Jump to bottom

Dustin Sallings edited this page on 27 Sep 2012 · 1 revision

Managing Databases

Creating a Database

`PUT /dbname` will create the `dbname` database. Example using curl:

```
curl -X PUT http://localhost:3133/dbname
```

Listing Databases

`GET /_all_dbs` will list all known databases on the server. Example:

```
curl http://localhost:3133/_all_dbs
```

yields

```
["dbname"]
```

DB Info

`GET /dbname` will give you some info about the database.

Example using curl:

```
curl http://localhost:3133/dbname
```

should produce the following output:

```
{"deleted_count":0,
 "doc_count":3329786,
 "header_pos":750080000,
 "last_seq":3329788,
 "space_used":210578761}
```

In the above case, you'll notice that `header_pos` is way beyond `space_used`, indicating this database may benefit from compaction.

Compacting a Database

`POST /dbname/_compact` will clean up extra space in `dbname`.

Example using curl:

```
curl -X POST http://localhost:3133/dbname/_compact
```

Online compaction works in a (mostly) non-blocking way. If you exceed a database's `maxOpQueue` during compaction, writes to that database will begin blocking. If you feel that this may affect you, adjust `maxOpQueue` appropriately and/or compact during a low period. Queries will never be blocked.

Deleting a Database

`DELETE /dbname` will delete the `dbname` database. Example using curl:

```
curl -X DELETE http://localhost:3133/dbname
```

Storing Data

To store a JSON document with a system-generated timestamp, just `POST` to `/dbname`. Example:

```
curl -X POST -d @sample.json http://localhost:3133/testdb
```

User-Specified Timestamps

If you perform the same post as above, but add a `ts` parameter to the URL, e.g.:

```
curl -d@/tmp/test1.json 'http://localhost:3133/t?ts=1346189075374651880'
```

the document will be stored with your user-specified timestamp. Several input formats are available. All of the following will be stored with the same key (± resolution as specified):

Nanosecond Granularity

- `2012-08-28T21:24:35.37465188Z` - RFC3339 (this is the canonical format)

- `1346189075374651880` - nanoseconds since 1970-1-1

Millisecond Granularity

- `1346189075374` - milliseconds since 1970-1-1, common in java

Second Granularity

- `1346189075` - seconds since 1970-1-1, common in unix
- `2012-08-28T21:24:35Z` - RFC3339
- `Tue, 28 Aug 2012 21:24:35 +0000` - RFC1123 + numeric timezone
- `Tue, 28 Aug 2012 21:24:35 UTC` RFC1123
- `Tue Aug 28 21:24:35 UTC 2012` - Unix date
- `Tue Aug 28 21:24:35 2012` - ansi C timestamp
- `Tue Aug 28 21:24:35 +0000 2012` - ruby datestamp

Other Timestamp Formats

These formats are also available for posting data, but are generally more useful for querying.

Minute Granularity

- `2012-08-28T21:24`

This will be considered the same as `2012-08-28T21:24:00Z`

Hour Granularity

- `2012-08-28T21`

This will be considered the same as `2012-08-28T21:00:00Z`

Day Granularity

- `2012-08-28`

This will be considered the same as `2012-08-28T00:00:00Z`

Month Granularity

- `2012-08`

This will be considered the same as `2012-08-01T00:00:00Z`

Year Granularity

- `2012`

This will be considered the same as `2012-01-01T00:00:00Z`

Querying

Querying data in seriesly is generally about determining what happened within a given time window. When formulating a query, there are a few things you need to consider:

- time range
- grouping
- keys
- aggregation

For the quickstarters, I'm going to start an example:

Example query: `http://localhost:3133/testdb/_query?from=2012&to=2013&group=3600000&ptr=/data/children/0/data/num_comments&ptr=/data/children/0/data/score&ptr=/data/children/0/data/score&reducer=avg&reducer=avg&reducer=count`

This asks to consider data from `2012` to `2013`, grouped by hour. We pull the average `/data/children/0/data/num_comments` and both the average and total `/data/children/0/data/score`. The resulting object will be in the following form:

```
{"1346050800000":[1232,2675,1001]}
```

Query Reference

Query parameters are passed in as regular URL form parameters.

Time Range

Timestamps are stored in UTC in the following format: `2012-08-27T07:52:05.151331069Z`

`from` and `to` represent starting and ending points. You can use any time format acceptable for storing samples for querying (e.g. `1346189075374651880` for nanosecond granularity epoch time or `2012` to indicate the start of the year 2012).

Grouping

`group` is the number of milliseconds over which the data are chunked together. For example, if you want the hourly aggregations, you pass `group=3600000`.

Field Selection

Field selection (the parts of the document you're interested in) is done by pairs of JSON Pointerreferences and reducer function names. You specify as many pointers as you want as `ptr` params, but each one *MUST* have a corresponding `reducer` specified.

Example:

```
ptr=/some/sub/field&reducer=avg
```

Document Filtering

Documents can be filtered on an exact string match of a field by specifying a filter key (jsonpointer) as `f` with a corresponding value of `fv` . Multiple `f` and `fv` pairs may be specified and documents will match when all filters are satisfied.

Example:

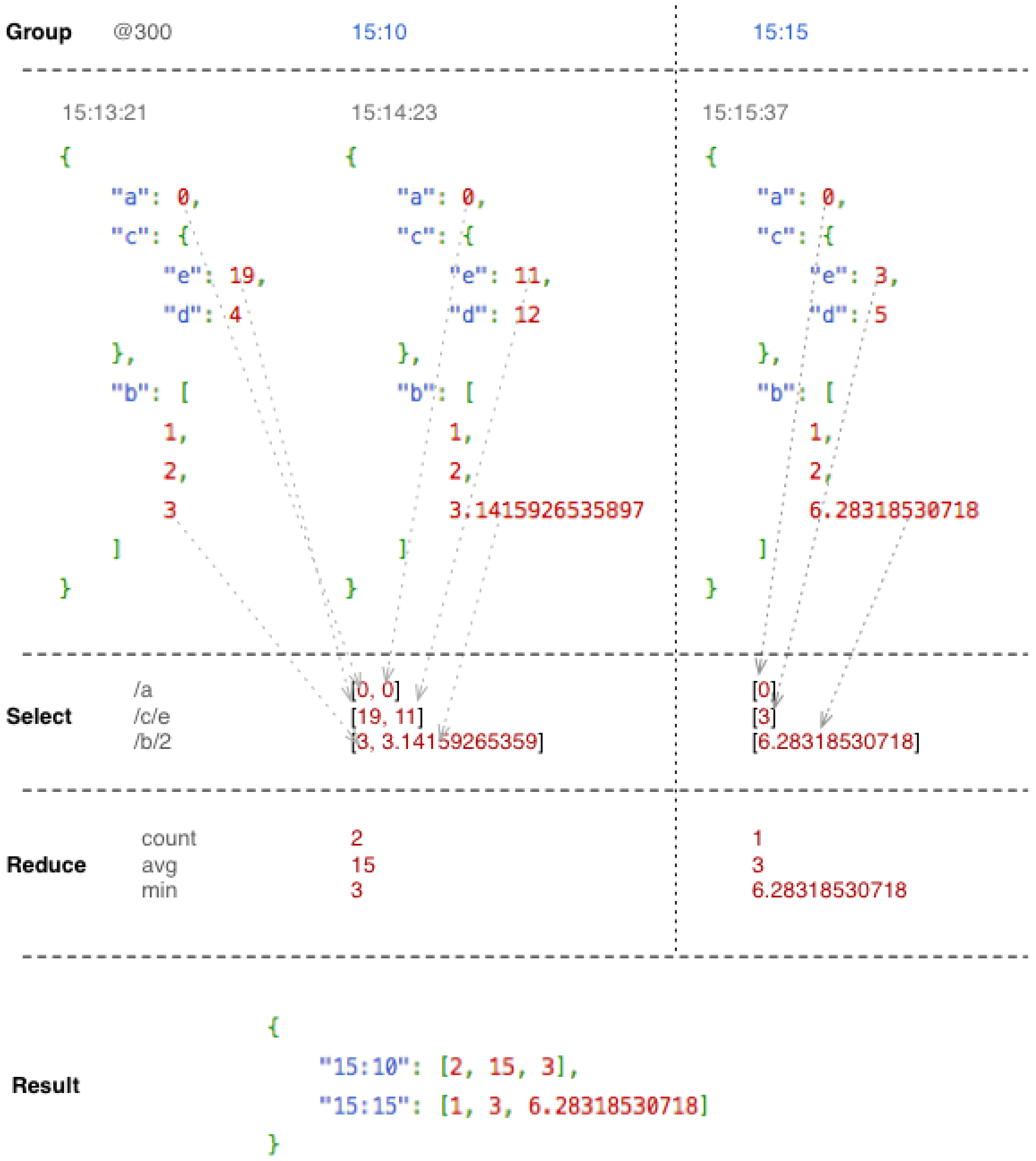
```
f=/some/field&fv=important
```

Available Reducers

- `any` - pull an arbitrary value from the group
- `count` - the number of non-null values in the group
- `sum` - the sum of numeric values from the group
- `sumsq` - the sum of the squares of the numeric values from the group
- `max` - the maximum numeric value in the group
- `min` - the minimum numeric value in the group
- `avg` - the average numeric value in the group (considering only numeric values in the count)
- `c_min` - minimum per-second rate of a counter stat
- `c_avg` - avg per-second rate of a counter stat
- `c_max` - max per-second rate of a counter stat
- `identity` - the entire group verbatim

Example

Here's a sort of visual example of how querying works. For this query, I've asked for the `count` of `/a` , the `avg` of `/c/e` and the `min` of `/b/2` grouped in 5 minute windows (`300` seconds).



The key in the result does not represent the actual key that will be emitted. I used a human-readable time representation for illustration purposes only. Had this been an actual query, the timestamps would've all been absolute and emitted as the number of milliseconds since UNIX epoch.

Retrieving Raw Docs

Individual

If you know an individual document to retrieve, you can ask for it by its single authoritative key. e.g.

```
% curl http://localhost:3133/1013A51E00000035/2005-07-10T02:38:46Z
{"r": 24.06}
```

Bulk

Although often not required, you can retrieve all of a range of docs using the `GET /dbname/_all`. This takes a `from` and a `to` parameter as described in query above.

The results come back as a single object with the key as the timestamp and the value as stored.