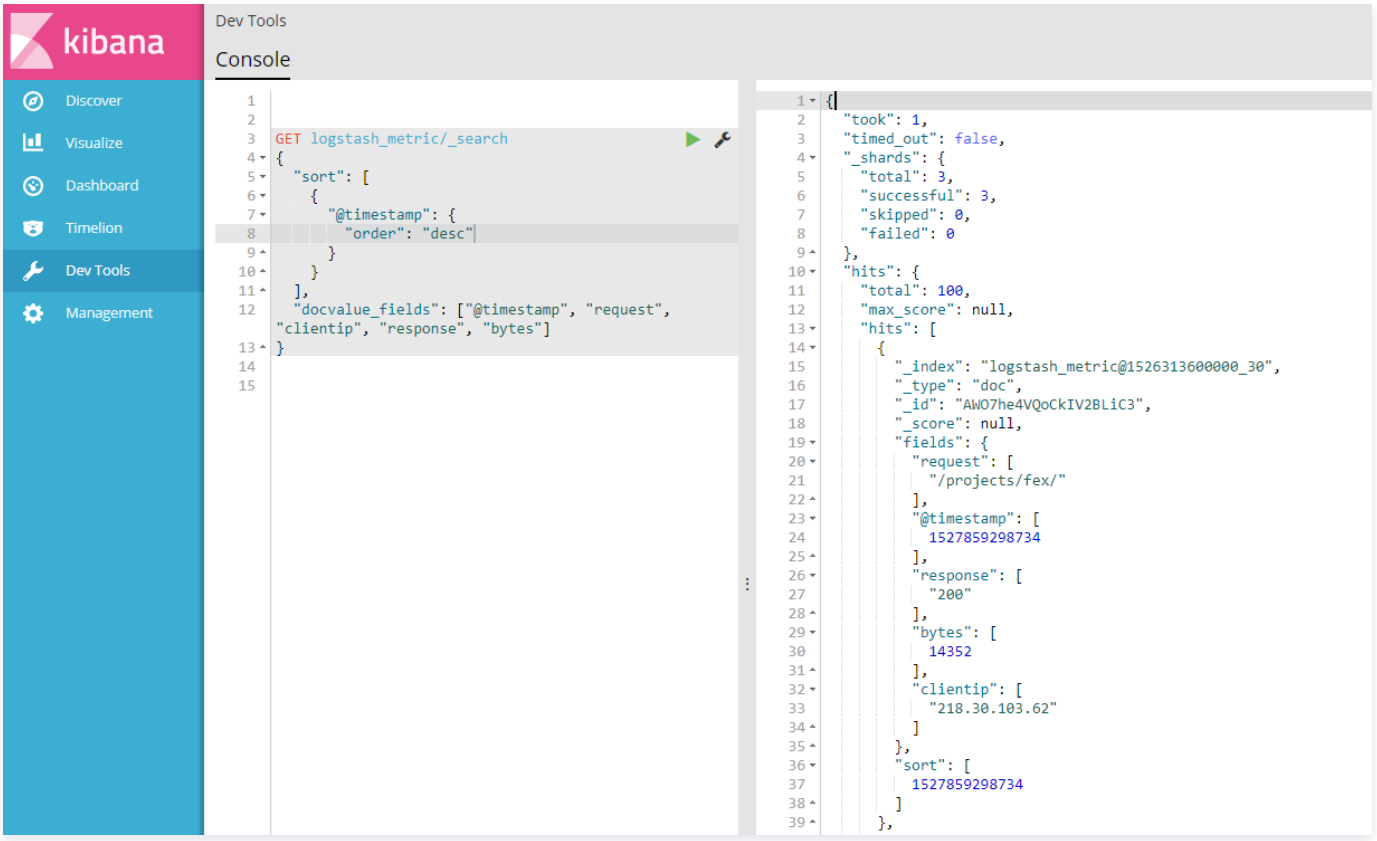
```
# config/kibana.yml
# Kibana server监听的端口
server.port: 5601
# Kibana server所绑定的服务器ip
server.host: 127.0.0.1
# 所要连接的CTSDB的url
elasticsearch.url: "http://127.0.0.1:9200"
# 若使用带权限的CTSDB，则需要指定用户名和密码
elasticsearch.username: "root"
elasticsearch.password: "changeme"
```

- 启动，并通过浏览器访问 kibana

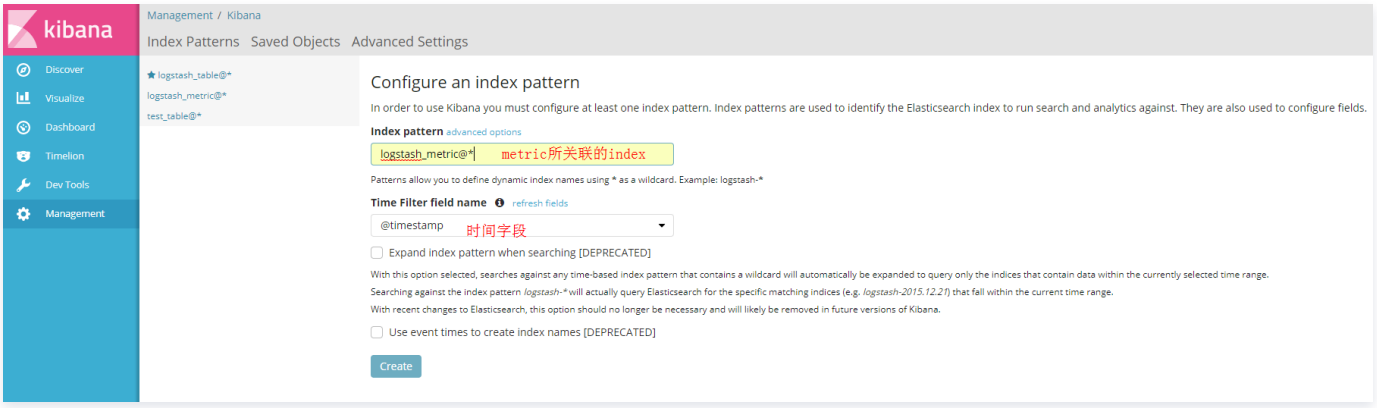
```
nohup bin/kibana &
```

利用 ip:port 或者域名访问 kibana server，如下：

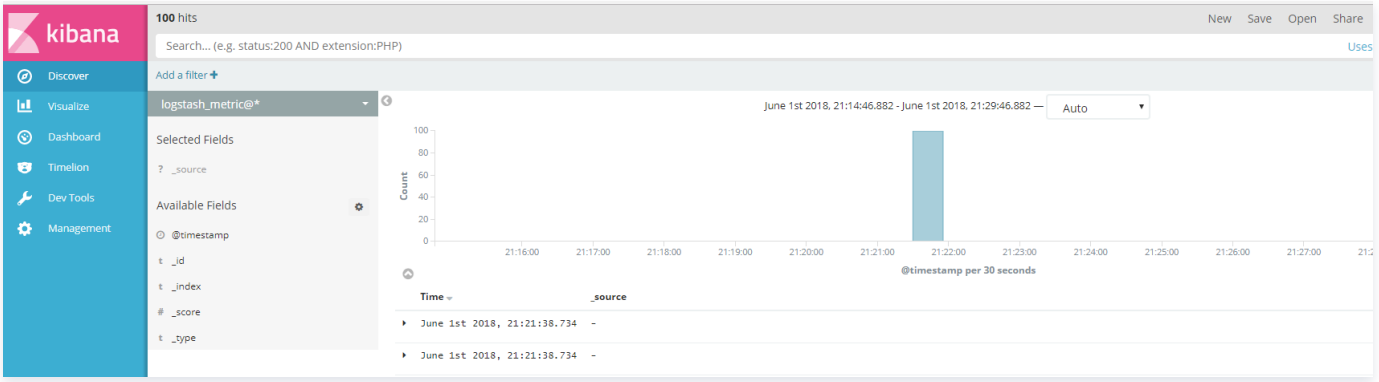
利用开发工具，我们可以很方便的访问 CTSDB，如下图所示：



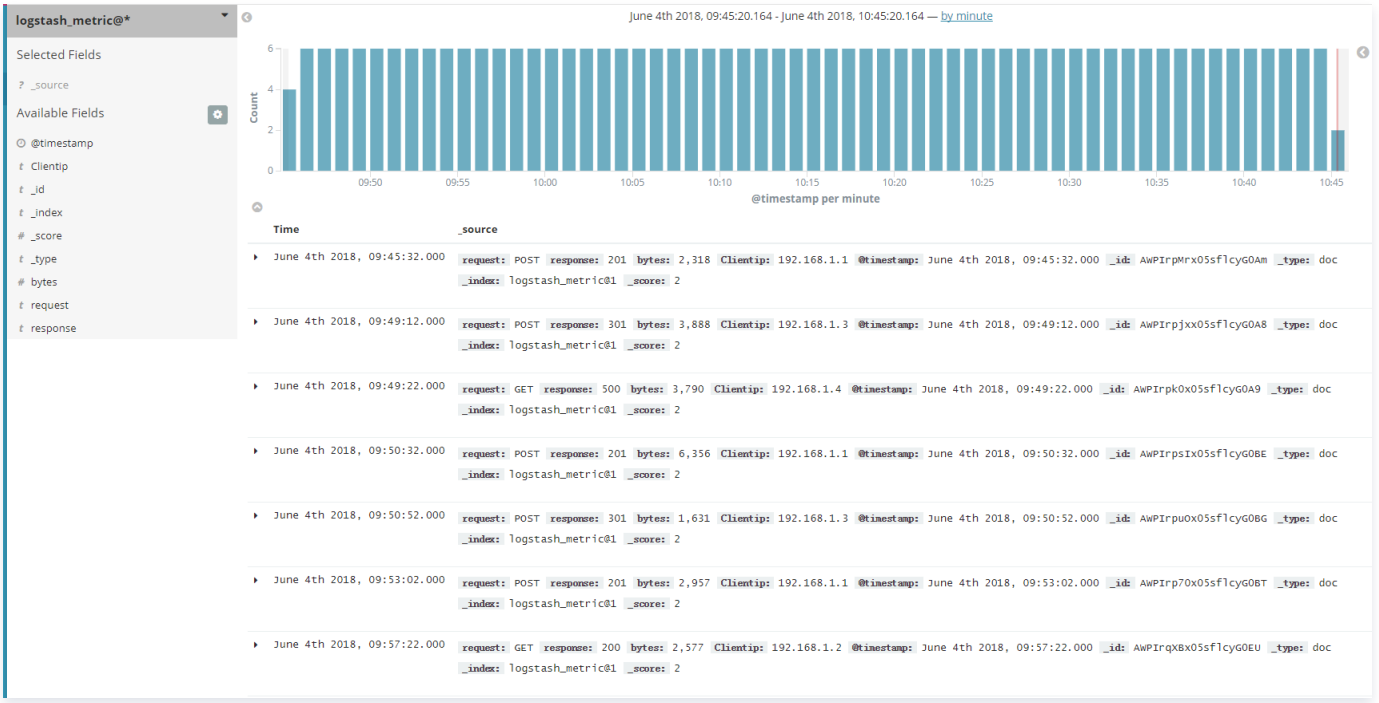
在管理页面创建需要访问的索引，如下所示：



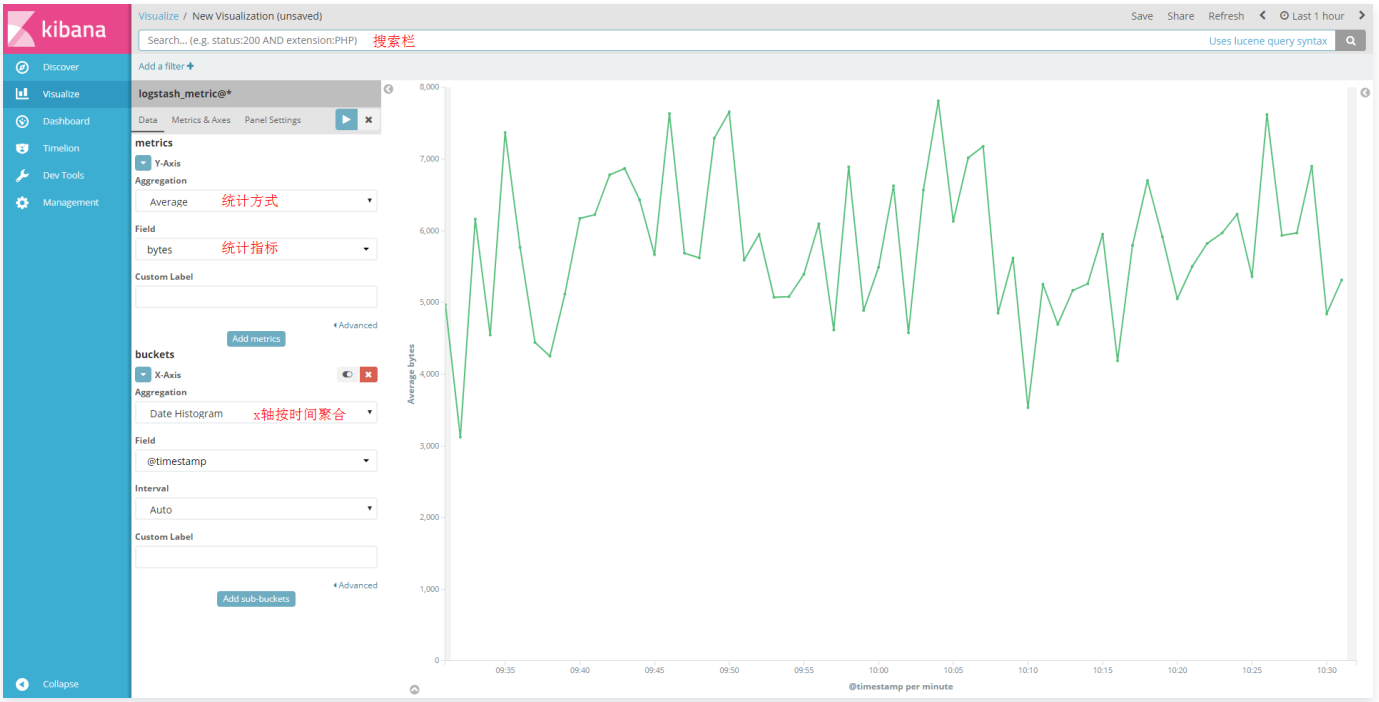
如果您有日志搜索的需求，可能会使用到搜索功能，如下图所示：



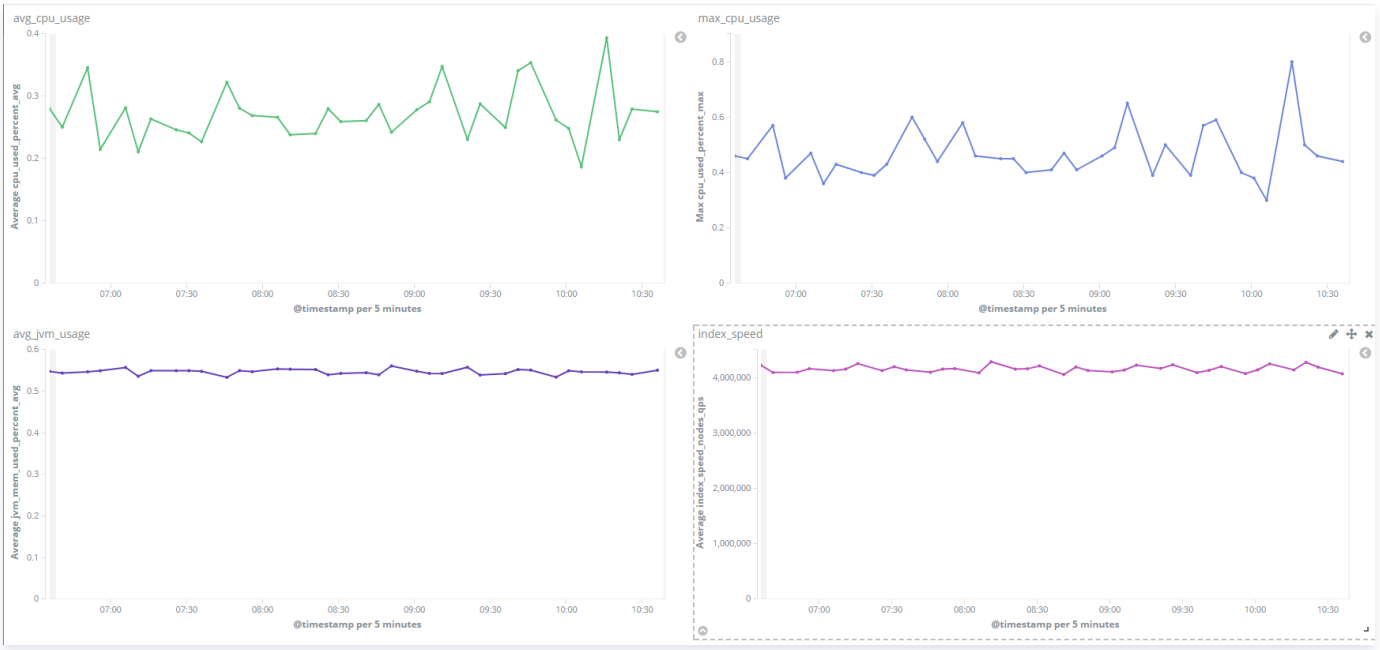
从上图，我们发现，搜索的结果出了时间以外，没有其他，原因是 CTSDB 为了节省存储空间默认没有开启 source 功能，如果您有日志搜索的需求，请联系我们开启 source。开启 source 后的效果如下图所示：



利用可视化，我们可以构建各种图表，这里以创建 Line Chart 为例，展示 Kibana 的图表效果，如下所示：



Kibana 不仅提供了丰富的可视化图表功能，而且还可以利用仪表盘将我们保存的可视化图表统一展示都一个页面上，非常方便地查看多个指标的变化状态。这里为了展示效果，贴一张我们内部真实的监控数据的仪表盘视图，如下所示：



Grafana

Grafana 是一款开源的仪表盘工具，它提供了丰富的图表功能，类似 Kibana，利用 Grafana 精细的展示效果，可以帮助您有效地进行数据分析。和 Kibana 不同的是，Grafana 支持的数据源种类更多，包含 influxdb、opentsdb、Elasticsearch，下面演示利用 Grafana 来可视化的分析 CTSDB 中的数据。

Grafana

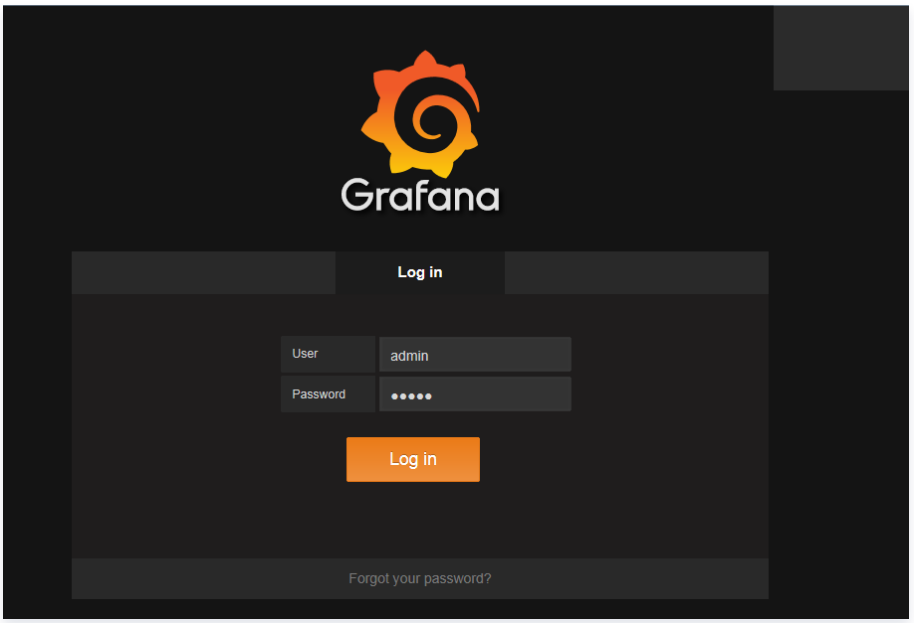
1. 安装
   Grafana 的安装请参见 官方文档 。
2. 配置
   Grafana 的配置项较多，可以使用默认的配置，具体配置说明参考 该地址 。
3. 运行
   Grafana 运行时，默认使用 /etc/grafana/grafana.ini 作为配置。

Grafana 使用示例

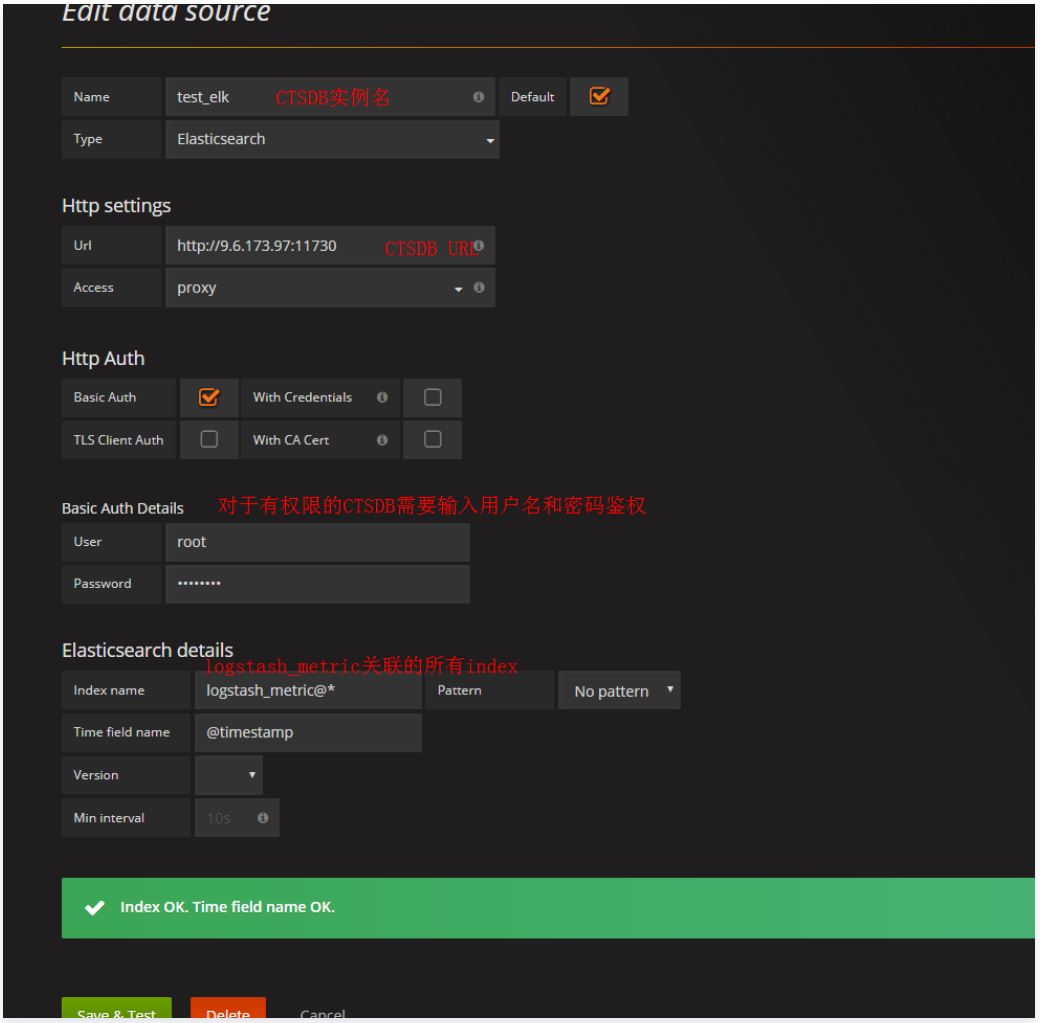
1. 首先，启动 grafana 服务。

```
sudo service grafana-server start
```

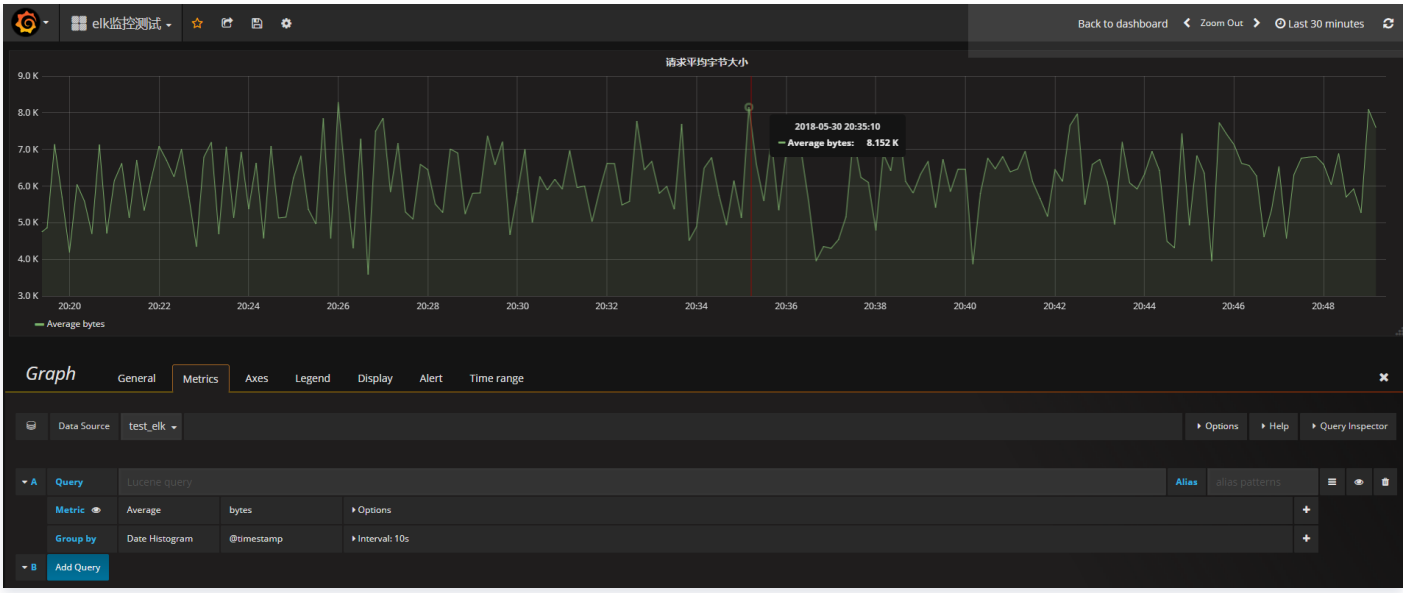
2. 然后，通过浏览器访问 Grafana 服务：



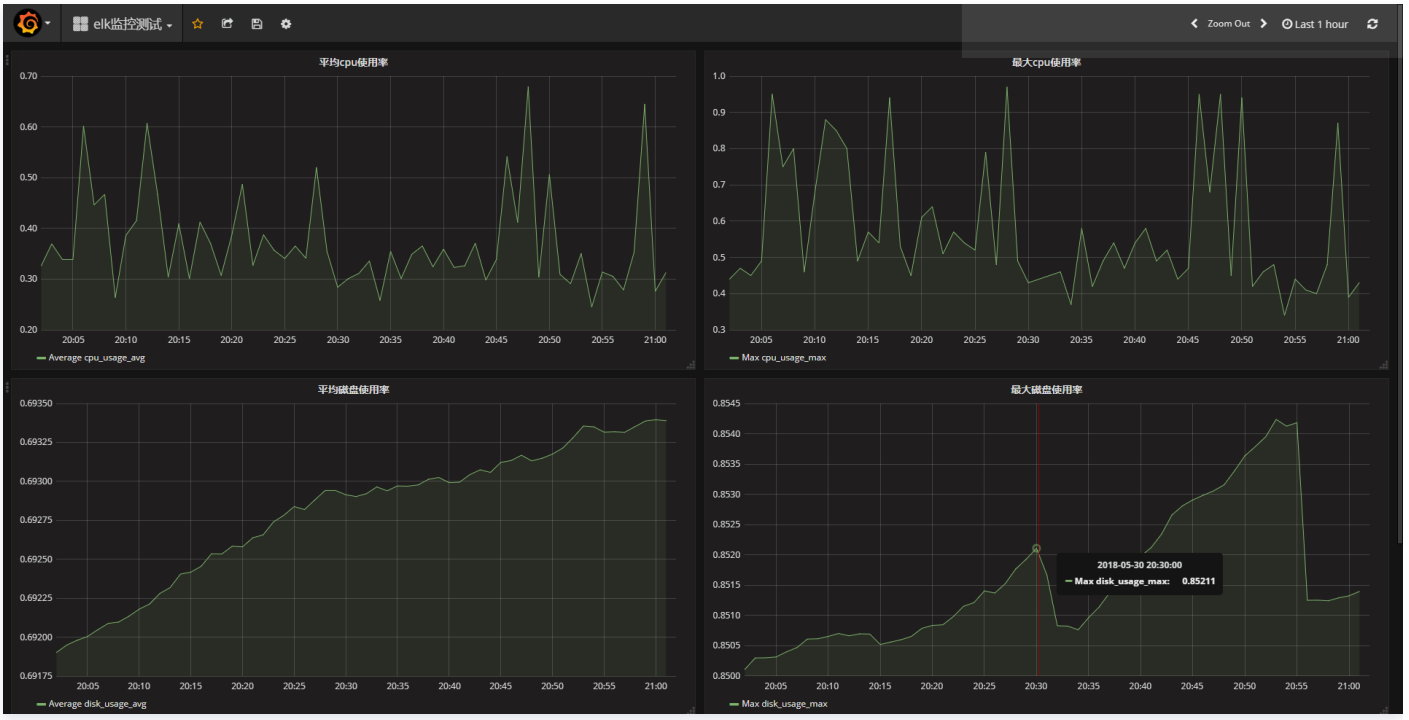
3. 创建数据源，建立 dashboard，如下所示：



4. 利用 dashboard 创建可视化图表，如下图所示：



5. 从上图可以看出，Grafana 的图表展示效果和 Kibana 略有区别，但是功能本质上是一样的，这个看您的个人使用习惯和爱好。同样，Grafana 的 dashboard 也能同时展示多个可视化图表，如下图所示：



小结

以上为 ELK 生态组件及 Grafana 对接 CTSDB 的详细使用过程，如在使用过程中遇到问题，可通过 在线支持 解决。

MySQL 数据导入 CTSDB

最近更新时间：2024-08-30 15:46:32

前言

CTSDB 是一款分布式、可扩展、支持近实时数据搜索与分析的时序数据库，且兼容 Elasticsearch 常用的 API 接口。对于很多用户，想要将 MySQL 中的数据导入到 CTSDB 中，而又找不到一种较好的方法，这里给出一种简单快捷的方式，轻松将 MySQL 中的数据同步到 CTSDB。

工具介绍

go-mysql-elasticsearch 是一款开源的高性能的 MySQL 数据同步 Elasticsearch 的工具，其由 go 语言开发，编译及使用都非常简单。go-mysql-elasticsearch 的原理也很简单，首先使用 mysqldump 获取当前 MySQL 的数据，然后在通过此时 binlog 的 name 和 position 获取增量数据，再根据 binlog 构建 restful api 写入数据到 Elasticsearch 中。由于 CTSDB 基于 Elasticsearch 开发，因此，可以完美对接 go-mysql-elasticsearch，导入 MySQL 数据。

MySQL 数据同步 CTSDB 步骤

MySQL 样例数据构建

既然读者有 MySQL 导入 CTSDB 的需求，那 MySQL 的安装就不用多说了。这里为了整个流程的完整性，就从样例数据的灌入开始，用 go 写了一个小工具，生成一些样例数据并灌入到 MySQL 中，表结构如下：

```
mysql> desc test_table;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| timestamp | bigint(20)  | YES  |     | NULL    |                |
| cpu_usage | float       | YES  |     | NULL    |                |
| host_ip   | varchar(20) | YES  |     | NULL    |                |
| region    | varchar(20) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

以上创建了一个名为 test_table 的表，然后向该表灌入2000条样例数据，部分数据如下所示：

```
mysql> select * from test_table;
+------+------------+-----------+-------------+-----------+
| id   | timestamp  | cpu_usage | host_ip     | region|
+------+------------+-----------+-------------+-----------+
|1 | 1527676339 |  0.23 | 192.168.1.1 | beijing   |
|2 | 1527676399 |  0.78 | 192.168.1.2 | shanghai  |
|3 | 1527676459 |   0.2 | 192.168.1.3 | guangzhou |
|4 | 1527676519 |  0.47 | 192.168.1.4 | shanghai  |
|5 | 1527676579 |  0.13 | 192.168.1.5 | beijing   |
|6 | 1527676639 |  0.15 | 192.168.1.1 | beijing   |
|7 | 1527676699 |  0.07 | 192.168.1.2 | shanghai  |
|8 | 1527676759 |  0.17 | 192.168.1.3 | guangzhou |
|9 | 1527676819 |  0.94 | 192.168.1.4 | shanghai  |
|10| 1527676879 |  0.06 | 192.168.1.5 | beijing   |
```

至此，MySQL 端的样例数据准备完毕。

CTSDB metric 创建

现在，我们在 CTSDB 上创建一个和 MySQL 一样的表结构，用于存储对应的数据，创建接口如下所示：

```
POST /_metric/test_metric
{
  "time": {
```

```
        "name": "timestamp",    # 与 MySQL 表中的 timestamp 对应，CTSDB 常用的时间域
        "format": "strict_date_optional_time || epoch_second"
      },
      "tags": {
        "region": "string",
        "host_ip": "string"
      },
      "fields": {
        "cpu_usage": "float"    # fields 域代表指标列，很明显 cpu_usage 代表需要监控 CPU 使用率指标
      }
    }
```

至此，CTSDB 中的表结构也准备好了，下面我们使用 go-mysql-elasticsearch 来同步数据。

go-mysql-elasticsearch 使用

由于 go-mysql-elasticsearch 是用 go 语言开发，因此首先安装 go，官方要求的版本是 1.6 以上，go 的安装非常简单，参考官方文档 下载、安装，然后开始安装 go-mysql-elasticsearch，整个步骤如下：

```
$ go get github.com/siddontang/go-mysql-elasticsearch
$ cd $GOPATH/src/github.com/siddontang/go-mysql-elasticsearch
$ make
```

工具安装好后，需要进行一些合理地配置我们才能愉快地使用，下面将会给出一个配置范例，并给予相应地注释说明：

```
# 注意：go-mysql-elasticsearch 的默认配置文件在 go-mysql-elasticsearch/etc/river.toml
    # MySQL address, user and password
    # user must have replication privilege in MySQL.
    my_addr = "127.0.0.1:3306"
    my_user = "root"
    my_pass = "123456"
    my_charset = "utf8"
    # Set true when elasticsearch use https
    #es_https = false
    # CTSDB 地址
    es_addr = "9.6.174.42:13982"
    # 如果使用的是带权限的 CTSDB，需要设置用户名和密码
    es_user = "root"
    es_pass = "changeme"
    # Path to store data, like master.info, if not set or empty,
    # we must use this to support breakpoint resume syncing.
    # TODO: support other storage, like etcd.
    data_dir = "./var"  # 存储的是 binlog 的名字及位置
    # Inner http status address
    stat_addr = "127.0.0.1:12800"
    # pseudo server id like a slave
    server_id = 1001
    # mysql or mariadb
    flavor = "mysql"
    # mysqldump execution path
    # if not set or empty, ignore mysqldump.
    mysqldump = "mysqldump"
    # minimal items to be inserted in one bulk
    bulk_size = 512
    # force flush the pending requests if we don't have enough items >= bulk_size
    flush_bulk_time = "200ms"
    # Ignore table without primary key
    skip_no_pk_table = true
    # MySQL data source
    [[source]]
```

```
    schema = "mysql_es"
    # Only below tables will be synced into Elasticsearch.
    # "t_[0-9]{4}" is a wildcard table format, you can use it if you have many sub tables, like table_0000
- table_1023
    # I don't think it is necessary to sync all tables in a database.
    tables = ["test_*"]
    [[rule]]
    schema = "mysql_es"    # MySQL 数据库名
    table = "test_table"   # MySQL 表名
    index = "test_metric"   # CTSDB 中 metric 名
    type = "doc"           # 文档类型
```

以上配置，为测试所使用的配置，如果您有更高级的需求可以参考官方文档，合理进行配置。配置 ok 后，我们来运行 go-mysql-elasticsearch，如下所示：

```
    $  ./bin/go-mysql-elasticsearch -config=./etc/river.toml
    2018/05/31 21:43:44 INFO  create BinlogSyncer with config {1001 mysql 127.0.0.1 3306 root   utf8 false
false <nil> false false 0 0s 0s 0}
    2018/05/31 21:43:44 INFO  run status http server 127.0.0.1:12800
    2018/05/31 21:43:44 INFO  skip dump, use last binlog replication pos (mysql-bin.000002, 194296) or
GTID %!s(<nil>)
    2018/05/31 21:43:44 INFO  begin to sync binlog from position (mysql-bin.000002, 194296)
    2018/05/31 21:43:44 INFO  register slave for master server 127.0.0.1:3306
    2018/05/31 21:43:44 INFO  start sync binlog at binlog file (mysql-bin.000002, 194296)
    2018/05/31 21:43:44 INFO  rotate to (mysql-bin.000002, 194296)
    2018/05/31 21:43:44 INFO  rotate binlog to (mysql-bin.000002, 194296)
    2018/05/31 21:43:44 INFO  save position (mysql-bin.000002, 194296)
```

这里需要注意，由于 go-mysql-elasticsearch 需要利用 binlog，而且 binlog 一定要变成 row-based format 格式，因此在 MySQL 必须配置如下参数：

```
# binlog 参数必须要配置如下：
log_bin=mysql-bin
binlog_format = ROW
server-id=1
```

现在，我们来看一下 CTSDB 中是否成功导入了 MySQL 中的数据：

```
    GET test_metric/_search?size=1000
    {
      "sort": [
    {
      "timestamp": {
    "order": "desc"
      }
    }
      ],
      "docvalue_fields": ["timestamp", "host_ip", "region", "cpu_usage"]
    }
    #结果：
    {
      "took": 8,
      "timed_out": false,
      "_shards": {
    "total": 3,
    "successful": 3,
    "skipped": 0,
    "failed": 0
      },
```

```
    "hits": {
  "total": 2000,
  "max_score": null,
  "hits": [
    {
  "_index": "test_metric@1525363200000_30",
  "_type": "doc",
  "_id": "2000",
  "_score": null,
  "fields": {
    "host_ip": [
  "192.168.1.5"
    ],
    "region": [
  "beijing"
    ],
    "cpu_usage": [
  0.05000000074505806
    ],
    "timestamp": [
  1527807286000
    ]
  },
  "sort": [
    1527807286000
  ]
    },
    ......
  }
```

小结

可以看到，使用 go-mysql-elasticsearch，仅需要在配置文件里面写规则，就能非常方便的将数据从 MySQL 同步给 ES。上面仅举了一些简单的例子，如果有更多的需求可以参考 go-mysql-elasticsearch的官方文档。

除了本文所介绍的工具外，这里再推荐两种工具：

- py-mysql-elasticsearch-sync，该工具是使用 Python语言编写，与 go-mysql-elasticsearch 的原理类似，都是使用 binlog 来实现数据的同步，安装及使用见 官方文档 。
- logstash，使用 logstash 同步数据时需要安装 logstash-input-jdbc、logstash-output-elasticsearch 两个插件，具体使用参考 官方文档 、elastic 官方文档 。

  如果您在使用上述工具中遇到问题，可通过 在线支持 解决。

快速选择实例

最近更新时间：2024-09-09 14:17:01

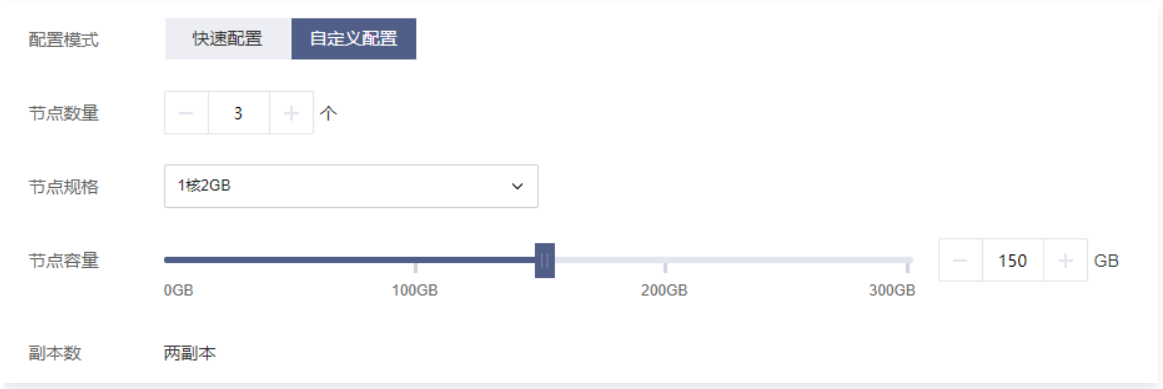
时序数据库 CTSDB 支持快速配置和自定义配置两种选择模式，下面分别详细叙述。建议您主要从三个方面需求来考虑如何选择合适的实例规格：性能需求、容量需求、其他要求。实例性能请参考 产品性能 。
进入 CTSDB 购买页，在选择模式处可按需选择对应模式。

快速配置

时序数据库 CTSDB 提供快速配置模式，该模式下，您只需要选择所存储的数据量，系统自动根据3节点两副本的模式来创建实例。另外，系统自动选择节点配置时，在节点存储容量相同的前提下，会优先选择 CPU 和内存较小的节点配置来分配实例。

自定义配置

在自定义配置模式下，您可自主选择节点数量，节点规格和节点容量。具体如下图所示：



注意：

在生产环境中，选择节点数量与节点规格，建议如下：

- 节点数量：配置至少3个节点。仅有2个节点的实例，无法完全保证数据的高可靠性，仅适用于测试环境。
- 节点规格：配置至少2核9GB。节点规格为1核2GB，内存过小无法保证高可用性，仅适用于测试环境。

HTTP API 参考
CTSDB 2.0
简介

最近更新时间：2024-07-30 16:35:13

CTSDB 2.0 支持 HTTP 协议进行数据写入和查询等操作，以下仍称为 CTSDB。提供的 HTTP API 是 RESTful API，对资源的请求方式是通过向资源对应的 URI 发送标准的 HTTP 请求，例如 GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源等。用户通过 HTTP API 几乎可以实现所有的数据操作。CTSDB 通过提供 VPC 网络隔离和访问时提供用户名和密码的身份认证方式来保证数据的安全性，通过 JSON 格式的结构体进行数据交换，每个请求都会返回一个标准的 HTTP 响应状态码和响应内容。若操作失败，用户可以根据响应内容获取到具体错误信息。通过 RESTful API 接口连接实例方式请参见 连接实例 。

命名规则

CTSDB 中 metric（表）、tags（维度列）、fields（数据列）的命名应简明扼要，tags、fields 详细介绍可参见 请求内容：

- **metric 命名规则**：允许使用小写英文字母、数字、 _（下划线）、 -（短划线）、 .（圆点）的组合，且不能以 .（圆点）、_（下划线）或 -（短划线）开头。长度为1到200个字符。
- **metric 中 tags 和 fields 命名规则**：允许大小写英文字母、数字、 _（下划线）、 -（短划线）、 .（圆点）的任意组合，且不能以 .（圆点）开头。长度为1到255个字符。

系统限制

- **metric 中 tags 和 fields 限制**：metric 中字段总数不超过1000。
- **批量写入 metric 时写入点限制**：建议单次 bulk 条数在1000 – 5000之间，物理大小在1MB – 15MB大小之间。

系统默认规则

- **写入数据时新增字段处理**
  如果您往 metric 表写入数据时有未定义的新增字段，CTSDB 默认会将新增字段存储为 tags。您也可通过修改 metric 的 options 选项的 default_type 字段来修改，修改 metric 请参考 更新 metric 。
  - 若新增字段为整型，则 CTSDB 将其存储为 long 类型。
  - 若新增字段为小数，则 CTSDB 将其存储为 float 类型。
  - 若新增字段为字符串，则 CTSDB 将其存储为 string 类型，其长度为 max_string_length，超过部分会被系统丢弃，max_string_length 的长度可自定义，默认为256个字符，修改请参考 更新 metric 。
- **日期与时间处理**：日期与时间在 CTSDB 中以 UTC 格式存储，因此，查询数据时涉及时间范围的查询请通过 time_zone 参数来指定时区，其格式为 ISO 8601 UTC 偏移（例如 +01:00 或 -08:00），具体时区需根据实例所处的地域来确定，一般国内地域是东八区。具体请参考 常用查询示例 。

连接实例

CTSDB 实例目前只提供 VPC 网络下的连接方式。您可以通过控制台连接实例，也可以通过 RESTful API 接口连接实例，通过 API 接口连接实例时需要提供 root 账号的密码，以确保安全性。
CURL 连接实例创建表的示例如下，其中 `${user:password}` 是实例的用户名和密码，`${vip}:${vport}` 是实例的 IP 和 Port，`${metric_name}` 是新建的表名称。

```
curl -u ${user:password} -H 'Content-Type:application/json' -X PUT ${vip}:${vport}/_metric/${metric_name}
-d'
{
  "tags": {
      "region": "string",
      "set":  "long",
      "host": "string"
  },
  "time": {
       "name": "timestamp",
       "format": "epoch_second"
  },
  "fields": {
```

```
        "cpu_usage":  "float"
    },
    "options": {
        "expire_day": 7,
        "refresh_interval": "10s",
        "number_of_shards": 5,
        "number_of_replicas": 1,
        "rolling_period": 1
    }
}'
```

新建 metric

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

- 路径：`/_metric/${metric_name}`，`${metric_name}` 为新建的 metric 的名称。
- 方法：PUT

> **说明**
> `metric` 命名限制请参见 系统限制 。

请求参数

无

请求内容

参数名称	必选	类型	描述
tags	是	map	维度列，用于唯一标识数据，须至少包含一个维度，支持的数据类型：text（带有分词、全文索引的字符串）、string（不分词的字符串）、long、integer、short、byte、double、float、date、boolean。格式如 `{"region": "string","set": "long","host": "string"}`
time	是	map	时间列相关配置，用于存储数据入库的唯一时间，例如 `{"name": "timestamp", "format": "epoch_second"}`，填写时，需填写完整，name和format的值不能为空
fields	否	map	数据列，用于存储数据，为了节省空间，建议使用最适合实际业务使用的类型，支持的数据类型：string（字符串）、long、integer、short、byte、double、float、date、boolean。 例如 `{"cpu_usage":"float"}`
options	否	map	常用的调优配置信息，例如 `{"expire_day":7,"refresh_interval":"10s","number_of_shards":5,"number_of_replicas":1,"rolling_period":1,"max_string_length":256,"default_date_format":"strict_date_optional_time","indexed_fields":["host"]}`

- time 字段的 name 默认为 timestamp，时间格式（format）完全兼容 Elasticsearch 的时间格式，如 epoch_millis（以毫秒为单位的 Unix 时间戳）、epoch_second（以秒为单位的 Unix 时间戳）、basic_date（格式如 yyyyMMdd）、basic_date_time（格式如 yyyyMMdd'T'HHmmss.SSSZ）等。
- options 选项及解释如下：
  - expire_day：数据过期时间（单位：天），取值范围为非零整数，过期后数据自动清理，缺省情况下为最小值-1（代表永不过期）。
  - refresh_interval：数据刷新频率，写入的数据从内存刷新到磁盘后可查询。默认为10秒。
  - number_of_shards：表分片数，取值范围为正整数，小表可以忽略，大表按照一个分片至多25G设置分片数，默认为3。
  - number_of_replicas：副本数，取值范围须设置为正整数，例如一主一副为1，缺省为1；不可设置为0，否则数据没有副本，存在丢失风险。
  - rolling_period
    ：子表时长（单位：天），取值范围为非零整数，CTSDB 存储数据时，为了方便做数据过期和提高查询效率，根据特定时间间隔划分**子表**
    ，缺省情况下由数据过期时间决定，下面具体说明缺省子表时长和过期时间的关系。
  - max_string_length：自定义字符串 string 类型的值最大可支持的长度，取值范围为正整数，最大为32765，默认为256。
  - default_date_format：自定义维度列和指标列 date类型的格式，默认为 strict_date_optional_time 或 epoch_millis。
  - indexed_fields：指定指标列中需要保留索引的字段，可指定多个，以数组形式指定。
  - default_type：指定新增字段的默认类型。可选项为 tag、field，系统默认值为 tag。

过期时间	子表时长
≤ 7天	1天
> 7天，≤ 20天	3天
> 20天，≤ 49天	7天
> 49天，≤ 3个月	15天
> 3个月	30天
永不过期	30天

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT 172.xx.xx.4:9201/_metric/ctsdb_test -d'
{
    "tags":
    {
        "region":"string"
    },
    "time":
    {
        "name":"timestamp",
        "format":"epoch_second"
    },
    "fields":
    {
        "cpuUsage":"float"
    },
    "options":
    {
        "expire_day":7,
        "refresh_interval":"10s",
        "number_of_shards":5
    }
   }
'
```

成功的返回：

```
{
    "acknowledged": true,
    "message": "create ctsdb metric ctsdb_test success!"
   }
```

失败的返回：

```
  {
    "acknowledged": true,
    "message": "table ctsdb_test already exist"
   }
```

查询 metric

最近更新时间：2024-07-30 16:35:13

获取所有 metric

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metrics`
方法：GET

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_metrics
```

返回：

```
{
  "result":
{
   "metrics":
[
    "ctsdb_test",
    "ctsdb_test1"
   ]
  },
  "status": 200
}
```

获取特定 metric

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}`，`${metric_name}`为 metric 的名称。
方法：GET

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_metric/ctsdb_test
```

返回：

```
{
   "result":
{
    "ctsdb_test":
    {
        "tags":
        {
            "region": "string"
        },
        "time":
        {
            "name": "timestamp",
            "format": "epoch_second"
        },
        "fields":
        {
            "cpuUsage": "float"
        },
        "options":
        {
            "expire_day": 7,
            "refresh_interval": "10s",
            "number_of_shards": 5
        }
    }
   },
   "status": 200
}
```

更新 metric

最近更新时间：2024-07-30 16:35:13

请求地址和方法

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求参数

无

请求内容

字段 tags、time、fields、options 均为 map 类型的，选填，格式请参考 API 新建 metric 。具体要求如下：

- tags：允许新增维度字段和修改已存在的维度字段类型，不会删除原有字段。
- time：name 不能修改，format 允许修改。
- fields：允许新增指标字段和修改已存在的指标字段类型，不会删除原有字段。
- options 属性如下：

属性名称	必选	类型	描述
expire_day	否	integer	数据过期时间，取值范围为非零整数，过期后数据自动清理，缺省情况下永不过期
refresh_interval	否	string	数据刷新频率，写入的数据从内存刷新到磁盘后可查询，默认为10秒
number_of_shards	否	integer	表分片数，取值范围为正整数，小表可忽略，大表按照一个分片至多25G设置分片数，默认为3
number_of_replicas	否	integer	副本数，取值范围须设置为正整数，例如一主一副为1，缺省为1；不可设置为0，否则数据没有副本，存在丢失风险。
rolling_period	否	integer	子表时长（单位：天），取值范围为非零整数 CTSDB 存储数据时，为了方便做数据过期和提高查询效率，根据特定时间间隔划分子表，缺省情况下由数据过期时间决定
max_string_length	否	integer	自定义字符串 string 类型的值最大可支持的长度，取值范围为正整数，最大为32765，默认为256
default_date_format	否	string	自定义维度列和指标列 date 类型的格式，默认为 strict_date_optional_time 或 epoch_millis
indexed_fields	否	array	指定指标列中需要保留索引的字段，可指定多个，以数组形式指定
default_type	否	string	指定新增字段的默认类型。可选项为 tag、field，系统默认值为 tag

说明：

- 由于历史数据不可被修改，更新字段后 metric 信息不会立即变更，需要等待下一个 子表 产生。如果需要确认更新操作是否成功，可通过 `GET /_metric/${metric_name}?v` 接口进行确认。
- 只有为 short、integer、float 的字段才允许修改类型。其中 short 类型可被修改为 integer 和 long 类型；integer 类型可被修改为 long 类型；float 类型可被修改为 double 类型。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT 172.xx.xx.4:9201/_metric/ctsdb_test/update
-d'
```

```
{
    "tags":{
        "set":"string"
    },
    "time":{
        "name":"timestamp",
        "format":"epoch_second"
    },
    "fields":{
        "diskUsage":"float"
    },
    "options":{
        "expire_day":15,
        "number_of_shards":10
    }
   }'
```

返回：

```
{
   "acknowledged": true,
   "message": "update ctsdb metric test111 success!"
}
```

删除 metric

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}`，`${metric_name}` 为需要删除的 metric 的名称。
方法：DELETE

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
DELETE /_metric/ctsdb_test1
```

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.xx.xx.4:9201/_metric/ctsdb_test1
```

返回：

```
{
   "acknowledged": true,
   "message": "delete metric ctsdb_test1 success!"
}
```

删除 metric 字段

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}/delete`，`${metric_name}` 为 metric 的名称。
方法：PUT

请求参数

无

请求内容

参数名称	必选	类型	描述
tags	否	Array	枚举需要删除的维度字段，如 `"tags": ["ip"]`
fields	否	Array	枚举需要删除的数据字段，如 `"fields": ["diskUsage"]`

说明
由于历史数据不可被修改，删除字段后 metric 信息不会立即变更，需要等待下一个 子表 产生。如果需要确认删除操作是否成功，可通过 `GET /_metric/${metric_name}?v` 接口进行确认。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.xx.xx.4:9201/_metric/ctsdb_test1/delete -d'
{
"tags": ["ip"],
"fields": ["cpu"]
}'
```

返回：

```
{
   "acknowledged": true,
   "message": "update ctsdb_test1 metric success!"
}
```

查询数据

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`${metric_name}/_search`，`${metric_name}`为 metric 的名称。
方法：GET

请求参数

可设置写入数据时的 `_routing` 参数值来提高查询效率，具体请参见示例。

请求内容

查询主要有普通查询和聚合查询，查询请求完全兼容 Elasticsearch API，具体请求内容请参照示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

具体查询示例请参考 常用查询示例 。

批量查询数据

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_msearch`
方法：GET

请求参数

可通过设置写入数据时的 `_routing` 参数值来提高查询效率，具体请参照示例。也可通过设置 filter_path 参数来过滤并简化返回结果，其参数值为按照逗号分隔的多个 json 路径，其中单个 json 路径用点号连接，可以使用 `*` 通配符来适配任意字符， `**` 通配符可以用来匹配任意路径，可以使用 `-` 前缀来去除某些返回字段。
例如， `responses._shard.f*` 表示 `responses` 下 `_shards` 中的以 f 开头的字段； `responses.**._score` 表示 `responses` 中所有 `_score` 的字段。

请求内容

查询主要有普通查询和聚合查询，查询请求完全兼容 Elasticsearch API，具体请求内容请参照示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。这里列举几个查询通用返回字段，详情见下表，若需要简化查询返回结果，可设置 filter_path 参数。

字段名称	描述
hits	返回匹配的查询结果，其中 total 字段表示匹配到的数据条数。里面的 hits 字段为数组结构，若无指定，则包含所有查询结果的前十条。hits 数组里的每个结果中包含 _index（查询涉及到的 子表），若查询中指定了 docvalue_fields，则会返回 fields 字段具体指明每个字段的值。
took	整个查询耗费的毫秒数。
_shards	参与查询的分片数。其中 total 标识总分片数，successful 标识执行成功的数量，failed 标识执行失败的数量，skipped 标识跳过执行的分片数。
timed_out	查询是否超时，取值有 false 和 true。
status	查询返回的状态码，2XX 代表成功。

CURL 示例说明

无 filter_path 参数的请求

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_msearch -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

返回：

```
{
    "responses":
```

```
    [
        {
          "took": 0,
          "timed_out": false,
          "_shards": {
            "total": 0,
            "successful": 0,
            "skipped": 0,
            "failed": 0
          },
          "hits": {
            "total": 0,
            "max_score": 0,
            "hits": []
          },
          "status": 200
        },
        {
          "took": 1,
          "timed_out": false,
          "_shards": {
            "total": 3,
            "successful": 3,
            "skipped": 0,
            "failed": 0
          },
          "hits": {
            "total": 1,
            "max_score": 1,
            "hits": [
              {
                "_index": "ctsdb_test2017@-979200000_30",
                "_type": "doc",
                "_id": "AV_8fBhlUAkC9PF9L-2t",
                "_score": 1
              }
            ]
          },
          "status": 200
        }
      ]
}
```


有 filter_path 参数的请求

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_msearch?
filter_path=responses.\*\*.hits,responses._shards.f\*,-responses.\*\*._score -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

返回：


```
{
     "responses": [
   {
```

```
      "_shards": {
        "failed": 0
      }
    },
    {
      "_shards": {
        "failed": 0
      },
      "hits": {
        "hits": [
          {
            "_index": "ctsdb_test2017@-979200000_30",
            "_type": "doc",
            "_id": "AV_8fBhlUAkC9PF9L-2t"
          }
        ]
      }
    }
      ]
}
```

说明

如果 filter_path 参数同时设置了匹配条件和去除条件，则返回结果先执行去除条件再执行匹配条件。

常用查询示例

最近更新时间：2024-07-30 16:35:13

本文主要介绍 CTSDB 中常用查询及关键字的使用，并给出简单 CURL 示例。所有示例使用的 metric 结构如下所示：

```
{
    "ctsdb_test" : {
      "tags" : {
        "region" : "string"
      },
      "time" : {
        "name" : "timestamp",
        "format" : "epoch_millis"
      },
      "fields" : {
        "cpuUsage" : "float",
        "diskUsage" : "string",
        "dcpuUsage" : "integer"
      },
      "options" : {
        "expire_day" : 7,
        "refresh_interval" : "10s",
        "number_of_shards" : 5,
        "indexed_fields" : "cpuUsage"
      }
    }
}
```

组合查询

这里的组合查询即可以用于单查询也可用于复合查询。查询体中的 query 关键字通过 Query DSL（Domain Specific Language）来定义查询条件。下文会先介绍如何构造过滤条件，再介绍怎样组合过滤条件和如何处理返回结果集。

常用过滤条件

1. Range

Range 即区间查询，Range 查询支持的字段类型包括 string、long、integer、short、double、float、date。Range 查询可包含的参数如下表所示：

参数名称	描述
gte	大于或等于
gt	大于
lte	小于或等于
lt	小于

> **说明**
> 当 Range 查询涉及时间类型时，可通过 format 参数来指定时间格式。具体时间格式请参考 新建 metric 。

时间范围查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "range": {
            "timestamp": {
```

```
                    "gte": "01/01/2018",
                    "lte": "03/01/2018",
                    "format": "MM/dd/yyyy",
                    "time_zone":"+08:00"
                }
            }
        }
    }'
```

数字范围查询 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "range": {
            "cpuUsage": {
                "gte": 1.0,
                "lte": 10.0
            }
        }
    }
}'
```

2. Terms

Terms 关键字用于查询时匹配特定字段，其值需要用中括号包裹。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "terms": {
               "region": ["sh", "bj"]
           }
       }
   }'
```

过滤条件组合

复合查询常使用 Bool 关键字来组合多个查询条件，常见的用于 Bool 查询的组合关键字有 filter（类似于 AND）、must_not（类似于 NOT）、should（类似于 OR）。为提高查询性能，请务必加上 time 字段的 range 查询，且 time 字段，不论查询时以何种方式写入，返回值统一为 epoch_millis 格式。query-bool-filter 的组合形式可显著提升查询性能，请务必采用这种查询方式。

1. AND 条件 CURL 示例说明

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
     "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-03-06 23:05:00",
              "time_zone":"+08:00"
            }
```

```
          }
        },
        {
          "terms": {
            "region": ["sh"]
          }
        },
        {
          "terms": {
            "cpuUsage": ["2.0"]
          }
        }
      ]
    }
  },
     "docvalue_fields": [
    "cpuUsage",
    "timestamp"
       ]
}'
```

说明

此查询条件类似于

`timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-03-06 23:05:00' AND region=sh AND cpuUsage=2.0` 。

2. OR 条件 CURL 示例说明

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
     "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-11-06 23:05:00",
              "time_zone":"+08:00"
            }
          }
        },
        {
          "term": {
            "region": "gz"
          }
        }
      ],
      "should": [
        {
          "terms": {
            "cpuUsage": ["2.0"]
          }
        },
        {
          "terms": {
            "cpuUsage": ["2.5"]
          }
        }
```

```
      ],
      "minimum_should_match": 1
    }
     },
     "docvalue_fields": [
    "cpuUsage",
    "timestamp"
     ]
}'
```

说明

此查询条件类似于

`timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-11-06 23:05:00' AND region='gz' AND (cpuUsage=2.0 or cpuUsage=2.5)`

。minimum_should_match 参数的意义在于设置 cpuUsage=2.0 和 cpuUsage=2.5 至少匹配的个数，系统默认 minimum_should_match 为0。

3. NOT 条件 CURL 示例说明

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
     "query": {
    "bool": {
      "filter": [
                {
          "range": {
            "timestamp": {
              "format": "yyyy-MM-dd HH:mm:ss",
              "gte": "2017-11-06 23:00:00",
              "lt": "2018-11-06 23:05:00",
              "time_zone":"+08:00"
            }

          }
        },
        {
          "terms": {
            "region": ["gz"]
          }
        }
      ],
      "must_not": [
       {
          "terms": {
            "cpuUsage": ["2.0"]
          }
        }
      ]
    }
     },
     "docvalue_fields": [
    "cpuUsage",
    "timestamp"
        ]
     }'
```

说明

> 此查询条件类似于
> `timestamp>=2017-11-06 23:00:00 AND timestamp<2018-11-06 23:05:00 AND region='gz' AND cpuUsage !=2.0` 。

返回结果集处理

1. From/Size

通过设置 From 和 Size 关键字可对查询结果进行分页。From 关键字定义了查询结果中第一条数据的偏移量，Size 关键字配置返回结果的最大条数。From 系统默认为 0，Size 系统默认为 10，From 与 Size 的总和系统默认不能超过 65536。若想要更大的返回结果，请参考本文的 scroll 关键字查询。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "from": 0,
    "size": 5,
    "query": {
        "bool": {
            "filter": {
                "range": {
                    "timestamp": {
                        "gte": "01/01/2018",
                        "lte": "03/01/2018",
                        "format": "dd/MM/yyyy",
                        "time_zone":"+08:00"
                    }
                }
            },
            "must_not": {
                "terms": {
                    "region": ["bj"]
                }
            }
        }
    }
}'
```

2. Scroll

Scroll 可以理解为关系型数据库里的 cursor，因此 scroll 并不适合用来做实时搜索，而更适用于后台批处理任务。
可以把 scroll 分为初始化和遍历两个阶段，初始化时将所有符合搜索条件的搜索结果缓存起来，类似于快照，在遍历时，从这个快照里取数据。注意，在 Scroll 初始化后对 metric 的插入、删除、更新都不会影响遍历结果。
初始化阶段可通过 size 关键字指定返回结果集的大小，size 默认值为10，最大值为65536；初始化阶段可通过 query 关键字指定查询条件；初始化阶段可通过 docvalue_fields 关键字指定返回字段。初始化阶段和遍历阶段都返回 `_scroll_id` 字段，后续遍历可以通过指定前一次遍历返回的 `_scroll_id` 来进行新的遍历，直到返回结果为空；初始化和遍历两个阶段都可以通过指定 scroll 参数来设定遍历的上下文环境保留时间，过期后 `_scroll_id` 将变得无效。时间格式如下表所示：

格式	描述
d	days
h	hours
m	minutes
s	seconds
ms	milliseconds
micros	microseconds
nanos	nanoseconds

CURL 示例说明：

scroll 初始化：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST 172.xx.xx.4:9201/ctsdb_test/_search?
scroll=1m -d'
   {
      "size":5,
      "query": {
      "bool": {
        "filter": [
          {
            "terms": {
              "region": ["gz"]
            }
          }
        ]
      }
    },
   "docvalue_fields": [
         "cpuUsage",
       "region",
       "timestamp"
       ]
}'
```

scroll 初始化返回：

```
   {
     "_scroll_id":
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRkZTcGp4dFRXR18xMGtzSmhEUFJRA
AAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw==",
     "took": 10641,
     "timed_out": false,
     "_shards": {
      "total": 3,
      "successful": 3,
      "skipped": 0,
      "failed": 0
     },
     "hits": {
       "total": {
            "value":1592072666,
            "relation":"eq"
        },
       "max_score": 0.65708643,
       "hits": [
         {
           "_index": "ctsdb_test@0_-1",
           "_type": "doc",
           "_id": "oyylNU0U65cZjByyt7sW_JmPPgAACY4Bh0",
           "_score": 0.65708643,
           "_routing": "354d14eb",
           "fields": {
             "region": [
               "gz"
             ],
             "cpuUsage": [
               "2.0"
             ],
             "timestamp": [
```

```
            1509909300000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
        "_id": "oyykFN0yd1d9NDPfzjRdrJ8whQAACYsBqc",
        "_score": 0.65708643,
        "_routing": "14dd3277",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "1.8"
          ],
          "timestamp": [
            1509908340000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
        "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABHIBso",
        "_score": 0.65708643,
        "_routing": "1cba7d8e",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "2.5"
          ],
          "timestamp": [
            1509909720000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
        "_id": "oyylH2JsOKnFHGUinQ7jM-ZwkgAAAvcBso",
        "_score": 0.65708643,
        "_routing": "1f626c38",
        "fields": {
          "region": [
            "gz"
          ],
          "cpuUsage": [
            "2.1"
          ],
          "timestamp": [
            1509909720000
          ]
        }
      },
      {
        "_index": "ctsdb_test@0_-1",
        "_type": "doc",
```

```
            "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABGsBso",
            "_score": 0.65708643,
            "_routing": "1cba7d8e",
            "fields": {
              "region": [
                "gz"
              ],
              "cpuUsage": [
                "2.0"
              ],
              "timestamp": [
                1509909720000
              ]
            }
          }
        ]
          }
    }
```

scroll 遍历：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X POST 172.xx.xx.4:9201/_search/scroll  -d'
    {
    "scroll" : "1m",
    "scroll_id" :
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRkZTcGp4dFRXR18xMGtzSmhEUFJRA
AAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw=="
    }'
```

> **说明**
>
> - 此请求中的 scroll_id 是 scroll 初始化返回的 `_scroll_id` 值。而下一次遍历则需要将 scroll_id 参数调整为上一次遍历返回的 `_scroll_id` 值，即每次请求中的 scroll_id 参数是上一次请求返回的 `_scroll_id` 值，直到返回结果为空，遍历结束。
> - 两次遍历返回的 `_scroll_id` 值可能相同，无法利用 `_scroll_id` 进行指定页跳转。

3. Sort

Sort 关键字主要用于对查询结果进行排序。排序方式有 asc 和 desc 两种，CTSDB 对于用户自定义字段的默认排序方式为 asc。排序模式有 min、max、sum、avg、median。其中 sum、avg、median 只适用于类型为 array 并存储数字的字段。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
    {
        "query": {
            "bool": {
                "must": {
                    "range": {
                        "timestamp": {
                            "gte": "01/01/2018",
                            "lte": "03/01/2018",
                            "format": "MM/dd/yyyy",
                            "time_zone":"+08:00"
                        }
                    }
                }
            }
        },
        "sort": [
            {
```

```
            "cpuUsage": {
                "order": "asc",
                "mode": "min"
            }
        },
        {
            "timestamp": {
                "order": "asc"
            }
        },
        "diskUsage"
    ]
}'
```

4. docvalue_fields

docvalue_fields 关键词指定需要返回的字段名称，需要以数组的形式指定。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "terms": {
            "region": ["sh", "bj"]
        }
    },
    "docvalue_fields": ["timestamp", "cpuUsage"]
}'
```

聚合查询

agg 关键字主要用于构造聚合查询。用户可到返回的 aggregations 字段取聚合结果。聚合返回字段说明详见下表。如果只关注聚合结果，请在查询时设置 size 参数为0。

字段名称	描述
hits	返回匹配的查询结果，其中 total 字段表示参与聚合的数据条数。里面的 hits 字段为数组结构，若无指定，则包含所有查询结果的前十条。hits 数组里的每个结果中包含 `_index`（查询涉及到的 子表），若查询中指定了 docvalue_fields，则会返回 fields字段具体指明每个字段的值。
took	整个查询耗费的毫秒数。
_shards	参与查询的分片数。其中 total 标识总分片数，successful 标识执行成功的数量，failed 标识执行失败的数量，skipped 标识跳过执行的分片数。
timed_out	查询是否超时，取值有 false 和 true。
aggregations	聚合返回结果。

下面列举几种常用的聚合方式。

普通聚合

普通聚合需要指定聚合名称、聚合方式（常用的聚合方式有 min、max、avg、value_count、sum 等）和聚合所作用的字段。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "size":0,
    "query": {
        "terms": {
```

```
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "myname": {
                "max": {
                    "field": "cpuUsage"
                }
            }
        }
    }'
```

说明
上例中对字段 cpuUsage 做 max 聚合（用户也可指定 min、avg 等），返回结果取别名 myname（用户可任意指定该名称）。

返回结果：

```
{
 "took": 1,
 "timed_out": false,
 "_shards": {
   "total": 20,
   "successful": 20,
   "skipped": 0,
   "failed": 0
 },
 "hits": {
   "total": {
             "value":7,
             "relation":"eq"
         },
   "max_score": 0,
   "hits": []
 },
 "aggregations": {
   "myname": {
     "value": 4
   }
 }
}
```

terms 聚合

terms 聚合主要用于查询某个字段的所有唯一值以及该值的个数，用户可指定唯一值返回时的排序规则，以及返回结果数，并且可对参与聚合的数据字段进行模糊或者精确匹配，详情请参考如下示例，使用 filter_path 参数可定制返回结果字段，使用方法请参考 批量查询数据 。
CURL 示例说明：
请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
{
    "aggs": {
        "myname": {
            "terms": {"field":"region"}
        }
    }
}'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

说明

上例表示查看 ctsdb_test 的 region 字段中所有的唯一值及其出现的次数。分析返回字段 aggregations 中的 buckets 字段发现，region 字段共有7种类型的值，分别是 sh、Motor_sports、gz、bj、cd、Winter_sports、water_sports， 并且返回字段通过 doc_count 指明了每种值的个数。

用户可通过 size 字段指定返回的唯一值的个数，例如某字段名为 region，其唯一值有7个，可通过设置 size 字段为5指定只返回前5个，具体请参考示例。

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
{
    "aggs": {
        "myname": {
            "terms": {
               "field":"region",
                "size":5
            }
        }
```

```
    }
}'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 2,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        }
      ]
    }
  }
}
```

用户可对返回结果进行排序，详情请参考如下示例。

1.请求（返回结果按照唯一值的个数降序排列）：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
   {
     "aggs": {
         "myname": {
             "terms": {
               "field":"region",
               "order":{"_count":"desc"}
             }
         }
     }
   }'
```

返回：

```
  {
     "aggregations": {
       "myname": {
         "doc_count_error_upper_bound": 0,
         "sum_other_doc_count": 0,
```

```
        "buckets": [
          {
            "key": "sh",
            "doc_count": 10
          },
          {
            "key": "Motor_sports",
            "doc_count": 6
          },
          {
            "key": "gz",
            "doc_count": 3
          },
          {
            "key": "bj",
            "doc_count": 2
          },
          {
            "key": "cd",
            "doc_count": 2
          },
          {
            "key": "Winter_sports",
            "doc_count": 1
          },
          {
            "key": "water_sports",
            "doc_count": 1
          }
        ]
      }
    }
  }
```

2.请求（返回结果按照唯一值的字母升序进行排列）：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
  {
    "aggs": {
        "myname": {
            "terms": {
              "field":"region",
              "order":{"_term":"asc"}
            }
        }
    }
  }'
```

返回：

```
  {
    "aggregations": {
      "myname": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": [
          {
            "key": "Motor_sports",
```

```
            "doc_count": 6
          },
          {
            "key": "Winter_sports",
            "doc_count": 1
          },
          {
            "key": "bj",
            "doc_count": 2
          },
          {
            "key": "cd",
            "doc_count": 2
          },
          {
            "key": "gz",
            "doc_count": 3
          },
          {
            "key": "sh",
            "doc_count": 10
          },
          {
            "key": "water_sports",
            "doc_count": 1
          }
        ]
      }
    }
  }
```

用户可设置正则表达式对参与 terms 聚合的数据字段进行模糊匹配或者指定字段进行精确匹配，详情请参考示例。
模糊匹配示例：
请求（只针对 region 字段中有 sport 并且不是以 `water_` 开头的数据进行聚合并返回结果）：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
{
 "aggs": {
     "myname": {
         "terms": {
           "field":"region",
           "include" : ".*sport.*",
             "exclude" : "water_.*"
         }
     }
 }
}'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Motor_sports",
```

```
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

精确匹配示例

请求（region_zone 指定值为"sh"，"bj"，"cd"，"gz"的 region 字段进行聚合，而 region_sports 指定值不为"sh"，"bj"，"cd"，"gz"的 region 字段进行聚合）

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search?
filter_path=aggregations -d'
{
"aggs" : {
 "region_zone" : {
   "terms" : {
     "field" : "region",
     "include" : ["sh", "bj","cd","gz"]
     }
   },
   "region_sports" : {
    "terms" : {
        "field" : "region",
        "exclude" : ["sh", "bj","cd","gz"]
    }
   }
 }
}'
```

返回：

```
{
  "aggregations": {
    "region_sport": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    },
    "region_zone": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
```

```
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        }
      ]
    }
  }
}
```

Date Histogram 聚合

Date Histogram 主要对日期做直方图聚合。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
   {
       "query": {
           "terms": {
               "region": ["sh", "bj"]
           }
       },
       "aggs": {
           "time_1h_agg": {
               "date_histogram": {
                   "field": "timestamp",
                   "interval": "1h"
               },
               "aggs": {
                   "avgCpuUsage": {
                       "avg": {
                           "field": "cpuUsage"
                       }
                   }
               }
           }
       }
   }'
```

返回：

```
   {
     "took": 5,
     "timed_out": false,
     "_shards": {
       "total": 20,
       "successful": 20,
       "skipped": 0,
```

```
    "failed": 0
  },
  "hits": {
    "total": {
         "value":6,
         "relation":"eq"
     },
    "max_score": 0.074107975,
    "hits": [
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf-",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf_",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgA",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgB",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgC",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgD",
        "_score": 0.074107975,
        "_routing": "sh"
      }
    ]
  },
  "aggregations": {
    "time_1h_agg": {
      "buckets": [
        {
          "key_as_string": "1520222400",
          "key": 1520222400000,
          "doc_count": 1,
          "avgCpuUsage": {
```

```
            "value": 2.5
          }
        },
        {
          "key_as_string": "1520226000",
          "key": 1520226000000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520229600",
          "key": 1520229600000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520233200",
          "key": 1520233200000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520236800",
          "key": 1520236800000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520240400",
          "key": 1520240400000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520244000",
          "key": 1520244000000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520247600",
          "key": 1520247600000,
          "doc_count": 1,
          "avgCpuUsage": {
            "value": 2
          }
        },
        {
          "key_as_string": "1520251200",
```

```
            "key": 1520251200000,
            "doc_count": 4,
            "avgCpuUsage": {
              "value": 2.25
            }
          }
        ]
      }
    }
  }
```

说明
上例是对字段 cpuUsage 进行粒度为1小时的 date_histogram 聚合。返回结果中总的聚合名称为 time_1h_agg（用户可任意指定该名称），每个时间分段内的聚合名称为 avgCpuUsage（用户可任意指定该名称）。interval 字段可选的时间粒度有 year、quarter、 month、week、day、hour、minute、second，用户也可将时间粒度用时间单位来表示，例如1y代表1year，1h代表1hour。系统不支持小数粒度的时间单位，例如1.5h您需要转换为90min。

Percentiles 聚合

Percentiles 聚合即百分位聚合。百分位可以任意指定，系统默认的百分位是 1、5、25、50、75、95、99，可自行选择其他数值。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "terms": {
            "region": ["sh", "bj"]
        }
    },
    "aggs":
    {
        "myname":
        {
            "percentiles":
            {
                "field": "cpuUsage",
                "percents": [1,25,50,70,99]
            }
        }
    }
}'
```

返回：

```
  {
    "took": 18,
    "timed_out": false,
    "_shards": {
      "total": 20,
      "successful": 20,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": {
           "value":6,
           "relation":"eq"
       },
```

```
    "max_score": 0.074107975,
    "hits": [
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf-",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf_",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgA",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgB",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgC",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgD",
        "_score": 0.074107975,
        "_routing": "sh"
      }
    ]
  },
  "aggregations": {
    "myname": {
      "values": {
        "1.0": 2,
        "25.0": 2,
        "50.0": 2.25,
        "70.0": 2.5,
        "99.0": 2.5
      }
    }
  }
}
```


说明

上例中是对字段 cpuUsage 做 percentiles 聚合，选取的百分位为 1,25,50,70,99。聚合返回结果的别名是 myname（用户可任意指定该名称）。

Cardinality 聚合

Cardinality 聚合主要用于统计去重后的数量。默认情况下，当聚合结果小于等于3000时，Cardinality 返回的结果是精确值，大于3000时，为近似值。
CURL 示例说明：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/ctsdb_test/_search -d'
{
    "query": {
        "terms": {
            "region": ["sh", "bj"]
        }
    },
    "aggs":
    {
        "myname":
        {
            "cardinality":
            {
                "field": "cpuUsage"
            }
        }
    }
}'
```

返回：

```
  {
    "took": 15,
    "timed_out": false,
    "_shards": {
      "total": 20,
      "successful": 20,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": {
           "value":6,
           "relation":"eq"
       },
      "max_score": 0.074107975,
      "hits": [
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf-",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf_",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
```

```
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgA",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgB",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgC",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgD",
        "_score": 0.074107975,
        "_routing": "sh"
      }
    ]
  },
  "aggregations": {
    "myname": {
      "value": 2
    }
  }
}
```


说明

上例的聚合是对字段 cpuUsage 做 cardinality 聚合，返回的聚合结果取别名 myname（用户可任意指定该名称）。

批量写入数据

最近更新时间：2024-07-30 16:35:13

往单个 metric 批量写入数据

此接口同时适用于批量和非批量写入数据至单个 metric 的场景，为提高写入性能，建议批量写入数据。

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`/${metric_name}/_doc/_bulk`，${metric_name}为 metric 的名称。
方法：POST

> **说明**
> `_doc` 关键字为写入数据的 `_type`，为了便于以后系统做解析和升级，请务必加上 `_doc` 关键字。

请求参数

可通过设置 filter_path 参数来过滤并简化返回结果，具体请参考 批量查询数据 。

请求内容

批量写入 metric 需要一种 NDJSON 格式的结构数据，类似于如下：

```
元数据\n
需要写入的数据\n
....
元数据\n
需要写入的数据\n
```

元数据的格式如下所示：

```
{
"index" :
{
"_id" : "1",                #文档 ID(选填)
}
}
```

写入数据的格式类似于：

```
{
"field1" : "value1",
"field2" : "value2"
}
```

返回内容

需要注意的是，批量写入数据接口与其他接口返回结果稍有不同。请优先关注 json 结果的 errors（注意不是 error）字段，如果为 false，代表所有数据写入成功；如果为 true，代表有部分写入失败，具体失败的详情可以通过 items 字段得到。
items 字段为一个数组，数组中每一个元素与写入请求一一对应，可通过每个元素是否有 error 字段判断请求是否成功，若有 error 字段则表示请求失败，具体错误内容在 error 字段内，若无 error 字段表示请求成功。

CURL 示例说明

返回成功的示例

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST 172.xx.xx.4:9201/ctsdb_test/_doc/_bulk -
d'
 {"index":{"routing": "sh" }}
 {"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
 {"index":{"routing": "sh" }}
 {"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

返回：

```
{
   "took": 134,
   "errors": false,
   "items":
[
    {
        "index":
        {
            "_index": "ctsdb_test@1505232000000_1",
            "_type": "_doc",
            "_id": "AV_8eeo_UAkC9PF9L-2q",
            "_version": 1,
            "result": "created",
            "_shards":
             {
                "total": 2,
                "successful": 2,
                "failed": 0
            },
            "created": true,
            "status": 201
            }
        },
        {
        "index":
        {
            "_index": "ctsdb_test2@1505232000000_1",
            "_type": "_doc",
            "_id": "AV_8eeo_UAkC9PF9L-2r",
            "_version": 1,
            "result": "created",
            "_shards":
            {
                "total": 2,
                "successful": 2,
                "failed": 0
            },
            "created": true,
            "status": 201
        }
    }
   ]
}
```

说明

上面返回中的 errors 为 false，代表所有数据写入成功。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，`_index` 标识了写入的 metric 子表，`_shards` 记录副本写入情况，上例中 total 表示两副本，

successful 代表两副本均写入成功。

返回失败的示例

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/hcbs_client_trace/_doc/_bulk -d'
    {"index":{"_id":"5"}}
    {"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":2,"data_len":4096,"latency":3,"try_times":1,"errcode":0,"start_time":1503404266}
    {"index":{"_id":"6"}}
    {"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":"abc","data_len":4096,"latency":1,"try_times":1,"errcode":0,"start_time":150340426
6}
'
```

返回：

```
{
    "took":71,
    "errors":true,
    "items":[
        {
            "index":{
                "_index":"hcbs_client_trace@1505232000000_1",
                "_type":"_doc",
                "_id":"5",
                "_version":1,
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":2,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        },
        {
            "index":{
                "_index":"hcbs_client_trace@1505232000000_1",
                "_type":"_doc",
                "_id":"6",
                "status":400,
                "error":{
                    "type":"illegal_argument_exception",
                    "reason":"mapper [io_type] cannot be changed from type [long] to [keyword]"
                }
            }
        }
    ]
}
```

说明

> 上面返回中的 errors 为 true，代表部分数据写入失败。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，error 字段详细给出了错误内容，_index 标识了写入的 metric 子表，_shards 记录副本写入情况，上例中 total 表示两副本，successful 代表两副本均写入成功。

往多个 metric 批量写入数据

此接口同时适用于批量和非批量写入单个 metric 或者多个 metric 的场景。为提高写入性能，建议批量写入数据。

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`_bulk`
方法：PUT

请求参数

可通过设置 filter_path 参数来过滤并简化返回结果，具体请参考 批量查询数据 。

请求内容

批量写入 metric 需要一种 NDJSON 格式的结构数据，类似于如下：

```
元数据\n
需要写入的数据\n
....
元数据\n
需要写入的数据\n
```

元数据的格式如下所示：

```
{
"index" :
{
    "_index" :"metric_name",     #要写入的 metric
    "_id" : "1",                 #文档 ID(选填)
}
}
```

写入数据的格式类似于：

```
{
"field1" : "value1",
"field2" : "value2"
}
```

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误内容在 error 字段内。注意：若请求成功，但是 errors（注意不是 error）字段非等于 false，则该 errors 字段具体指出写入失败的具体数据。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT 172.xx.xx.4:9201/_bulk -d'
{"index":{"_index" : "ctsdb_test"}}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_index" : "ctsdb_test2"}}
```

```
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

返回：

```
{
    "took":494,
    "errors":false,
    "items":[
        {
            "index":{
                "_index":"ctsdb_test_ay@1505232000000_1",
                "_type":"_doc",
                "_id":"GgJiyXsBE2F__WP8B_ik",
                "_version":1,
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":1,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        },
        {
            "index":{
                "_index":"ctsdb_test2_ay@1505232000000_1",
                "_type":"_doc",
                "_id":"GwJiyXsBE2F__WP8B_ik",
                "_version":1,
                "result":"created",
                "_shards":{
                    "total":2,
                    "successful":1,
                    "failed":0
                },
                "_seq_no":0,
                "_primary_term":1,
                "status":201
            }
        }
    ]
}
```

说明

上面返回中的 errors 为 false，代表所有数据写入成功。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，`_index` 标识了写入的 metric 子表，`_shards` 记录副本写入情况，上例中 total 表示两副本，successful 代表两副本均写入成功。

删除数据

最近更新时间：2024-07-30 16:35:13

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`/${metric_name}/_delete_by_query`，`${metric_name}`为 metric 的名称。
方法：POST

请求参数

无

请求内容

删除 metric 时的查询条件，具体请参考示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误内容在 error 字段内。

CURL 示例说明

删除指定时间范围内的数据，请求示例：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/ctsdb_test/_delete_by_query -d'
{
  "query": {
    "range": {
      "timestamp": {
         "format": "yyyy-MM-dd HH:mm:ss",
          "gte": "2022-11-06 23:00:00",
          "lt": "2022-11-06 23:05:00",
          "time_zone":"+08:00"
      }
    }
  }
}'
```

> **说明：**
> 此删除条件类似于 `timestamp>='2022-11-06 23:00:00' AND timestamp<'2022-11-06 23:05:00'`。

返回：

```
{
  "took": 66,
  "timed_out": false,
  "total": 24,
  "deleted": 24,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
```

```
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "failures": [

  ]
}
```

说明：
通过以上删除示例的返回结果可知，总共待删除文档数24个，已经成功删除文档数24个，删除操作耗时66毫秒。

修改数据

最近更新时间：2024-07-30 16:35:13

CTSDB 修改数据可使用先删除数据再写入数据的方式。数据的删除和写入，请参见 删除数据 和 批量写入数据 。

Rollup 相关操作

最近更新时间：2024-07-30 16:35:13

建立 Rollup 任务

在海量数据场景下，业务系统每天甚至每小时会产生 PB 级别数据。而时序数据的最主要的特点就是海量性、时效性和趋势性，因此通常情况下，使用数据的系统（例如监控系统或数据分析系统）通常只需要最近时间段内的高精度数据，而历史数据只需降精度（Downsampling）保存即可。
用户可通过配置 Rollup 任务定时聚合历史数据保存至新的数据表。Rollup 任务不仅能降精度保存历史数据，也能提高查询性能，降低存储成本。需要注意的是，Rollup 任务会自动根据 base_metric 建立子表，继承父表的所有配置，如果指定 options，会覆盖父表配置。

1. 请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}`为 Rollup 任务的名称。
方法：PUT

3. 请求参数

无

4. 请求内容

参数名称	必选	类型	描述
base_metric	是	string	Rollup 依赖的 metric 名称（父表）
rollup_metric	是	string	Rollup 产生的 metric 名称（子表）
base_rollup	否	string	依赖的 Rollup 任务，任务执行前会检查相应时间段的依赖任务是否完成执行
query	否	string	过滤数据的查询条件，由很多个元素和操作对组成，例如：`name:host AND type:max OR region:gz`
group_by	是	Array	进行聚合的维度列，可以包含多列
function	是	Map	指定聚合的名称、方法和字段，其字段只能选自 base_metric 里的 fields 字段，如果 base_metric 的 fields 为空，则无法设置 rollup，function 有 sum、avg、min、max、set、any、first、last、percentiles 等。例如： `{"cost_total":{"sum": {"field":"cost"}},"cpu_usage_avg":{ "avg": { "field":"cpu_usage"}}}`
interval	是	string	聚合粒度（rollup 产生数据的时间精度），例如1s、5m（5分钟）、1h、1d等
frequency	否	string	调度频率，例如5m、1h、1d等，默认等于 interval
delay	否	string	延迟执行时间，写入数据通常有一定的延时，避免丢失数据，例如5m、1h等
start_time	否	string	开始时间，从该时间开始周期性执行 Rollup，默认为当前时间
end_time	否	string	结束时间，到达该时间后不再调度，默认为时间戳最大值
options	否	map	rollup_metric 选项，跟新建 metric 选项一致

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test -d'
{
   "base_metric": %{base_metric_name},
   "rollup_metric": %{rollup_metric_name},
   "base_rollup": %{base_rollup_name},
   "query" : "name:host AND type:max",
   "group_by": ["host"],
   "function": {
       "cost_total":  {
           "sum": {
               "field": "cost"
           }
       },
       "cpu_usage_avg": {
           "avg": {
               "field": "cpu_usage"
           }
       },
       "value": {
         "percentiles": {
           "field": "value",
           "percents": [
             95
           ]
         }
       },
       "metricName": {
         "set": {
           "value": "cpu_usage"
         }
       },
       "appid": {
         "any": {
           "field": "appid"
         }
       },
        "first_value": {
         "first": {
           "field": "value"
         }
       },
        "last_value": {
         "last": {
           "field": "value"
         }
       }
   },
   "interval": "1m",
   "frequency": "5m",
   "delay": "1m",
   "start_time": "1502892000",
   "end_time": "2147483647",
```

```
    "options": {
        "expire_day": 365
    }
}
'
```

返回：

```
{
    "acknowledged": true,
    "message": "create rollup success"
}
```

获取所有 Rollup 任务的名称

1. 请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollups`
方法：GET

3. 请求参数

无

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.4:9201/_rollups
```

返回：

```
{
    "result":
{
     "rollups":
     [
         "rollup_jgq_6",
         "rollup_jgq_60"
     ]
    },
    "status": 200
}
```

获取某个 Rollup 任务的详细信息

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}`为 Rollup 任务的名称。
方法：GET

3. 请求参数

指定 v 参数可以查看 rollup 的具体进度，返回结构中的 @last_end_time 为 rollup 最新进度。

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.xx.xx.44:9201/_rollup/rollup_jgq_6?v
```

返回：

```
{
 "result": {
   "rollup_jgq_6": {
   "base_metric": "cvm_device-300",
      "rollup_metric": "cvm_device-86400",
      "query": "metricName:cpu_usage AND statType:max",
      "group_by": [
        "vm_uuid"
      ],
      "function": {
        "value": {
          "percentiles": {
            "field": "value",
            "percents": [
              95
            ]
          }
        },
        "metricName": {
          "set": {
            "value": "cpu_usage"
          }
        },
        "appid": {
          "any": {
            "field": "appid"
          }
        }
      },
      "interval": "1d",
      "delay": "5m",
      "options": {
        "expire_day": 186
      },
      "frequency": "1d",
      "start_time": 1534003200,
      "end_time": 2147483647,
      "@state": "running",              // 运行状态
      "@timestamp": 1550766085000,      // rollup 任务信息更新的时间点
      "@last_end_time": 1550764800      // rollup 任务正确执行到的时间点
    }
```

```
  },
  "status": 200
}
```

删除 Rollup 任务

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}`，为 Rollup 任务的名称。
方法：DELETE

3. 请求参数

无

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test
```

返回：

```
{
   "acknowledged": true,
   "message": "delete rollup success"
}
```

更新 Rollup 任务

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}/update`，`${rollup_task_name}` 为 Rollup 任务的名称。
方法：POST

3. 请求参数

无

4. 请求内容

参数名称	必选	类型	描述
state	是	string	running/pause
start_time	否	string	开始时间，从该时间开始周期性执行 Rollup，默认为当前时间
end_time	否	string	结束时间，到达改时间后不再调度，默认为时间戳最大值
options	否	map	聚合选项，跟新建 metric 选项一致

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.xx.xx.4:9201/_rollup/ctsdb_rollup_task_test/update -d'
{
   "state":"running",
   "start_time": "1511918989",
   "end_time": "1512019765",
   "options":
{
       "expire_day": 365
   }
}'
```

返回：

```
{
   "acknowledged": true,
   "message": "update rollup success"
}
```

类 SQL 查询支持

最近更新时间：2024-07-30 16:35:13

CTSDB 2.0 支持使用类 SQL 语句进行查询，仅为了方便初学者使用；如关注查询时效，仍推荐使用 CTSDB 原生查询语句。

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：_nlpcn/sql
方法：GET

请求参数

可在查询时加入 pretty 参数值来获得整理格式后的返回响应，具体请参见示例。

请求内容

查询主要有普通查询和聚合查询，具体请求内容请参照示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。如参数名或表名的错误，需要自行检查更正。

CURL 示例说明

具体查询请参考下列示例。
所有示例使用的 metric 结构如下所示：

```
{
    "ctsdb_test" : {
      "tags" : {
        "region" : "string"
      },
      "time" : {
        "name" : "timestamp",
        "format" : "epoch_millis"
      },
      "fields" : {
        "cpuUsage" : "float",
        "diskUsage" : "string",
        "dcpuUsage" : "integer"
      },
      "options" : {
        "expire_day" : 7,
        "refresh_interval" : "10s",
        "number_of_shards" : 5
      }
    }
}
```

普通查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET 172.xx.xx.4:9201/_nlpcn/sql?pretty -d
'select docvalue(cpuUsage) from ctsdb_test limit 1'
```

普通查询的响应示例：

```
{
  "took" : 22,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "ctsdb_test@1646064000000_1",
        "_type" : "_doc",
        "_id" : "XMzBRX8BfqojiIOn8jFa",
        "_score" : 1.0,
        "fields" : {
          "cpuUsage" : [
            10.0
          ]
        }
      }
    ]
  }
}
```

说明

普通查询时，所查询字段，需要加 docvalue 使实例从其列存区拿取数据。

带搜索条件的普通查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET 172.xx.xx.4:9201/_nlpcn/sql?pretty -d
'select docvalue(cpuUsage) from ctsdb_test where region="shanghai" limit 1'
```

带搜索条件的普通查询的响应示例：

```
{
  "took" : 13,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.0,
    "hits" : [
      {
```

```
        "_index" : "ctsdb_test@1646064000000_1",
        "_type" : "_doc",
        "_id" : "XszBRX8BfqojiIOn8jFb",
        "_score" : 0.0,
        "fields" : {
          "cpuUsage" : [
            30.0
          ]
        }
      }
    ]
  }
}
```

说明

带搜索条件的普通查询时，用做查询条件的字段，不需要加docvalue。

按某 tag 字段分组聚合查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET 172.xx.xx.14:9201/_nlpcn/sql?pretty -d
'select max(cpuUsage) from ctsdb_test group by region'
```

按某 tag 字段分组聚合查询的响应示例：

```
{
  "took" : 33,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "region" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "beijing",
          "doc_count" : 1,
          "MAX(cpuUsage)" : {
            "value" : 20.0
          }
        },
        {
          "key" : "chengdu",
          "doc_count" : 1,
          "MAX(cpuUsage)" : {
            "value" : 10.0
          }
```

```
        },
        {
          "key" : "shanghai",
          "doc_count" : 1,
          "MAX(cpuUsage)" : {
            "value" : 30.0
          }
        }
      ]
    }
  }
}
```

说明
按某 tag 字段分组聚合查询时，用于分组的字段，不需要加 docvalue。

按时间聚合分组查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type: application/json' -XGET 172.xx.xx.4:9201/_nlpcn/sql?pretty -d
'select max(cpuUsage) from ctsdb_test GROUP BY date_histogram(field="timestamp","interval"="1d")'
```

按时间聚合分组查询的响应示例：

```
{
  "took" : 30,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "date_histogram(field=timestamp,interval=1d)" : {
      "buckets" : [
        {
          "key_as_string" : "2022-03-01 00:00:00",
          "key" : 1646092800000,
          "doc_count" : 3,
          "MAX(cpuUsage)" : {
            "value" : 30.0
          }
        }
      ]
    }
  }
}
```

CTSDB 1.0
简介

最近更新时间：2024-08-30 11:31:01

CTSDB 支持 HTTP 协议进行数据写入和查询等操作。提供的 HTTP API 是 RESTful API，对资源的请求方式是通过向资源对应的 URI 发送标准的 HTTP 请求，例如 GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源等。
用户通过 HTTP API 几乎可以实现所有的数据操作。CTSDB 通过提供 VPC 网络隔离和访问时提供用户名和密码的身份认证方式来保证数据的安全性，通过 JSON 格式的结构体进行数据交换，每个请求都会返回一个标准的 HTTP 响应状态码和响应内容。若操作失败，用户可以根据响应内容获取到具体错误信息。
通过 RESTful API 接口连接实例方式请参见 连接实例 。

命名规则

CTSDB 中 metric（表）、tags（维度列）、fields（数据列）的命名应简明扼要，tags、fields 详细介绍可参见 请求内容：

- **metric 命名规则**：允许使用小写英文字母、数字、_（下划线）、-（短划线）、.（圆点）的组合，且不能以 .（圆点）、_（下划线）或 -（短划线）开头。长度为1到200个字符。
- **metric 中 tags 和 fields 命名规则**：允许大小写英文字母、数字、_（下划线）、-（短划线）、.（圆点）的任意组合，且不能以 .（圆点）开头。长度为1到255个字符。

系统限制

- **metric 中 tags 和 fields 限制**：metric 中字段总数不超过1000。
- **批量写入 metric 时写入点限制**：建议单次 bulk 条数在1000 - 5000之间，物理大小在1MB - 15MB大小之间。

系统默认规则

- **写入数据时新增字段处理**：如果您往 metric 表写入数据时有未定义的新增字段，CTSDB 默认会将新增字段存储为 tags。您也可通过修改 metric 的 options 选项的 default_type 字段来修改，修改 metric 请参考 更新 metric 。
- 若新增字段为整型，则 CTSDB 将其存储为 long 类型。
- 若新增字段为小数，则 CTSDB 将其存储为 float 类型。
- 若新增字段为字符串，则 CTSDB 将其存储为 string 类型，其长度为 max_string_length，超过部分会被系统丢弃，max_string_length 的长度可自定义，默认为256个字符，修改请参考 更新 metric 。
- **日期与时间处理**：日期与时间在 CTSDB 中以 UTC 格式存储，因此，查询数据时涉及时间范围的查询请通过 time_zone 参数来指定时区，其格式为 ISO 8601 UTC 偏移（例如 +01:00 或 -08:00），具体时区需根据实例所处的地域来确定，一般国内地域是东八区。具体请参考 常用查询示例 。

连接实例

CTSDB 实例目前只提供 VPC 网络下的连接方式。您可以通过控制台连接实例，也可以通过 RESTful API 接口连接实例，通过 API 接口连接实例时需要提供 root 账号的密码，以确保安全性。
CURL 连接实例创建表示例如下，其中 `${user:password}` 是实例的用户名和密码，`${vip}:${vport}` 是实例的 IP 和 Port，`${metric_name}` 是新建的表名称。

```
curl -u ${user:password} -H 'Content-Type:application/json' -X PUT ${vip}:${vport}/_metric/${metric_name}
-d'
{
    "tags": {
        "region": "string",
        "set":  "long",
        "host": "string"
    },
    "time": {
         "name": "timestamp",
         "format": "epoch_second"
    },
    "fields": {
        "cpu_usage":  "float"
    },
    "options": {
```

```
        "expire_day": 7,
        "refresh_interval": "10s",
        "number_of_shards": 5,
        "number_of_replicas": 1,
        "rolling_period": 1
    }
}'
```

新建 metric

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

- 路径：`/_metric/${metric_name}`，`${metric_name}` 为新建的 metric 的名称。
- 方法：PUT

> **说明：**
> `metric` 命名限制请参见 系统限制 。

请求参数

无

请求内容

参数名称	必选	类型	描述
tags	是	map	维度列，用于唯一标识数据，须至少包含一个维度，支持的数据类型：text（带有分词、全文索引的字符串）、string（不分词的字符串）、long、integer、short、byte、double、float、date、boolean。 格式如 `{"region": "string","set": "long","host": "string"}`
time	是	map	时间列相关配置，用于存储数据入库的唯一时间，例如 `{"name": "timestamp", "format": "epoch_second"}`，填写时，需填写完整，name 和 format 的值不能为空
fields	否	map	数据列，用于存储数据，为了节省空间，建议使用最适合实际业务使用的类型，支持的数据类型：string（字符串）、long、integer、short、byte、double、float、date、boolean。 例如 `{"cpu_usage":"float"}`
options	否	map	常用的调优配置信息 例如 `{"expire_day":7,"refresh_interval":"10s","number_of_shards":5,"number_of_replicas":1,"rolling_period":1,"max_string_length":256,"default_date_format":"strict_date_optional_time","indexed_fields":["host"]}`

- time 字段的 name 默认为 timestamp，时间格式（format）完全兼容 Elasticsearch 的时间格式，如 epoch_millis（以毫秒为单位的 Unix 时间戳）、epoch_second（以秒为单位的 Unix 时间戳）、basic_date（格式如 yyyyMMdd）、basic_date_time（格式如 yyyyMMdd'T'HHmmss.SSSZ）等。
- options 选项及解释如下：
- expire_day：数据过期时间（单位：天），取值范围为非零整数，过期后数据自动清理，缺省情况下为最小值-1（代表永不过期）。
- refresh_interval：数据刷新频率，写入的数据从内存刷新到磁盘后可查询。默认为10秒。
- number_of_shards：表分片数，取值范围为正整数，小表可以忽略，大表按照一个分片至多25G设置分片数，默认为3。
- number_of_replicas：副本数，取值范围须设置为正整数，例如一主一副为1，缺省为1；不可设置为0，否则数据没有副本，存在丢失风险。
- rolling_period
  ：子表时长（单位：天），取值范围为非零整数，CTSDB 存储数据时，为了方便做数据过期和提高查询效率，根据特定时间间隔划分**子表**
  ，缺省情况下由数据过期时间决定，下面具体说明缺省子表时长和过期时间的关系。
- max_string_length：自定义字符串 string 类型的值最大可支持的长度，取值范围为正整数，最大为32765，默认为256。
- default_date_format：自定义维度列和指标列 date类型的格式，默认为 strict_date_optional_time 或 epoch_millis。
- indexed_fields：指定指标列中需要保留索引的字段，可指定多个，以数组形式指定。
- default_type：指定新增字段的默认类型。可选项为 tag、field，系统默认值为 tag。

过期时间	子表时长
≤ 7天	1天
> 7天，≤ 20天	3天
> 20天，≤ 49天	7天
> 49天，≤ 3个月	15天
>3个月	30天
永不过期	30天

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT 172.16.345.14:9201/_metric/ctsdb_test -d'
 {
        "tags":
        {
            "region":"string"
        },
        "time":
        {
            "name":"timestamp",
            "format":"epoch_second"
        },
        "fields":
        {
            "cpuUsage":"float"
        },
        "options":
        {
            "expire_day":7,
            "refresh_interval":"10s",
            "number_of_shards":5
        }
    }
    '
```

成功的返回：

```
    {
        "acknowledged": true,
        "message": "create ctsdb metric ctsdb_test success!"
    }
```

失败的返回：

```
   {
        "error": {
        "reason": "metric ctsdb_test already exist",
        "type": "metric_exception"
        },
        "status": 429
```

```
    }
```

查询 metric

最近更新时间：2024-08-30 11:31:01

获取所有 metric

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metrics`
方法：GET

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_metrics
```

返回：

```
{
    "result":
    {
        "metrics":
        [
            "ctsdb_test",
            "ctsdb_test1"
        ]
    },
    "status": 200
}
```

获取特定 metric

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}`，`${metric_name}`为metric的名称。
方法：GET

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_metric/ctsdb_test
```

返回：


```
{
    "result":
    {
        "ctsdb_test":
        {
            "tags":
            {
                "region": "string"
            },
            "time":
            {
                "name": "timestamp",
                "format": "epoch_second"
            },
            "fields":
            {
                "cpuUsage": "float"
            },
            "options":
            {
                "expire_day": 7,
                "refresh_interval": "10s",
                "number_of_shards": 5
            }
        }
    },
    "status": 200
}
```

更新 metric

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}/update`，`${metric_name}` 为 metric 的名称。
方法：PUT

请求参数

无

请求内容

字段 tags、time、fields、options 均为 map 类型的，选填，格式请参考 API 新建 metric 。具体要求如下：
tags：允许新增维度字段和修改已存在的维度字段类型，不会删除原有字段。
time：name 不能修改，format 允许修改。
fields：允许新增指标字段和修改已存在的指标字段类型，不会删除原有字段。
options 属性如下：

属性名称	必选	类型	描述
expire_day	否	integer	数据过期时间，取值范围为非零整数，过期后数据自动清理，缺省情况下永不过期
refresh_interval	否	string	数据刷新频率，写入的数据从内存刷新到磁盘后可查询，默认为10秒
number_of_shards	否	integer	表分片数，取值范围为正整数，小表可忽略，大表按照一个分片至多25G设置分片数，默认为3
number_of_replicas	否	integer	副本数，取值范围须设置为正整数，例如一主一副为1，缺省为1；不可设置为0，否则数据没有副本，存在丢失风险。
rolling_period	否	integer	子表时长（单位：天），取值范围为非零整数 CTSDB 存储数据时，为了方便做数据过期和提高查询效率，根据特定时间间隔划分子表，缺省情况下由数据过期时间决定
max_string_length	否	integer	自定义字符串 string 类型的值最大可支持的长度，取值范围为正整数，最大为32765，默认为256
default_date_format	否	string	自定义维度列和指标列 date 类型的格式，默认为 strict_date_optional_time 或 epoch_millis
indexed_fields	否	array	指定指标列中需要保留索引的字段，可指定多个，以数组形式指定
default_type	否	string	指定新增字段的默认类型。可选项为 tag、field，系统默认值为 tag

说明：

- 由于历史数据不可被修改，更新字段后 metric 信息不会立即变更，需要等待下一个 子表 产生。如果需要确认更新操作是否成功，可通过 `GET /_metric/${metric_name}?v` 接口进行确认。
- 只有为 short、integer、float 的字段才允许修改类型。其中 short 类型可被修改为 integer 和 long 类型；integer 类型可被修改为 long 类型；float 类型可被修改为 double 类型。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_metric/ctsdb_test/update -d'
    {
        "tags":{
            "set":"string"
        },
        "time":{
            "name":"timestamp",
            "format":"epoch_second"
        },
        "fields":{
            "diskUsage":"float"
        },
        "options":{
            "expire_day":15,
            "number_of_shards":10
        }
    }'
```

返回：

```
{
    "acknowledged": true,
    "message": "update ctsdb metric test111 success!"
}
```

删除 metric 字段

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}/delete`，`${metric_name}` 为 metric 的名称。
方法：PUT

请求参数

无

请求内容

参数名称	必选	类型	描述
tags	否	Array	枚举需要删除的维度字段，如 `"tags": ["ip"]`
fields	否	Array	枚举需要删除的数据字段，如 `"fields": ["diskUsage"]`

> **说明**
> 由于历史数据不可被修改，删除字段后 metric 信息不会立即变更，需要等待下一个 子表 产生。如果需要确认删除操作是否成功，可通过 `GET /_metric/${metric_name}?v` 接口进行确认。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_metric/ctsdb_test1/delete -d'
{
    "tags": ["ip"],
    "fields": ["cpu"]
}'
```

返回：

```
{
    "acknowledged": true,
    "message": "update ctsdb_test1 metric success!"
}
```

删除 metric

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_metric/${metric_name}`，`${metric_name}` 为需要删除的 metric 的名称。
方法：DELETE

请求参数

无

请求内容

无

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

请求：

```
DELETE /_metric/ctsdb_test1
```

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.16.345.14:9201/_metric/ctsdb_test1
```

返回：

```
{
    "acknowledged": true,
    "message": "delete metric ctsdb_test1 success!"
}
```

查询数据

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`${metric_name}/_search`，`${metric_name}`为 metric 的名称。
方法：GET

请求参数

可设置写入数据时的 `_routing` 参数值来提高查询效率，具体请参见 常用查询示例 。

请求内容

查询主要有普通查询和聚合查询，查询请求完全兼容 Elasticsearch API，具体请求内容请参照 常用查询示例 。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

CURL 示例说明

具体查询示例请参考 常用查询示例 。

批量查询数据

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

路径：`/_msearch`
方法：GET

请求参数

可通过设置写入数据时的 _routing 参数值来提高查询效率，具体请参照示例。也可通过设置 filter_path 参数来过滤并简化返回结果，其参数值为按照逗号分隔的多个 json 路径，其中单个 json 路径用点号连接，可以使用 * 通配符来适配任意字符， ** 通配符可以用来匹配任意路径，可以使用 - 前缀来去除某些返回字段。
例如，responses._shards.f* 表示 responses 下 _shards 中的以 f 开头的字段； responses.**._score 表示 responses 中所有 _score 的字段。

请求内容

查询主要有普通查询和聚合查询，查询请求完全兼容 Elasticsearch api，具体请求内容请参照示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。这里列举几个查询通用返回字段，详情见下表，若需要简化查询返回结果，可设置 filter_path 参数。

字段名称	描述
hits	返回匹配的查询结果，其中 total 字段表示匹配到的数据条数。里面的 hits 字段为数组结构，若无指定，则包含所有查询结果的前十条。hits 数组里的每个结果中包含 _index（查询涉及到的 子表），若查询中指定了 docvalue_fields，则会返回 fields 字段具体指明每个字段的值。
took	整个查询耗费的毫秒数。
_shards	参与查询的分片数。其中 total 标识总分片数，successful 标识执行成功的数量，failed 标识执行失败的数量，skipped 标识跳过执行的分片数。
timed_out	查询是否超时，取值有 false 和 true。
status	查询返回的状态码，2XX 代表成功。

CURL 示例说明

无 filter_path 参数的请求
请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_msearch -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

返回：

```
{
      "responses":
        [
            {
```

```
            "took": 0,
            "timed_out": false,
            "_shards": {
              "total": 0,
              "successful": 0,
              "skipped": 0,
              "failed": 0
            },
            "hits": {
              "total": 0,
              "max_score": 0,
              "hits": []
            },
            "status": 200
          },
          {
            "took": 1,
            "timed_out": false,
            "_shards": {
              "total": 3,
              "successful": 3,
              "skipped": 0,
              "failed": 0
            },
            "hits": {
              "total": 1,
              "max_score": 1,
              "hits": [
                {
                  "_index": "ctsdb_test2017@-979200000_30",
                  "_type": "doc",
                  "_id": "AV_8fBhlUAkC9PF9L-2t",
                  "_score": 1
                }
              ]
            },
            "status": 200
          }
        ]
    }
```

有 filter_path 参数的请求

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_msearch?
filter_path=responses.\*\*.hits,responses._shards.f\*,-responses.\*\*._score -d'
{"index" : "amyli_weather","routing":"host_20"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
{"index" : "ctsdb_test2017","routing":"host_100"}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 10}
'
```

返回：

```
{
      "responses": [
    {
      "_shards": {
        "failed": 0
```

```
      }
    },
    {
      "_shards": {
        "failed": 0
      },
      "hits": {
        "hits": [
          {
            "_index": "ctsdb_test2017@-979200000_30",
            "_type": "doc",
            "_id": "AV_8fBhlUAkC9PF9L-2t"
          }
        ]
      }
    }
      ]
    }
```

说明

如果 filter_path 参数同时设置了匹配条件和去除条件，则返回结果先执行去除条件再执行匹配条件。

常用查询示例

最近更新时间：2024-08-30 11:31:01

本文主要介绍 CTSDB 中常用查询及关键字的使用，并给出简单 CURL 示例。所有示例使用的 metric 结构如下所示：

```
{
    "ctsdb_test" : {
      "tags" : {
        "region" : "string"
      },
      "time" : {
        "name" : "timestamp",
        "format" : "epoch_millis"
      },
      "fields" : {
        "cpuUsage" : "float",
        "diskUsage" : "string",
        "dcpuUsage" : "integer"
      },
      "options" : {
        "expire_day" : 7,
        "refresh_interval" : "10s",
        "number_of_shards" : 5
      }
    }
}
```

组合查询

这里的组合查询即可以用于单查询也可用于复合查询。查询体中的 query 关键字通过 Query DSL（Domain Specific Language）来定义查询条件。下文会先介绍如何构造过滤条件，再介绍怎样组合过滤条件和如何处理返回结果集。

常用过滤条件

1. Range

Range 即区间查询，Range 查询支持的字段类型包括 string、long、integer、short、double、float、date。Range 查询可包含的参数如下表所示：

参数名称	描述
gte	大于或等于
gt	大于
lte	小于或等于
lt	小于

> **说明**
> 当 Range 查询涉及时间类型时，可通过 format 参数来指定时间格式。具体时间格式请参考 新建 metric 。

时间范围查询的 CURL 示例：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
{
    "query": {
        "range": {
            "timestamp": {
```

```
                    "gte": "01/01/2018",
                    "lte": "03/01/2018",
                    "format": "MM/dd/yyyy",
                    "time_zone":"+08:00"
                }
            }
        }
    }'
```

数字范围查询 CURL 示例：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "range": {
                "cpuUsage": {
                    "gte": 1.0,
                    "lte": 10.0
                }
            }
        }
    }'
```

2. Terms

Terms 关键字用于查询时匹配特定字段，其值需要用中括号包裹。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        }
    }'
```

过滤条件组合

复合查询常使用 Bool 关键字来组合多个查询条件，常见的用于 Bool 查询的组合关键字有 filter（类似于 AND）、must_not（类似于 NOT）、should（类似于 OR）。为提高查询性能，请务必加上 time 字段的 range 查询，且 time 字段，不论查询时以何种方式写入，返回值统一为 epoch_millis 格式。query-bool-filter 的组合形式可显著提升查询性能，请务必采用这种查询方式。

1. AND 条件 CURL 示例说明

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
      "query": {
        "bool": {
          "filter": [
            {
              "range": {
                "timestamp": {
                  "format": "yyyy-MM-dd HH:mm:ss",
                  "gte": "2017-11-06 23:00:00",
                  "lt": "2018-03-06 23:05:00",
```

```
                "time_zone":"+08:00"
              }

            }
          },
          {
            "terms": {
              "region": ["sh"]
            }
          },
          {
            "terms": {
              "cpuUsage": ["2.0"]
            }
          }
        ]
      }
    },
     "docvalue_fields": [
      "cpuUsage",
      "timestamp"
      ]
  }'
```

说明
此查询条件类似于 timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-03-06 23:05:00' AND region=sh AND cpuUsage=2.0。

2. OR 条件 CURL 示例说明

```
  curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
  {
    "query": {
      "bool": {
        "filter": [
          {
            "range": {
              "timestamp": {
                "format": "yyyy-MM-dd HH:mm:ss",
                "gte": "2017-11-06 23:00:00",
                "lt": "2018-11-06 23:05:00",
                "time_zone":"+08:00"
              }
            }
          },
          {
            "term": {
              "region": "gz"
            }
          }
        ],
        "should": [
          {
            "terms": {
              "cpuUsage": ["2.0"]
            }
          },
          {
```

```
            "terms": {
              "cpuUsage": ["2.5"]
            }
          }
        ],
        "minimum_should_match": 1
      }
    },
    "docvalue_fields": [
      "cpuUsage",
      "timestamp"
    ]
  }'
```

说明

此查询条件类似于 timestamp>='2017-11-06 23:00:00' AND timestamp<'2018-11-06 23:05:00' AND region='gz' AND (cpuUsage=2.0 or cpuUsage=2.5)。minimum_should_match 参数的意义在于设置 cpuUsage=2.0 和 cpuUsage=2.5 至少匹配的个数，系统默认 minimum_should_match 为0。

3. NOT 条件 CURL 示例说明

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
      "query": {
        "bool": {
          "filter": [
                    {
              "range": {
                "timestamp": {
                  "format": "yyyy-MM-dd HH:mm:ss",
                  "gte": "2017-11-06 23:00:00",
                  "lt": "2018-11-06 23:05:00",
                  "time_zone":"+08:00"
                }

              }
            },
            {
              "terms": {
                "region": ["gz"]
              }
            }
          ],
          "must_not": [
           {
              "terms": {
                "cpuUsage": ["2.0"]
              }
            }
          ]
        }
      },
      "docvalue_fields": [
        "cpuUsage",
        "timestamp"
          ]
      }'
```

> **说明**
> 此查询条件类似于 timestamp>=2017-11-06 23:00:00 AND timestamp<2018-11-06 23:05:00 AND region='gz' AND cpuUsage !=2.0。

返回结果集处理

1. From/Size

通过设置 From 和 Size 关键字可对查询结果进行分页。From 关键字定义了查询结果中第一条数据的偏移量，Size 关键字配置返回结果的最大条数。From 系统默认为 0，Size 系统默认为 10，From 与 Size 的总和系统默认不能超过 65536。若想要更大的返回结果，请参考本文的 scroll 关键字查询。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "from": 0,
        "size": 5,
        "query": {
            "bool": {
                "filter": {
                    "range": {
                        "timestamp": {
                            "gte": "01/01/2018",
                            "lte": "03/01/2018",
                            "format": "dd/MM/yyyy",
                            "time_zone":"+08:00"
                        }
                    }
                },
                "must_not": {
                    "terms": {
                        "region": ["bj"]
                    }
                }
            }
        }
    }'
```

2. Scroll

Scroll 可以理解为关系型数据库里的 cursor，因此 scroll 并不适合用来做实时搜索，而更适用于后台批处理任务。
可以把 scroll 分为初始化和遍历两个阶段，初始化时将所有符合搜索条件的搜索结果缓存起来，类似于快照，在遍历时，从这个快照里取数据。注意，在 Scroll 初始化后对 metric 的插入、删除、更新都不会影响遍历结果。
初始化阶段可通过 size 关键字指定返回结果集的大小，size 默认值为10，最大值为65536；初始化阶段可通过 query 关键字指定查询条件；初始化阶段可通过 docvalue_fields 关键字指定返回字段。初始化阶段和遍历阶段都返回 `_scroll_id` 字段，后续遍历可以通过指定前一次遍历返回的 `_scroll_id` 来进行新的遍历，直到返回结果为空；初始化和遍历两个阶段都可以通过指定 scroll 参数来设定遍历的上下文环境保留时间，过期后 `_scroll_id` 将变得无效。时间格式如下表所示：

格式	描述
d	days
h	hours
m	minutes
s	seconds
ms	milliseconds

micros	microseconds
nanos	nanoseconds

CURL 示例说明：

scroll 初始化：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/ctsdb_test/_search?scroll=1m -d'
    {
       "size":5,
       "query": {
       "bool": {
         "filter": [
           {
             "terms": {
               "region": ["gz"]
             }
           }
         ]
       }
     },
    "docvalue_fields": [
          "cpuUsage",
        "region",
        "timestamp"
        ]
    }'
```

scroll 初始化返回：

```
    {
      "_scroll_id":
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRkZTcGp4dFRXR18xMGtzSmhEUFJRA
AAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw==",
      "took": 10641,
      "timed_out": false,
      "_shards": {
       "total": 3,
       "successful": 3,
       "skipped": 0,
       "failed": 0
      },
      "hits": {
        "total": 1592072666,
        "max_score": 0.65708643,
        "hits": [
          {
            "_index": "ctsdb_test@0_-1",
            "_type": "doc",
            "_id": "oyylNU0U65cZjByyt7sW_JmPPgAACY4Bh0",
            "_score": 0.65708643,
            "_routing": "354d14eb",
            "fields": {
              "region": [
                "gz"
              ],
              "cpuUsage": [
                "2.0"
```

```
        ],
        "timestamp": [
          1509909300000
        ]
      }
    },
    {
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyykFN0yd1d9NDPfzjRdrJ8whQAACYsBqc",
      "_score": 0.65708643,
      "_routing": "14dd3277",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "1.8"
        ],
        "timestamp": [
          1509908340000
        ]
      }
    },
    {
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABHIBso",
      "_score": 0.65708643,
      "_routing": "1cba7d8e",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "2.5"
        ],
        "timestamp": [
          1509909720000
        ]
      }
    },
    {
      "_index": "ctsdb_test@0_-1",
      "_type": "doc",
      "_id": "oyylH2JsOKnFHGUinQ7jM-ZwkgAAAvcBso",
      "_score": 0.65708643,
      "_routing": "1f626c38",
      "fields": {
        "region": [
          "gz"
        ],
        "cpuUsage": [
          "2.1"
        ],
        "timestamp": [
          1509909720000
        ]
      }
    },
    {
```

```
            "_index": "ctsdb_test@0_-1",
            "_type": "doc",
            "_id": "oyylHLp9jl_3sF4N2rnh67h4SgAABGsBso",
            "_score": 0.65708643,
            "_routing": "1cba7d8e",
            "fields": {
              "region": [
                "gz"
              ],
              "cpuUsage": [
                "2.0"
              ],
              "timestamp": [
                1509909720000
              ]
            }
          }
        ]
          }
    }
```

scroll 遍历：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X POST 172.16.345.14:9201/_search/scroll  -
d'
    {
    "scroll" : "1m",
    "scroll_id" :
"DnF1ZXJ5VGhlbkZldGNoAwAAAAADrOFFm5YSEhnMjdnUWNPcndHS1k5Wjc3bHcAAAAAAAz_1RZiRkZTcGp4dFRXR18xMGtzSmhEUFJRA
AAAAAAP5vQWOXFOR29lc0hROHFWMmFGTkVmSkxmZw=="
    }'
```

说明

- 此请求中的 scroll_id 是 scroll 初始化返回的 _scroll_id 值。而下一次遍历则需要将 scroll_id 参数调整为上一次遍历返回的 _scroll_id 值，即每次请求中的 scroll_id 参数是上一次请求返回的 _scroll_id 值，直到返回结果为空，遍历结束。
- 两次遍历返回的 _scroll_id 值可能相同，无法利用 _scroll_id 进行指定页跳转。

3. Sort

Sort 关键字主要用于对查询结果进行排序。排序方式有 asc 和 desc 两种，CTSDB 对于用户自定义字段的默认排序方式为 asc。排序模式有 min、max、sum、avg、median。其中 sum、avg、median 只适用于类型为 array 并存储数字的字段。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "bool": {
                "must": {
                    "range": {
                        "timestamp": {
                            "gte": "01/01/2018",
                            "lte": "03/01/2018",
                            "format": "MM/dd/yyyy",
                            "time_zone":"+08:00"
                        }
```

```
                    }
                }
            }
        },
        "sort": [
            {
                "cpuUsage": {
                    "order": "asc",
                    "mode": "min"
                }
            },
            {
                "timestamp": {
                    "order": "asc"
                }
            },
            "diskUsage"
        ]
    }'
```

4. docvalue_fields

docvalue_fields 关键词指定需要返回的字段名称，需要以数组的形式指定。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "docvalue_fields": ["timestamp", "cpuUsage"]
    }'
```

聚合查询

agg 关键字主要用于构造聚合查询。用户可到返回的 aggregations 字段取聚合结果。聚合返回字段说明详见下表。如果只关注聚合结果，请在查询时设置 size 参数为0。

字段名称	描述
hits	返回匹配的查询结果，其中 total 字段表示参与聚合的数据条数。里面的 hits 字段为数组结构，若无指定，则包含所有查询结果的前十条。hits 数组里的每个结果中包含 `_index`（查询涉及到的 子表），若查询中指定了 docvalue_fields，则会返回 fields字段具体指明每个字段的值。
took	整个查询耗费的毫秒数。
_shards	参与查询的分片数。其中 total 标识总分片数，successful 标识执行成功的数量，failed 标识执行失败的数量，skipped 标识跳过执行的分片数。
timed_out	查询是否超时，取值有 false 和 true。
aggregations	聚合返回结果。

下面列举几种常用的聚合方式。

普通聚合

普通聚合需要指定聚合名称、聚合方式（常用的聚合方式有 min、max、avg、value_count、sum 等）和聚合所作用的字段。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "size":0,
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "myname": {
                "max": {
                    "field": "cpuUsage"
                }
            }
        }
    }'
```

说明
上例中对字段 cpuUsage 做 max 聚合（用户也可指定 min、avg 等），返回结果取别名 myname（用户可任意指定该名称）。

返回结果：

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 20,
    "successful": 20,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 7,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "myname": {
      "value": 4
    }
  }
}
```

terms 聚合

terms 聚合主要用于查询某个字段的所有唯一值以及该值的个数，用户可指定唯一值返回时的排序规则，以及返回结果数，并且可对参与聚合的数据字段进行模糊或者精确匹配，详情请参考如下示例,使用 filter_path 参数可定制返回结果字段，使用方法请参考 批量查询数据 。
CURL 示例说明：
请求：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?
filter_path=aggregations -d'
    {
```

```
        "aggs": {
            "myname": {
                "terms": {"field":"region"}
            }
        }
    }'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

说明

上例表示查看 ctsdb_test 的 region 字段中所有的唯一值及其出现的次数。分析返回字段 aggregations 中的 buckets 字段发现，region 字段共有7种类型的值，分别是 sh、Motor_sports、gz、bj、cd、Winter_sports、water_sports， 并且返回字段通过 doc_count 指明了每种值的个数。

用户可通过 size 字段指定返回的唯一值的个数，例如某字段名为 region，其唯一值有7个，可通过设置 size 字段为5指定只返回前5个，具体请参考示例。

请求：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?
filter_path=aggregations -d'
    {
        "aggs": {
            "myname": {
                "terms": {
                   "field":"region",
                    "size":5
                }
            }
        }
    }'
```

返回:

```
    {
      "aggregations": {
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 2,
          "buckets": [
            {
              "key": "sh",
              "doc_count": 10
            },
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "gz",
              "doc_count": 3
            },
            {
              "key": "bj",
              "doc_count": 2
            },
            {
              "key": "cd",
              "doc_count": 2
            }
          ]
        }
      }
    }
```

用户可对返回结果进行排序，详情请参考如下示例。

1.请求（返回结果按照唯一值的个数降序排列）：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?
filter_path=aggregations -d'
    {
      "aggs": {
          "myname": {
              "terms": {
                "field":"region",
                "order":{"_count":"desc"}
```

```
            }
        }
    }
}'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "sh",
          "doc_count": 10
        },
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "gz",
          "doc_count": 3
        },
        {
          "key": "bj",
          "doc_count": 2
        },
        {
          "key": "cd",
          "doc_count": 2
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        },
        {
          "key": "water_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

2.请求（返回结果按照唯一值的字母升序进行排列）：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
  "aggs": {
      "myname": {
          "terms": {
            "field":"region",
            "order":{"_term":"asc"}
          }
```

```
        }
      }
    }'
```

返回：

```
    {
      "aggregations": {
        "myname": {
          "doc_count_error_upper_bound": 0,
          "sum_other_doc_count": 0,
          "buckets": [
            {
              "key": "Motor_sports",
              "doc_count": 6
            },
            {
              "key": "Winter_sports",
              "doc_count": 1
            },
            {
              "key": "bj",
              "doc_count": 2
            },
            {
              "key": "cd",
              "doc_count": 2
            },
            {
              "key": "gz",
              "doc_count": 3
            },
            {
              "key": "sh",
              "doc_count": 10
            },
            {
              "key": "water_sports",
              "doc_count": 1
            }
          ]
        }
      }
    }
```

用户可设置正则表达式对参与 terms 聚合的数据字段进行模糊匹配或者指定字段进行精确匹配，详情请参考示例。
模糊匹配示例：
请求（只针对 region 字段中有 sport 并且不是以 water_ 开头的数据进行聚合并返回结果）：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?
filter_path=aggregations -d'
    {
      "aggs": {
          "myname": {
              "terms": {
                "field":"region",
                "include" : ".*sport.*",
                "exclude" : "water_.*"
```

```
            }
        }
    }
}'
```

返回：

```
{
  "aggregations": {
    "myname": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Motor_sports",
          "doc_count": 6
        },
        {
          "key": "Winter_sports",
          "doc_count": 1
        }
      ]
    }
  }
}
```

精确匹配示例

请求（region_zone 指定值为"sh"，"bj"，"cd"，"gz"的 region 字段进行聚合，而 region_sports 指定值不为"sh"，"bj"，"cd"，"gz"的 region 字段进行聚合）

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search?filter_path=aggregations -d'
{
"aggs" : {
  "region_zone" : {
    "terms" : {
      "field" : "region",
      "include" : ["sh", "bj","cd","gz"]
      }
    },
    "region_sports" : {
     "terms" : {
         "field" : "region",
         "exclude" : ["sh", "bj","cd","gz"]
     }
    }
  }
}'
```

返回：

```
{
  "aggregations": {
    "region_sport": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Motor_sports",
```

```
            "doc_count": 6
          },
          {
            "key": "Winter_sports",
            "doc_count": 1
          },
          {
            "key": "water_sports",
            "doc_count": 1
          }
        ]
      },
      "region_zone": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": [
          {
            "key": "sh",
            "doc_count": 10
          },
          {
            "key": "gz",
            "doc_count": 3
          },
          {
            "key": "bj",
            "doc_count": 2
          },
          {
            "key": "cd",
            "doc_count": 2
          }
        ]
      }
    }
  }
```

Date Histogram 聚合

Date Histogram 主要对日期做直方图聚合。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs": {
            "time_1h_agg": {
                "date_histogram": {
                    "field": "timestamp",
                    "interval": "1h"
                },
                "aggs": {
                    "avgCpuUsage": {
                        "avg": {
                            "field": "cpuUsage"
```

```
                    }
                }
            }
        }
    }
}'
```

返回：

```
{
  "took": 5,
  "timed_out": false,
  "_shards": {
    "total": 20,
    "successful": 20,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 6,
    "max_score": 0.074107975,
    "hits": [
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf-",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf_",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgA",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgB",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgC",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
```

```
        "_id": "AWH2RGfF5xcjRaw2ETgD",
        "_score": 0.074107975,
        "_routing": "sh"
      }
    ]
  },
  "aggregations": {
    "time_1h_agg": {
      "buckets": [
        {
          "key_as_string": "1520222400",
          "key": 1520222400000,
          "doc_count": 1,
          "avgCpuUsage": {
            "value": 2.5
          }
        },
        {
          "key_as_string": "1520226000",
          "key": 1520226000000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520229600",
          "key": 1520229600000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520233200",
          "key": 1520233200000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520236800",
          "key": 1520236800000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520240400",
          "key": 1520240400000,
          "doc_count": 0,
          "avgCpuUsage": {
            "value": null
          }
        },
        {
          "key_as_string": "1520244000",
          "key": 1520244000000,
          "doc_count": 0,
```

```
            "avgCpuUsage": {
              "value": null
            }
          },
          {
            "key_as_string": "1520247600",
            "key": 1520247600000,
            "doc_count": 1,
            "avgCpuUsage": {
              "value": 2
            }
          },
          {
            "key_as_string": "1520251200",
            "key": 1520251200000,
            "doc_count": 4,
            "avgCpuUsage": {
              "value": 2.25
            }
          }
        ]
      }
    }
  }
```

说明

上例是对字段 cpuUsage 进行粒度为1小时的 date_histogram 聚合。返回结果中总的聚合名称为 time_1h_agg（用户可任意指定该名称），每个时间分段内的聚合名称为 avgCpuUsage（用户可任意指定该名称）。interval 字段可选的时间粒度有 year、quarter、 month、week、day、hour、minute、second，用户也可将时间粒度用时间单位来表示，例如1y代表1year，1h代表1hour。系统不支持小数粒度的时间单位，例如1.5h您需要转换为90min。

Percentiles 聚合

Percentiles 聚合即百分位聚合。百分位可以任意指定，系统默认的百分位是 1、5、25、50、75、95、99，可自行选择其他数值。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs":
        {
            "myname":
            {
                "percentiles":
                {
                    "field": "cpuUsage",
                    "percents": [1,25,50,70,99]
                }
            }
        }
    }'
```

返回：

```
{
  "took": 18,
  "timed_out": false,
  "_shards": {
    "total": 20,
    "successful": 20,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 6,
    "max_score": 0.074107975,
    "hits": [
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf-",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2QtGR5xcjRaw2ETf_",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgA",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2Q5Xr5xcjRaw2ETgB",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgC",
        "_score": 0.074107975,
        "_routing": "sh"
      },
      {
        "_index": "ctsdb_test@1520092800000_3",
        "_type": "doc",
        "_id": "AWH2RGfF5xcjRaw2ETgD",
        "_score": 0.074107975,
        "_routing": "sh"
      }
    ]
  },
  "aggregations": {
    "myname": {
      "values": {
```

```
            "1.0": 2,
            "25.0": 2,
            "50.0": 2.25,
            "70.0": 2.5,
            "99.0": 2.5
          }
        }
      }
    }
```

说明

上例中是对字段 cpuUsage 做 percentiles 聚合，选取的百分位为 1,25,50,70,99。聚合返回结果的别名是 myname（用户可任意指定该名称）。

Cardinality 聚合

Cardinality 聚合主要用于统计去重后的数量。默认情况下，当聚合结果小于等于3000时，Cardinality 返回的结果是精确值，大于3000时，为近似值。
CURL 示例说明：

```
    curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/ctsdb_test/_search
-d'
    {
        "query": {
            "terms": {
                "region": ["sh", "bj"]
            }
        },
        "aggs":
        {
            "myname":
            {
                "cardinality":
                {
                    "field": "cpuUsage"
                }
            }
        }
    }'
```

返回：

```
    {
      "took": 15,
      "timed_out": false,
      "_shards": {
        "total": 20,
        "successful": 20,
        "skipped": 0,
        "failed": 0
      },
      "hits": {
        "total": 6,
        "max_score": 0.074107975,
        "hits": [
          {
            "_index": "ctsdb_test@1520092800000_3",
            "_type": "doc",
            "_id": "AWH2QtGR5xcjRaw2ETf-",
            "_score": 0.074107975,
            "_routing": "sh"
```

```
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2QtGR5xcjRaw2ETf_",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgA",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2Q5Xr5xcjRaw2ETgB",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgC",
          "_score": 0.074107975,
          "_routing": "sh"
        },
        {
          "_index": "ctsdb_test@1520092800000_3",
          "_type": "doc",
          "_id": "AWH2RGfF5xcjRaw2ETgD",
          "_score": 0.074107975,
          "_routing": "sh"
        }
      ]
    },
    "aggregations": {
      "myname": {
        "value": 2
      }
    }
  }
```


说明

上例的聚合是对字段 cpuUsage 做 cardinality 聚合，返回的聚合结果取别名 myname（用户可任意指定该名称）。

批量写入数据

最近更新时间：2024-08-30 11:31:01

往单个 metric 批量写入数据

此接口同时适用于批量和非批量写入数据至单个 metric 的场景，为提高写入性能，建议批量写入数据。

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`/${metric_name}/doc/_bulk`，${metric_name}为 metric 的名称。
方法：POST

> **说明**
> doc 关键字为写入数据的 _type，为了便于以后系统做解析和升级，请务必加上 doc 关键字。

请求参数

可通过设置 filter_path 参数来过滤并简化返回结果，具体请参考 批量查询数据 。

请求内容

批量写入 metric 需要一种 NDJSON 格式的结构数据，类似于如下：

```
元数据\n
需要写入的数据\n
....
元数据\n
需要写入的数据\n
```

元数据的格式如下所示：

```
{
    "index" :
    {
        "_id" : "1",                #文档 ID(选填)
        "_routing": "sh"             #路由值(选填)
    }
}
```

写入数据的格式类似于：

```
{
    "field1" : "value1",
    "field2" : "value2"
}
```

> **注意**
> 写入数据时可通过设置 _routing 参数来路由数据写入的分片，该参数选填，可指定任意值。指定相同的 _routing 值可将不同数据路由至同一分片上。另外，查询时指定已设置的 _routing 参数值，系统会根据路由到指定分片查询数据，可以极大优化查询速度。请求体中最后一行需要加换行符。

返回内容

需要注意的是，批量写入数据接口与其他接口返回结果稍有不同。请优先关注 json 结果的 errors（注意不是 error）字段，如果为 false，代表所有数据写入成功；如果为 true，代表有部分写入失败，具体失败的详情可以通过 items 字段得到。

items 字段为一个数组，数组中每一个元素与写入请求一一对应，可通过每个元素是否有 error 字段判断请求是否成功，若有 error 字段则表示请求失败，具体错误内容在 error 字段内，若无 error 字段表示请求成功。

CURL 示例说明

返回成功的示例：

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST 172.16.345.14:9201/ctsdb_test/doc/_bulk -
d'
{"index":{"_routing": "sh" }}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_routing": "sh" }}
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

返回：

```
{
    "took": 134,
    "errors": false,
    "items":
    [
        {
            "index":
            {
                "_index": "ctsdb_test@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2q",
                "_version": 1,
                "result": "created",
                "_shards":
                 {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
                }
            },
            {
            "index":
            {
                "_index": "ctsdb_test2@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2r",
                "_version": 1,
                "result": "created",
                "_shards":
                {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
            }
        }
    ]
```

```
}
```

> **说明**
> 上面返回中的 errors 为 false，代表所有数据写入成功。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，_index 标识了写入的 metric 子表，_shards 记录副本写入情况，上例中 total 表示两副本，successful 代表两副本均写入成功。

返回失败的示例：
请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/hcbs_client_trace/doc/_bulk -d'
{"index":{"_type":"type"}}
{"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":2,"data_len":4096,"latency":3,"try_times":1,"errcode":0,"start_time":1503404266}
{"index":{"_type":"type"}}
{"vol_id":"c57e008c-0ae0-41cd-8da8-
6989d0522fc6","io_type":"abc","data_len":4096,"latency":1,"try_times":1,"errcode":0,"start_time":150340426
6}
'
```

返回：

```
{
  "took": 7,
  "errors": true,
  "items": [
    {
      "index": {
        "_index": "hcbs_client_trace",
        "_type": "type",
        "_id": "AWMe9r9lNifptzIWMVPT",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "created": true,
        "status": 201
      }
    },
    {
      "index": {
        "_index": "hcbs_client_trace",
        "_type": "type",
        "_id": "AWMe9r9lNifptzIWMVPU",
        "status": 400,
        "error": {
          "type": "mapper_parsing_exception",
          "reason": "failed to parse [io_type]",
          "caused_by": {
            "type": "number_format_exception",
            "reason": "For input string: \"abc\""
          }
        }
      }
```

```
    }
  ]
}
```

> **说明**
> 上面返回中的 errors 为 true，代表部分数据写入失败。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，error 字段详细给出了错误内容，_index 标识了写入的 metric 子表，_shards 记录副本写入情况，上例中 total 表示两副本，successful 代表两副本均写入成功。

往多个 metric 批量写入数据

此接口同时适用于批量和非批量写入单个 metric 或者多个 metric 的场景。为提高写入性能，建议批量写入数据。

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`_bulk`
方法：PUT

> **说明**
> doc 关键字为写入数据的 _type，为了便于以后系统做解析和升级，请务必加上 doc 关键字。

请求参数

可通过设置 filter_path 参数来过滤并简化返回结果，具体请参考 批量查询数据 。

请求内容

批量写入 metric 需要一种 NDJSON 格式的结构数据，类似于如下：

```
元数据\n
需要写入的数据\n
....
元数据\n
需要写入的数据\n
```

元数据的格式如下所示：

```
{
    "index" :
    {
        "_index" :"metric_name",     #要写入的 metric
        "_type" : "doc",             #写入的文档类型
        "_id" : "1",                #文档 ID(选填)
        "_routing": "sh"             #路由值(选填)
    }
}
```

写入数据的格式类似于：

```
 {
    "field1" : "value1",
    "field2" : "value2"
}
```

> **说明**
> 写入数据时可通过设置 _routing 参数来路由数据写入的分片，该参数选填，可指定任意值。指定相同的 _routing 值可将不同数据路由至同一分片上。另外，查询时指定已设置的 _routing 参数值，系统会根据路由到指定分片查询数据，可以极大优化查询速度。注意，请求体中最后一行需要加换行符。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误内容在 error 字段内。注意：若请求成功，但是 errors（注意不是 error）字段非等于 false，则该 errors 字段具体指出写入失败的具体数据。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT 172.16.345.14:9201/_bulk -d'
{"index":{"_index" : "ctsdb_test", "_type" : "doc", "_routing": "sh" }}
{"region":"sh","cpuUsage":2.5,"timestamp":1505294654}
{"index":{"_index" : "ctsdb_test2", "_type" : "doc", "_routing": "sh" }}
{"region":"sh","cpuUsage":2.0,"timestamp":1505294654}
'
```

返回：

```
{
    "took": 134,
    "errors": false,
    "items":
    [
        {
            "index":
            {
                "_index": "ctsdb_test@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2q",
                "_version": 1,
                "result": "created",
                "_shards":
                 {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
                "created": true,
                "status": 201
                }
            },
            {
            "index":
            {
                "_index": "ctsdb_test2@1505232000000_1",
                "_type": "doc",
                "_id": "AV_8eeo_UAkC9PF9L-2r",
                "_version": 1,
                "result": "created",
                "_shards":
                {
                    "total": 2,
                    "successful": 2,
                    "failed": 0
                },
```

```
                "created": true,
                "status": 201
            }
        }
    ]
}
```

说明

上面返回中的 errors 为 false，代表所有数据写入成功。items 数组标识每一条记录写入结果，与 bulk 请求中的每一条写入顺序对应。items 的单条记录中，status 为 2XX 代表此条记录写入成功，_index 标识了写入的 metric 子表，_shards 记录副本写入情况，上例中 total 表示两副本，successful 代表两副本均写入成功。

删除数据

最近更新时间：2024-08-30 11:31:01

请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如10.13.20.15:9200。

请求路径和方法

请求路径：`/${metric_name}/_delete_by_query`，`${metric_name}`为 metric 的名称。
方法：POST

请求参数

无

请求内容

删除 metric 时的查询条件，具体请参考示例。

返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误内容在 error 字段内。

CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/ctsdb_test/_delete_by_query -d'
{
"query": {
    "bool": {
        "filter": {
            "match_all": {}
            }
        }
    }
}'
```

返回：

```
{
    "took": 43,
    "timed_out": false,
    "total": 1,
    "deleted": 1,
    "batches": 1,
    "version_conflicts": 0,
    "noops": 0,
    "retries": {
        "bulk": 0,
        "search": 0
    }
}
```

修改数据

最近更新时间：2024-08-30 11:31:01

CTSDB 修改数据可使用先删除数据再写入数据的方式。数据的删除和写入，请参见 删除数据 和 批量写入数据 。

Rollup 相关操作

最近更新时间：2024-08-30 11:31:01

建立 Rollup 任务

在海量数据场景下，业务系统每天甚至每小时会产生 PB 级别数据。而时序数据的最主要的特点就是海量性、时效性和趋势性，因此通常情况下，使用数据的系统（例如监控系统或数据分析系统）通常只需要最近时间段内的高精度数据，而历史数据只需降精度（Downsampling）保存即可。用户可通过配置 Rollup 任务定时聚合历史数据保存至新的数据表。Rollup 任务不仅能降精度保存历史数据，也能提高查询性能，降低存储成本。需要注意的是，Rollup 任务会自动根据 base_metric 建立子表，继承父表的所有配置，如果指定 options，会覆盖父表配置。

1. 请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}`为 Rollup 任务的名称。
方法：PUT

3. 请求参数

无

4. 请求内容

参数名称	必选	类型	描述
base_metric	是	string	Rollup 依赖的 metric 名称（父表）
rollup_metric	是	string	Rollup 产生的 metric 名称（子表）
base_rollup	否	string	依赖的 Rollup 任务，任务执行前会检查相应时间段的依赖任务是否完成执行
query	否	string	过滤数据的查询条件，由很多个元素和操作对组成，例如：`name:host AND type:max OR region:gz`
group_by	是	Array	进行聚合的维度列，可以包含多列
function	是	Map	指定聚合的名称、方法和字段，其字段只能选自 base_metric 里的 fields 字段，如果 base_metric 的 fields 为空，则无法设置 rollup，function 有 sum、avg、min、max、set、any、first、last、percentiles 等。例如： `{"cost_total":{"sum": {"field":"cost"}},"cpu_usage_avg":{ "avg": { "field":"cpu_usage"}}}`
interval	是	string	聚合粒度（rollup 产生数据的时间精度），例如1s、5m（5分钟）、1h、1d等
frequency	否	string	调度频率，例如5m、1h、1d等，默认等于 interval
delay	否	string	延迟执行时间，写入数据通常有一定的延时，避免丢失数据，例如5m、1h等
start_time	否	string	开始时间，从该时间开始周期性执行 Rollup，默认为当前时间
end_time	否	string	结束时间，到达该时间后不再调度 ，默认为时间戳最大值
options	否	map	rollup_metric 选项，跟新建 metric 选项一致

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X PUT
172.16.345.14:9201/_rollup/ctsdb_rollup_task_test -d'
{
    "base_metric": %{base_metric_name},
    "rollup_metric": %{rollup_metric_name},
    "base_rollup": %{base_rollup_name},
    "query" : "name:host AND type:max",
    "group_by": ["host"],
    "function": {
        "cost_total":  {
            "sum": {
                "field": "cost"
            }
        },
        "cpu_usage_avg": {
            "avg": {
                "field": "cpu_usage"
            }
        },
        "value": {
          "percentiles": {
            "field": "value",
            "percents": [
              95
            ]
          }
        },
        "metricName": {
          "set": {
            "value": "cpu_usage"
          }
        },
        "appid": {
          "any": {
            "field": "appid"
          }
        },
         "first_value": {
          "first": {
            "field": "value"
          }
        },
         "last_value": {
          "last": {
            "field": "value"
          }
        }
    },
    "interval": "1m",
    "frequency": "5m",
    "delay": "1m",
    "start_time": "1502892000",
    "end_time": "2147483647",
    "options": {
        "expire_day": 365
    }
}
'
```

返回：

```
{
    "acknowledged": true,
    "message": "create rollup success"
}
```

获取所有 Rollup 任务的名称

1. 请求地址

地址为实例的 IP 和 PORT，可从控制台获取到，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollups`
方法：GET

3. 请求参数

无

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_rollups
```

返回：

```
{
    "result":
    {
        "rollups":
        [
            "rollup_jgq_6",
            "rollup_jgq_60"
        ]
    },
    "status": 200
}
```

获取某个 Rollup 任务的详细信息

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}` 为 Rollup 任务的名称。
方法：GET

3. 请求参数

指定 v 参数可以查看 rollup 的具体进度，返回结构中的 @last_end_time 为 rollup 最新进度。

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X GET 172.16.345.14:9201/_rollup/rollup_jgq_6?v
```

返回：

```
{
  "result": {
    "rollup_jgq_6": {
    "base_metric": "cvm_device-300",
       "rollup_metric": "cvm_device-86400",
       "query": "metricName:cpu_usage AND statType:max",
       "group_by": [
         "vm_uuid"
       ],
       "function": {
         "value": {
           "percentiles": {
             "field": "value",
             "percents": [
               95
             ]
           }
         },
         "metricName": {
           "set": {
             "value": "cpu_usage"
           }
         },
         "appid": {
           "any": {
             "field": "appid"
           }
         }
       },
       "interval": "1d",
       "delay": "5m",
       "options": {
         "expire_day": 186
       },
       "frequency": "1d",
       "start_time": 1534003200,
       "end_time": 2147483647,
       "@state": "running",              // 运行状态
       "@timestamp": 1550766085000,      // rollup 任务信息更新的时间点
       "@last_end_time": 1550764800      // rollup 任务正确执行到的时间点

    }
  },
  "status": 200
}
```

删除 Rollup 任务

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}`，`${rollup_task_name}`，为 Rollup 任务的名称。
方法：DELETE

3. 请求参数

无

4. 请求内容

无

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X DELETE 172.16.345.14:9201/_rollup/ctsdb_rollup_task_test
```

返回：

```
{
    "acknowledged": true,
    "message": "delete rollup success"
}
```

更新 Rollup 任务

1. 请求地址

地址为实例的 IP 和 PORT，例如：10.13.20.15:9200。

2. 请求路径和方法

路径：`/_rollup/${rollup_task_name}/update`，`${rollup_task_name}` 为 Rollup 任务的名称。
方法：POST

3. 请求参数

无

4. 请求内容

参数名称	必选	类型	描述
state	是	string	running/pause
start_time	否	string	开始时间，从该时间开始周期性执行 Rollup，默认为当前时间
end_time	否	string	结束时间，到达改时间后不再调度，默认为时间戳最大值
options	否	map	聚合选项，跟新建 metric 选项一致

5. 返回内容

需要通过 error 字段判断请求是否成功，若返回内容有 error 字段则请求失败，具体错误详情请参照 error 字段描述。

6. CURL 示例说明

请求：

```
curl -u root:le201909 -H 'Content-Type:application/json' -X POST
172.16.345.14:9201/_rollup/ctsdb_rollup_task_test/update -d'
{
    "state":"running",
    "start_time": "1511918989",
    "end_time": "1512019765",
    "options":
    {
        "expire_day": 365
    }
}'
```

返回：

```
{
    "acknowledged": true,
    "message": "update rollup success"
}
```