https://docs.influxdata.com/influxdb/v1.7/query\_language/continuous\_queries/#configuring-execution-intervals

基本语法

|  |
| --- |
| CREATE CONTINUOUS QUERY <cq\_name> ON <database\_name>  BEGIN  <cq\_query>  END |

cq\_query

|  |
| --- |
| SELECT <function[s]> INTO <destination\_measurement> FROM <measurement> [WHERE <stuff>] GROUP BY time(<interval>)[,<tag\_key[s]>] |

Schedule and coverage

Cq 操作实时数据，使用本地服务器时间标签，group by 时间长段，和influxdb 重置时间边界，来决定cq什么时候来执行，覆盖的时间范围。

Cq 执行时间间隔跟 group by time() 一样， 并且是在influxdb 重新设置的时间边界的开始出执行。比如 group by time(1h) ，cq每次会在每个小时的开始执行。

Cq 执行覆盖的时间范围 也是由group by time()指定。一般是now 到 now - group by time()

比如group by time(1h)， cq每次执行covered time range 是 now 到 now – 1h

测试数据：

|  |
| --- |
| name: bus\_data  --------------  time passengers complaints  2016-08-28T07:00:00Z 5 9  2016-08-28T07:15:00Z 8 9  2016-08-28T07:30:00Z 8 9  2016-08-28T07:45:00Z 7 9  2016-08-28T08:00:00Z 8 9  2016-08-28T08:15:00Z 15 7  2016-08-28T08:30:00Z 15 7  2016-08-28T08:45:00Z 17 7  2016-08-28T09:00:00Z 20 7 |

**自动downsample data**——就是将数据按时间段聚合一下

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_basic" ON "transportation"  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h)  END |

说明：在db transportion上面创建一个名为cq\_basic的cq。功能是，将bus\_data表中的

Passengers字段求平均值，放入average\_passengers表中。每个一个小时执行一次，每次执行覆盖的时间范围为一个小时，就是每次计算 time < now 到 time >= now – 1h之间的数据 ，结果point的time字段用时间段开始时间，就是 now – 1h。

执行效果：

|  |
| --- |
| >  At \*\*8:00\*\* `cq\_basic` executes a query with the time range `time >= '7:00' AND time < '08:00'`.  `cq\_basic` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  >  At \*\*9:00\*\* `cq\_basic` executes a query with the time range `time >= '8:00' AND time < '9:00'`.  `cq\_basic` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:00:00Z 13.75 |

执行的结果集

|  |
| --- |
| > SELECT \* FROM "average\_passengers"  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  2016-08-28T08:00:00Z 13.75 |

自动地 downsample data 并且放入其他的 retention policy

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_basic\_rp" ON "transportation"  BEGIN  SELECT mean("passengers") INTO "transportation"."three\_weeks"."average\_passengers" FROM "bus\_data" GROUP BY time(1h)  END |

说明："transportation"."three\_weeks"."average\_passengers" 这个叫fully qualify measurement

在transportation库上创建一个名为cq\_basic\_rp的cq 。功能是：每个小时执行一次，每次统计now 到 now – 1h 时间段 passengers 字段的平均值，并存入non-default rp 中，"transportation"."three\_weeks"."average\_passengers"。

执行过程：

|  |
| --- |
| >  At \*\*8:00\*\* `cq\_basic\_rp` executes a query with the time range `time >= '7:00' AND time < '8:00'`.  `cq\_basic\_rp` writes one point to the `three\_weeks` RP and the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  >  At \*\*9:00\*\* `cq\_basic\_rp` executes a query with the time range  `time >= '8:00' AND time < '9:00'`.  `cq\_basic\_rp` writes one point to the `three\_weeks` RP and the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:00:00Z 13.75 |

执行结果：

|  |
| --- |
| > SELECT \* FROM "transportation"."three\_weeks"."average\_passengers"  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  2016-08-28T08:00:00Z 13.75 |

利用backreference来downsample data 一个数据库

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_basic\_br" ON "transportation"  BEGIN  SELECT mean(\*) INTO "downsampled\_transportation"."autogen".:MEASUREMENT FROM /.\*/ GROUP BY time(30m),\*  END |

backreference 语法： :MEASUREMENT 。在目标数据库维护一个和源数据库相同的表名。

mean(\*) 语法：对所有字段进作平均值行操

group by \* 语法：可以将源数据库的tags 在目标数据库中也保存为tag类型。如果没有的话，会默认保存成field类型。

regular expression 语法：规则表达式都用 双/包起来，/1/，/temperature/。表达式是用的golang的规则表达式。规则表达式可以用在 select 、from、where、group by子句中。详细见连接<https://docs.influxdata.com/influxdb/v1.7/query_language/data_exploration/#regular-expressions>

说明：上面的cq对transportation库中的所有表所有字段求平均值，并保存到downsampled\_transportation中autogen rp对应的表中， 原表中tag在新的表中还是tag类型，每隔30m执行一次。

执行过程：

|  |
| --- |
| >  At \*\*7:30\*\*, `cq\_basic\_br` executes a query with the time range `time >= '7:00' AND time < '7:30'`.  `cq\_basic\_br` writes two points to the `bus\_data` measurement in the `downsampled\_transportation` database:  >  name: bus\_data  --------------  time mean\_complaints mean\_passengers  2016-08-28T07:00:00Z 9 6.5  >  At \*\*8:00\*\*, `cq\_basic\_br` executes a query with the time range `time >= '7:30' AND time < '8:00'`.  `cq\_basic\_br` writes two points to the `bus\_data` measurement in the `downsampled\_transportation` database:  >  name: bus\_data  --------------  time mean\_complaints mean\_passengers  2016-08-28T07:30:00Z 9 7.5  >  [...]  >  At \*\*9:00\*\*, `cq\_basic\_br` executes a query with the time range `time >= '8:30' AND time < '9:00'`.  `cq\_basic\_br` writes two points to the `bus\_data` measurement in the `downsampled\_transportation` database:  >  name: bus\_data  --------------  time mean\_complaints mean\_passengers  2016-08-28T08:30:00Z 7 16 |

结果：

|  |
| --- |
| > SELECT \* FROM "downsampled\_transportation."autogen"."bus\_data"  name: bus\_data  --------------  time mean\_complaints mean\_passengers  2016-08-28T07:00:00Z 9 6.5  2016-08-28T07:30:00Z 9 7.5  2016-08-28T08:00:00Z 8 11.5  2016-08-28T08:30:00Z 7 16 |

自动downsample data 并且配置cq的时间边界

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_basic\_offset" ON "transportation"  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h,15m)  END |

这里的15m 叫作 offset interval，它修改了cq默认的执行时间，和重置时间边界

这个cq每隔小时执行一次，offset interval 修改了默认的执行时间，这里延后15m。每次执行的时间区间是1h。

执行过程：

|  |
| --- |
| >  At \*\*8:15\*\* `cq\_basic\_offset` executes a query with the time range `time >= '7:15' AND time < '8:15'`.  `cq\_basic\_offset` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:15:00Z 7.75  >  At \*\*9:15\*\* `cq\_basic\_offset` executes a query with the time range `time >= '8:15' AND time < '9:15'`.  `cq\_basic\_offset` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:15:00Z 16.75 |

结果：

|  |
| --- |
| > SELECT \* FROM "average\_passengers"  name: average\_passengers  ------------------------  time mean  2016-08-28T07:15:00Z 7.75  2016-08-28T08:15:00Z 16.75 |

基本语法的一些常见问题

处理时间区间没有数据

CQs do not write any results for a time interval if no data fall within that time range.

基本的语法不支持fill()，有的话也会忽略。高级的语法是支持的。

Resample 以前一段时间间隔

基本cq 单词执行覆盖 now 到 now – group by time() 的一段时间

Advanced cq 可以配置query time range

回填结果为更早的数据。

CQs operate on realtime data, that is, data with timestamps that occur relative to [**now()**](https://docs.influxdata.com/influxdb/v1.7/concepts/glossary/#now). Use a basic [**INTO query**](https://docs.influxdata.com/influxdb/v1.7/query_language/data_exploration/#the-into-clause) to backfill results for data with older timestamps.

丢失tags在cq结果中

默认的，所有的into查询都会讲源数据的tag类型变成目标数据的field类型

使用 group by \* 可以保留tag 在目标数据中。

Advanced 语法

|  |
| --- |
| CREATE CONTINUOUS QUERY <cq\_name> ON <database\_name>  RESAMPLE EVERY <interval> FOR <interval>  BEGIN  <cq\_query>  END |

CQ的执行会根据服务器本地时间，resample子句中的信息和group by time()，和influxdb 服务重置时间边界一起来决定cq执行的频率和cq执行覆盖时间range。

CQ执行的频率由every 指定，覆盖的时间range 由for指定。如果没有的话，就

The RESAMPLE clause works with either or both of the EVERY and FOR intervals configured.

CQs default to the relevant [**basic syntax behavior**](https://docs.influxdata.com/influxdb/v1.7/query_language/continuous_queries/#description-of-basic-syntax) if the EVERY interval or FOR interval is not provided

时间段单位：

|  |  |
| --- | --- |
| Units | Meaning |
| ns | nanoseconds (1 billionth of a second) |
| u or µ | microseconds (1 millionth of a second) |
| ms | milliseconds (1 thousandth of a second) |
| s | second |
| m | minute |
| h | hour |
| d | day |

测试数据：

|  |
| --- |
| name: bus\_data  --------------  time passengers  2016-08-28T06:30:00Z 2  2016-08-28T06:45:00Z 4  2016-08-28T07:00:00Z 5  2016-08-28T07:15:00Z 8  2016-08-28T07:30:00Z 8  2016-08-28T07:45:00Z 7  2016-08-28T08:00:00Z 8  2016-08-28T08:15:00Z 15  2016-08-28T08:30:00Z 15  2016-08-28T08:45:00Z 17  2016-08-28T09:00:00Z 20 |

配置执行间隔时长

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_advanced\_every" ON "transportation"  RESAMPLE EVERY 30m  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h)  END |

说明：每隔30m执行一次，单词执行的时间range 是 the current time bucket 。比如one-hour time bucket （intersect）包含now，意思就是包含当前时间的 time bucket，长度为1个小时。

隔every interval分钟执行一次，时间区间为 the current time bucket ，即是group by。bucket的起始时间点为数据time。

上面的cq意思就是每隔30分钟执行一次， 时间区间为1h，整点到整点，前包括后不包括。用bucket的起始时间点为数据time

the current time bucket 的概念 ，我的理解是 8点 <= time < 9点 ，表示大小为1h的bucket， 在这个时间范围内计算都属于这个bucket， 而且用起始时间8点来表示。

比如大小为2h的bucket，应该就是 9点 <= time < 11点，在这个时间范围内计算都属于这个bucket， 而且用起始时间9点来表示。不管执行多少次，这个bucket最后一次计算数据覆盖前面的数据。

执行过程：

|  |
| --- |
| >  At \*\*8:00\*\*, `cq\_advanced\_every` executes a query with the time range `WHERE time >= '7:00' AND time < '8:00'`.  `cq\_advanced\_every` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  >  At \*\*8:30\*\*, `cq\_advanced\_every` executes a query with the time range `WHERE time >= '8:00' AND time < '9:00'`.  `cq\_advanced\_every` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:00:00Z 12.6667  >  At \*\*9:00\*\*, `cq\_advanced\_every` executes a query with the time range `WHERE time >= '8:00' AND time < '9:00'`.  `cq\_advanced\_every` writes one point to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:00:00Z 13.75 |

说明：8:30和 9：00都属于从8:00开始的一个小时时长的time bucket。都用8:00表示结果point，但是8:30执行的实际效果是：time >= 8:00 且 time <= 8:30 计算这个范围的数据。

9:00 执行的效果是：time >= 8:00 且 time < 9:00 计算这个范围的数据。

结果：

|  |
| --- |
| > SELECT \* FROM "average\_passengers"  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 7  2016-08-28T08:00:00Z 13.75 |

最终的数据还是两条， 办个小时执行的结果被 后面一个整点的结果覆盖。半个小时的时候，计算的时间区间是整点到整点，但是半点之后算将来的数据，还没有产生。所以就产生了上面的执行效果。

用for配置时间范围

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_advanced\_for" ON "transportation"  RESAMPLE FOR 1h  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(30m)  END |

Use a FOR interval in the RESAMPLE clause to specify the length of the CQ’s time range.

说明：每group by interval执行一次， 时间区间为 now 到 now - for interval ； 将时间区间分成 for/group by 段时间， 分别执行 。

上面这cq就是 每隔30分钟执行一次，时间区间 now 到 now - 1h ，将时间区间分成 1h/30m = 2，每次执行会产生2个计算结果。

假设 for 1h ， group by time(7m) 。按照上面的结论：每7分钟执行一次， 每次计算的时间的范围是1个小时， 时间范围分成 1h/7m 约为8.57 向后取整为9，就是分成9段执行那就是说，个时间点会产生9个数据point。每个数据都是7分钟的计算结果。

执行过程：

|  |
| --- |
| >  At \*\*8:00\*\* `cq\_advanced\_for` executes a query with the time range `WHERE time >= '7:00' AND time < '8:00'`.  `cq\_advanced\_for` writes two points to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 6.5  2016-08-28T07:30:00Z 7.5  >  At \*\*8:30\*\* `cq\_advanced\_for` executes a query with the time range `WHERE time >= '7:30' AND time < '8:30'`.  `cq\_advanced\_for` writes two points to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T07:30:00Z 7.5  2016-08-28T08:00:00Z 11.5  >  At \*\*9:00\*\* `cq\_advanced\_for` executes a query with the time range `WHERE time >= '8:00' AND time < '9:00'`.  `cq\_advanced\_for` writes two points to the `average\_passengers` measurement:  >  name: average\_passengers  ------------------------  time mean  2016-08-28T08:00:00Z 11.5  2016-08-28T08:30:00Z 16 |

说明：cq半个小时执行一次，每次覆盖的时间范围是1个小时，在这个时间范围内，半个小时计算一次，半个小时计算一次，这样每次执行就得到2个结果。

结果：

|  |
| --- |
| > SELECT \* FROM "average\_passengers"  name: average\_passengers  ------------------------  time mean  2016-08-28T07:00:00Z 6.5  2016-08-28T07:30:00Z 7.5  2016-08-28T08:00:00Z 11.5  2016-08-28T08:30:00Z 16 |

time相同的，后面的覆盖前面的。

For和every都配置了

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_advanced\_every\_for" ON "transportation"  RESAMPLE EVERY 1h FOR 90m  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(30m)  END |

说明：这里就不用贴数据了

上面cq 的意思是没一个小时执行一次（every表示执行频率）， 每次计算的时间范围是90m， 每次统计30分钟的数据（30分钟的数据为一组，产生一个结果集）

配置for和使用fill()

|  |
| --- |
| CREATE CONTINUOUS QUERY "cq\_advanced\_for\_fill" ON "transportation"  RESAMPLE FOR 2h  BEGIN  SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h) fill(1000)  END |

说明：

一小时执行一次，每次计算的时长是2个小时，按一个小时计算一个结果

如果在一个时间区间内，没有任何数据，就没有结果集。

如果在时间区间内， 部分group有数据， 部分group没有数据，这时没有的部分 用fill来填充。

高级语法的一些常见问题：

如果every interval 大于group by time() interval

这时cq每 every interval执行一次， time range 是now 到 now – every 而不是 now –group by

这样特意设计，是为了防止丢失数据。

如果for 小于 group by或者every

这时会报错的：

error parsing query: **FOR** duration must be >= **GROUP** **BY** time duration: must be a minimum **of** <minimum-allowable-interval> got <**user**-specified-interval>

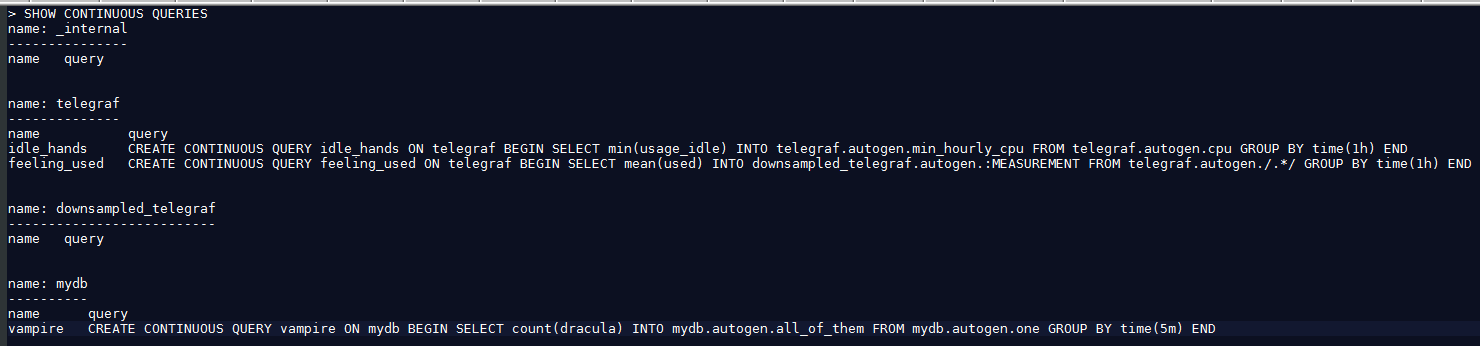
for必须大于 group by 或者every，如果指定了的话，为了保证数据不丢失。

CQ管理

列出influxdb服务上的所有CQ

>show continuous queries

会根据db分组



删除CQ

从自定数据库上面删掉CQ

|  |
| --- |
| DROP CONTINUOUS QUERY <cq\_name> ON <database\_name> |

语句返回空结果

例如：

|  |
| --- |
| > DROP CONTINUOUS QUERY "idle\_hands" ON "telegraf"  > |

修改CQ

CQs cannot be altered once they’re created. To change a CQ, you must DROP and reCREATE it with the updated settings.

CQ的使用场景

Downsample 和 数据保留

CQ和RP结合使用，可以自动的将high precision data downsample 成 lower precision data，并且删掉不必要的数据，高精度的数据。

提前计算耗时的query

CQ可以用来提前计算耗时的query，来缩短查询时间。自动的将common-queried（经常查询的）和 high precision data to lower precision，这样查询消耗的资源少，而且更快

替代 having 子句

Influxdb 不支持 having 子句的语法。可以使用CQ来聚合数据，然后查询CQ结果数据，这样来达到having相同的效果。（Influxdb支持的子查询可以达到having类似的效果）

例如：

SELECT mean("bees") FROM "farm" GROUP BY time(30m) HAVING mean("bees") > 20

想实现每30分钟计算一下bees的平均值，并查询出平均值大于20的数据。

CQ实现：

CREATE CONTINUOUS QUERY "bee\_cq" ON "mydb" BEGIN SELECT mean("bees") AS "mean\_bees" INTO "aggregate\_bees" FROM "farm" GROUP BY time(30m) END

SELECT "mean\_bees" FROM "aggregate\_bees" WHERE "mean\_bees" > 20

替代方法嵌套

InfluxQL 的一些方法支持嵌套其他方法。但是最好别那样。如果方法不支持嵌套，可以通过CQ来计算内层方法，然后查询CQ结果来计算外层方法，这样来达到相同的功能。

InfluxQL子查询可以实现方法嵌套的功能。

SELECT mean(count("bees")) FROM "farm" GROUP BY time(30m)

实现每30m计算bees的个数并计算平均值。

CREATE CONTINUOUS QUERY "bee\_cq" ON "mydb" BEGIN SELECT count("bees") AS "count\_bees" INTO "aggregate\_bees" FROM "farm" GROUP BY time(30m) END

SELECT mean("count\_bees") FROM "aggregate\_bees" WHERE time >= <start\_time> AND time <= <end\_time>

一些例子：

-- selects from DEFAULT retention policy and writes into 6\_months retention policy

CREATE CONTINUOUS QUERY "10m\_event\_count"

ON "db\_name"

BEGIN

SELECT count("value")

INTO "6\_months"."events"

FROM "events"

GROUP (10m)

END;

-- this selects from the output of one continuous query in one retention policy and outputs to another series in another retention policy

CREATE CONTINUOUS QUERY "1h\_event\_count"

ON "db\_name"

BEGIN

SELECT sum("count") as "count"

INTO "2\_years"."events"

FROM "6\_months"."events"

GROUP BY time(1h)

END;

-- this customizes the resample interval so the interval is queried every 10s and intervals are resampled until 2m after their start time

-- when resample is used, at least one of "EVERY" or "FOR" must be used

CREATE CONTINUOUS QUERY "cpu\_mean"

ON "db\_name"

RESAMPLE EVERY 10s FOR 2m

BEGIN

SELECT mean("value")

INTO "cpu\_mean"

FROM "cpu"

GROUP BY time(1m)

END;