Calculation of a customer's vCPU/CPU Usage with the use of New Relic Queries

Table of Contents

Definitions:

- **vCPU**: The virtual number of CPUs provided to the Cloud customer. It should be noted that virtual CPUs is a term used by major Cloud server providers such as Amazon Web Services and Microsoft Azure (by the time this article is written). Obviously, virtual CPUs are delivered from a set of actual/physical CPUs, allocated from the Cloud environments of the mentioned providers. Therefore, if a customer does not use Cloud resources, it then does not use virtual CPUs for the server needs, but actual CPUs. The methods described in this article can be used for both vCPUs (Cloud environments) and CPUs (local, on-premises environments). For the purpose of simplicity, this article uses the term "vCPU" for any CPU resources.
- **vCPU Time period Usage:** The number of actual vCPUs used by a customer in a given time period, multiplied by that time period. Given this definition, one example of vCPU usage is the following:

  - Given time period: 59 days
  - Provisioned vCPUs for the customer: 24 (For example: 3 servers of 8 vCPUs each)

  $$vCPU\ Time\ period\ Usage = 24 vCPUs \times 59 days = 1416\ vCPU - days$$

  Is should be mentioned that vCPU Time period usage is **different from the customer's actual CPU usage**, or simply vCPU usage on a given time period. The first one – vCPU Time period usage – measures **how many** vCPUs are used. The second one – vCPU usage – is **what portion of each vCPU** (measured either by percent % or actual CPU usage values) is used in the given time period.

- **vCPU-days Allocation:** The total amount of vCPU Time usage, measured in days, that a customer has been provided with. CPU Allocation can be described in various time units. The ones most frequently used are vCPU-days (such as the ones used in the above example) and vCPU-hours. It is a measurement of hardware-resource usage a customer is provided, for a given time period. Usually it is defined in a contract agreement. For example, Adobe Commerce by certain contract has provided 5000 vCPU-days to a Cloud customer, to use for a time period of 1 year (365 days). If this customer runs on a 3 servers x 4 vCPUs = 12 vCPUs total, assuming that this customer won't resize the server, it is estimated that it will consume 12 vCPUs x 365 days = 4380 vCPU-days in this given time period of 1 year. Therefore 5000 vCPU-days is a sufficient amount of vCPU-days allocation for this customer to smoothly run its services for the given time period.
- **Time allocation:** The (maximum) time period a given customer can be used. This is usually defined in a contract agreement. In the above example, the time allocation is 1 year. Time allocation is furtherly defined by a **starting and ending date**.

Some Clarifications:

- Conversions: vCPU-days can be converted to vCPU-hours and vice-versa, when we use the appropriate conversion factor. 12 vCPU-days convert to vCPU-hours by multiplying with the factor 24hrs/day. So this would be, for example: 12 vCPU-days converts to 12 vCPU-days x 24hrs/day = 576 vCPU-hours.

Importance:

vCPU-days Allocation defines the hardware resources a server/customer can access. From the contract starting date where a customer is in the state of "Production," certain amounts of hardware resources are used/consumed on a daily basis - or any other time unit of choice. The customer is responsible for the knowledge of that usage. The exact usage of resources should be available to both the customer's end-users as well as the company providing the computational resources, especially when a contract renewal is due.

The vCPU-days usage is perhaps the most important of all hardware resources. It's the most frequently used resource to measure consumption of a server or an individual computer application. It is also the computer resource's type most frequently purchased within contracts. Therefore, vCPU-days usage directly affects the operational costs of the customer. Because of this reason, tools and methods which quickly and correctly calculate vCPU-days usage on any required timeframes, are of utmost importance to both the customer as well as the customer's provider.

In many cases, vCPU-days usage is easily calculated. When the customer's hardware configuration in a given time period is known and unchanged, then vCPU-days usage is simply the total number of the customer's vCPUs multiplied by the time unit (For example: days) of choice of this time period. The vCPU-days calculation becomes a challenge, however, when the customer undergoes hardware configuration changes during a given time period. For example, the customer may need frequent upsizes (temporary provision of hardware configuration with stronger characteristics to tackle increased traffic/demand). The upsizes may be followed by scheduled downsizes to the original hardware configuration once the need for an upsize has ended. Additionally, the total number of instances may change, either intentionally by the customer's admins (For example: shutdown due to maintenance, permanent changes) or unintentionally when an outage occurs. In these examples, you can still calculate vCPU usage with the methods described above. However, the calculations become complex, with increased likelihood of invalid results.

This article provides methods where the customer will be able to calculate vCPU-days usage, in a variety of cases, -such as the occurrence of a resize- and for time periods of any magnitude: from a mere 1 hour to months.

The customer's scenario:

This article is reporting calculations based on the New Relic Application Performance Service utilities and UI environment. For all the New Relic queries described in this article, we are going to use a hypothetical customer with the following characteristics:

1. The customer has defined all environments (For example: the Production and Staging environments in Adobe Commerce) with a unique name. To find the customer's environment's names, you use the following query:

   ```
   SELECT uniques(apmApplicationNames) from SystemSample SINCE 20 days ago
   ```

   From the query's result, we pick the name whose environment we wish to study. In this scenario we will use the **hypothetical "prodname"** for the customer's Production environment. The time period of *20 days* used in the above example is purely empirical and can be changed appropriately by the customer. It is suggested, however, to use the appropriate time period for the customer's studies. The use of the correct name of the environment you wish to study is of utmost importance, since this name is used in all concurrent queries.
2. Let's **assume/hypothesize** that the **"prodname" starting hardware configuration consists of 3 instances**, with **16 vCPUs in each instance**. During the date of January 26, the customer undergoes a downsize (reduces the number of CPUs of each instance), from 16 to 8 vCPUs. The downsize occurs on January 26, 2021, between 13:00 – 15:00 UTC. The customer's downsize is described in Figure 1.

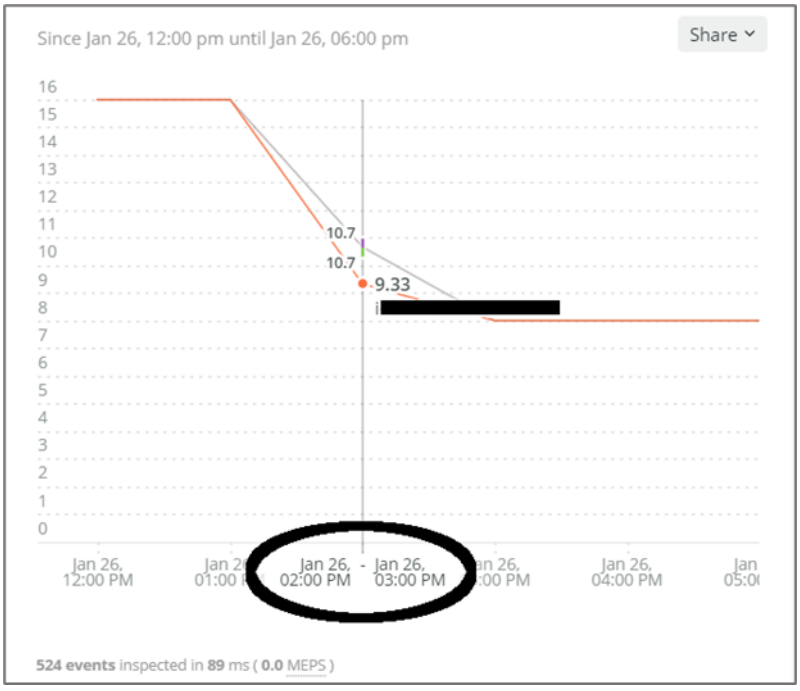


Figure 1: A customer whose environment undergoes downsizing. 3 instances downsized from 16 vCPUs (before 2:00 PM) into 8 vCPUs (after 3:00 PM). During transition (2:00 PM), the instances report to New Relic a vCPU active count of approximately 10.7, 10.7, and 9.33.

3. Before we start with the creation and description of the vCPU calculations queries, let's define the query which will be the workhorse of all our subsequent queries. The below query calculates the value (but not its usage through time) of the average vCPU used by the customer's environment named "prodname". The timeframe for the averaging is Jan 26, 2021, 12:00-13:00 (one hour):

```
SELECT average(numeric(processorCount)) as processorCount FROM
SystemSample where apmApplicationNames = 'prodname' facet
entityName SINCE '2021-01-26 12:00:00 UTC' UNTIL '2021-01-26
13:00:00 UTC'
```

The result of the above query, with the instances names covered, looks like the following:

If we use a timeframe later than 12:00-13:00 (For example: from 15:00 and after), the result would be showing the downsized average vCPU, which would be [8, 8, 8]. It is very interesting to mention the results of the query when used during the downsizing timeframe, which is 14:00 -15:00:

The above values are due to New Relic's rounding of the vCPU, which is downsized from its original value of *16* to the final value of *8*.

vCPU usage calculations:

Calculation of vCPU hours usage during 1-hour timeframe:

We wrap the workhorse query into a new one by adding a **`timeseries`** and sum it over the variable **`processorCount`**, while the timeframe is from 12:00 to 13:00. We add the keyword **`limit`** because if the total number of instances is greater than 5 (when this article is written, currently this value may have

changed), New Relic does not add them. **By increasing the limit, we make certain that New Relic adds all possible instances**. The customer can also use the term **`limit max`**, if it is desired to add all possible instances. The customers may use numeric values for the limits (For example: **`limit`** = *20*) to constrain the possible results to only 20 instances. The **`sum(processorCount)`** is the vCPU hours of the customer, in the timeframe of Jan 26, 2021, 12:00-13:00 UTC.

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
hours limit max) SINCE '2021-01-26 12:00:00 UTC' UNTIL '2021-01-26
13:00:00 UTC'
```

Result:

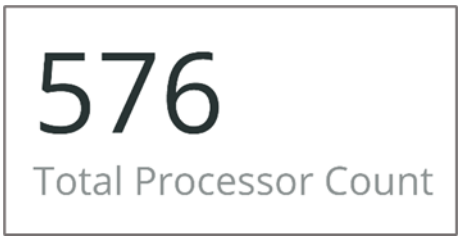
Usually, the customers are interested the value of vCPU-hours during the transition (14:00 – 15:00) timeframe. To calculate this value, we use the transition time (which we already know) into our previous query (`SINCE '2021-01-26 14:00:00 UTC' UNTIL '2021-01-26 15:00:00 UTC'`). The result is demonstrated below:

Calculation of vCPU-hours usage during many-hours timeframe:

We use the exact query mentioned on the above 1-hour timeframe example, and we simply change the timeframe into the desired one. For example, if we wish to calculate the vCPU for 12 hours, Jan 26, 2020, 00:00-12:00 UTC, we have:

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
hours limit max) SINCE '2021-01-26 00:00:00 UTC' UNTIL '2021-01-26
12:00:00 UTC'
```

Result:



This result can also be calculated as: vCPU-hours = 3 instances x 16 vCPUs x 12 hours = 576 CPU hours

If we use the above query for the entire 24-hour time period of Jan 26 (that means we use the **`SINCE '2021-01-26 00:00:00 UTC' UNTIL '2021-01-27 00:00:00 UTC'`** timeframe) we will also include the downsize transition. The result is the below:

This value can also be calculated by hand as follows: From Figure 1, we see the entire CPU profile. We observe that during the 24-hour time period, the environment is on a 3x16 configuration for 14 hours, downsizes into a 3x8 configuration for 9 hours, and finally has 1-hour transition where the instances have vCPU values of (see previous results) 11.4, 10.3, and 9.36, respectively. So the vCPU hours for this environment for the entire day is 3 instances x 16 vCPUs x 14 hours + 11.4 + 10.3 + 9.36 + 3 instances x 8 CPUs x 9 hours = 919.06 vCPU-hours.

Calculation of vCPU-hours usage during many-days timeframe

This is the most important case, because the customer would like to calculate vCPU hours on a large time period, like for example, 1 month or even an entire year. For this example, we have to change our approach to build the appropriate query.

The basis then becomes a **`timeseries`** sample on a *1-day* timeframe, instead of *1-hour*. The reason for that is the **`timeseries`** keyword in New Relic Queries can be used with up to 366 time-buckets. The term **Buckets** is what New Relic calls its sampling number of points for the query result. So, when we use a **`timeseries`** of *1-hour*, we can sample roughly 366 hours, which is approximately 15 days. If we use a study time period of more than 15-16 days (the **`SINCE`** keywords in the queries) on the previous examples, New Relic will produce a 500 error like the one below:

To avoid this error, we switch the **`timeseries`** to *1-day* instead of *1-hour*. With this switch, we will be able to sample with the **`timeseries`** approximately 366 days (roughly 1 year), instead of 2 weeks.

If we want to calculate CPU days allocation, we keep the variable **`sum(processorCount)`** as is. But if we want to calculate CPU hours as we did on all the previous examples, we multiply **`sum(processorCount)`** with the conversion factor which is number of hours per day, which is *24*. In this way, we convert the CPU days which are actually calculated, into vCPU hours, as noted in the clarifications section of this article. First, let's see what the result looks like if we use only a 1-day timeframe on the query we mentioned in this example:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit max) SINCE '2021-01-27 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

The **bold section** on the above query is emphasis to draw the attention to the reader. This value of *24* is the conversion factor to convert vCPU-days into vCPU-hours.

Result:

For two days, from Jan 26, 2020, 00:00 – Jan 28, 2020:
Result:

Finally, we can use a long time period, and see how much vCPU-hours have been used. The below example is for 28 days, Jan 1 to Jan 28, 2020:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit max) SINCE '2021-01-01 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

And the result is shown below:

Hint: If we want to get the vCPU hours value without rounding approximation, we can switch the type of report from the **Chart type** drop-down menu, from **Billboard** to **JSON**. Also, note that in the below example we use **`limit`** = *20* instead of *max*, to reduce the size of the JSON outcome (We may obviously use the value that suits us best.). Then we can view the exact value:

Calculation of vCPU-days usage during many-days timeframe

The same query, when used to calculate vCPU days (instead of vCPU hours) is the following:

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit max) SINCE '2021-01-01 00:00:00 UTC' UNTIL '2021-01-28
00:00:00 UTC'
```

In this example, we demonstrate how the results values change **when we request vCPU-days rather than vCPU-hours**. The query is identical to the queries of the previous examples, with the only difference that we do not multiply **`sum(processorCount)`** with the factor of 24 hours/day. The result is demonstrated below:

Chart of vCPU-hours usage during many-days timeframe

Usually, the customers as well as the in-house Adobe groups wish to calculate the total value of vCPU-days, as described in the above chapters. However it is very useful for the customers to create (and therefore study) a chart of the vCPU-hours per day an environment has used. We can use the previous query and include an additional **`timeseries`** keyword (emphasized in **bold**), to produce a chart. The final query looks like the one below:

```
SELECT sum(processorCount)*24 FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit max) timeseries 1 day SINCE '2021-01-01 00:00:00 UTC' UNTIL
'2021-02-01 00:00:00 UTC'
```

Notice that the above query requires two **`timeseries`** keywords, to create the desired graph. The original **`timeseries`** is highlighted in yellow, and the second one required to create the graph is in **bold**.

Result: The chart is demonstrated below. The customer can check the value of vCPU-hours used for any particular date by hovering their mouse over the desired date. Note that on the transition dates (the dates where a resize is taking place), the values demonstrated in the chart may not be as accurate relative to the total sum calculated with the previous queries. This is due to different rounding when New Relic creates charts.

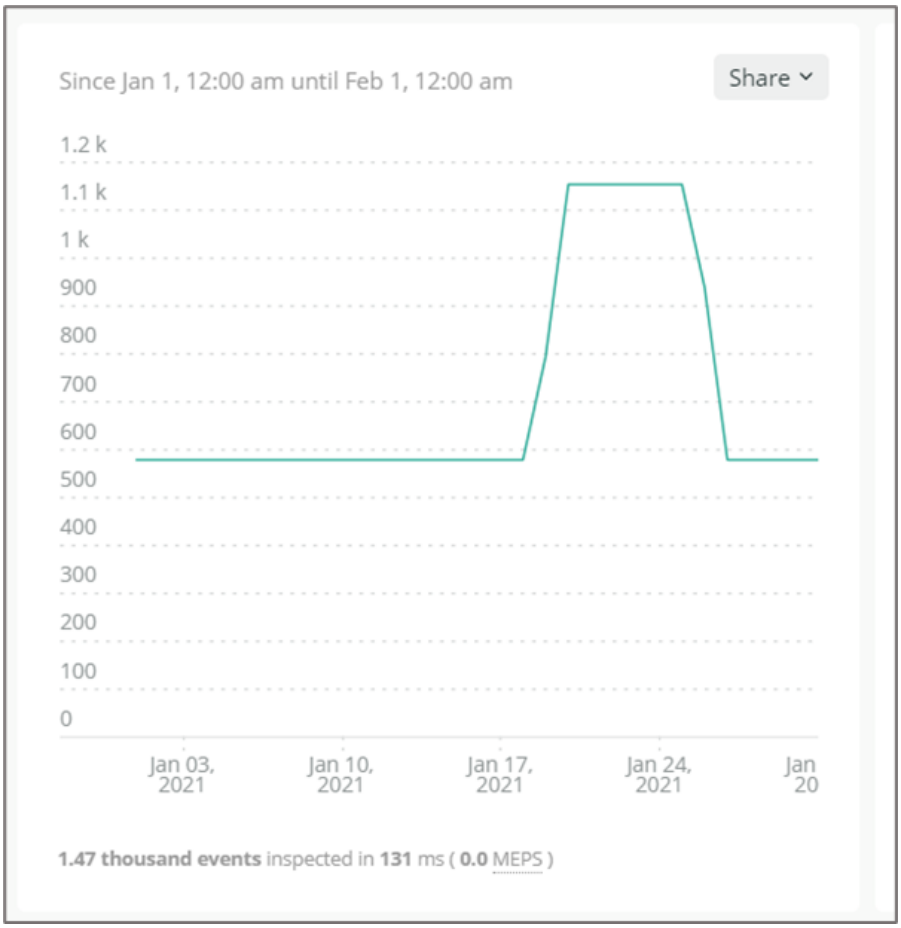
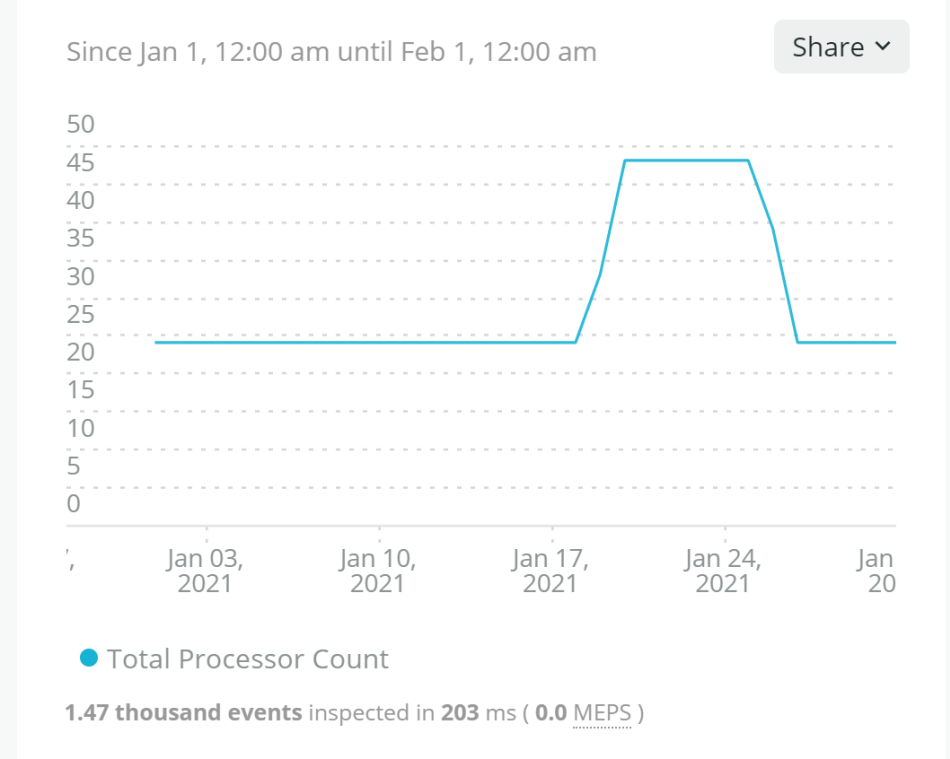


Chart of vCPU-days usage during many-days timeframe

The same query used to create the vCPU hours chart, can be used to calculate and create the equivalent vCPU days chart. We only need to **remove the 24hrs/day factor**.

```
SELECT sum(processorCount) FROM (SELECT
average(numeric(processorCount)) as processorCount FROM SystemSample
where apmApplicationNames = 'prodname' facet entityName timeseries 1
day limit max) timeseries 1 day SINCE '2021-01-01 00:00:00 UTC' UNTIL
'2021-02-01 00:00:00 UTC'
```

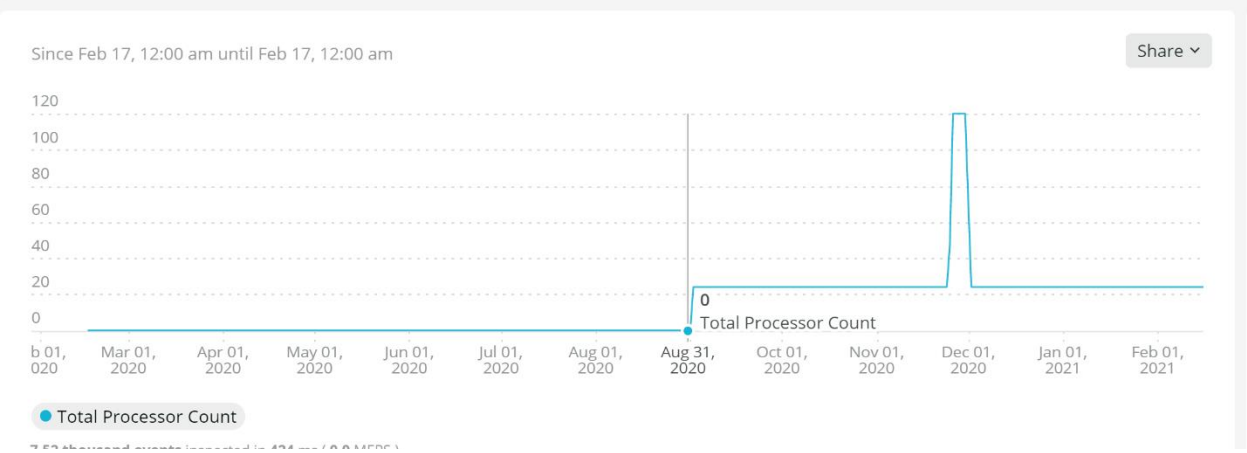
And the result is demonstrated below:



Note: The reader may observe that the two above graphs are identical by means of the line shape. Only the "plateau" values (all three) change. This result is because the two graphs defer only by the conversion factor of 24 hours/day. Usually, the customers are interesting in the vCPU-days value, rather than the vCPU-hours value.

Manual Calculation of vCPU-days Chart

All the above chapters have described cases where the Application Performance Suit (in this case New Relic) has been adequately installed, and the environments (defined by the keyword **`apmApplicationNames`**) contain the appropriate instances. There are many cases however, where New Relic environments do not tag the appropriate instances correctly into the relevant environments. There are many reasons why this phenomenon occurs, however it is not described in this article. Suffice it to say that when the server instances are not properly tagged, the query when using the keyword **`apmApplicationNames`** and the appropriate environment name, will produce an incorrect result. The chart below demonstrates such a case:



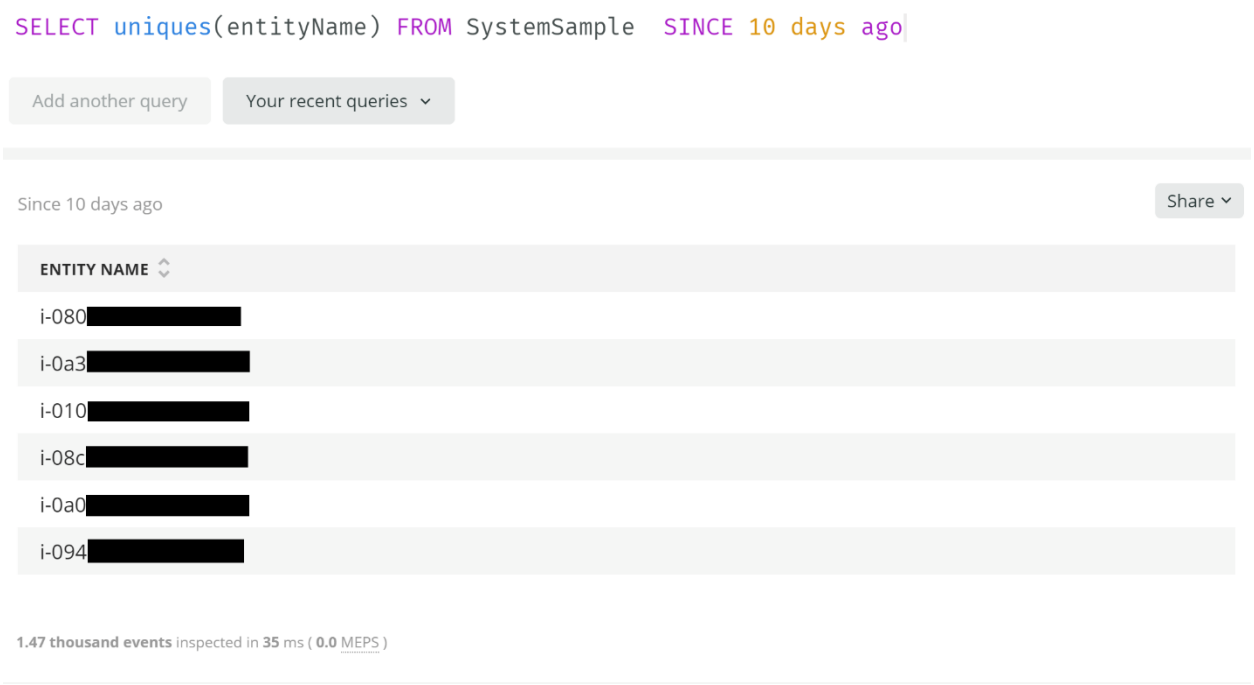
From the above chart we see that from the starting date of Feb 17, 2020, until the end of August (Aug 31 in the above graph), the vCPU-days value calculated is a horizontal line with the value of zero. Then at the beginning of September, the chart demonstrates some calculations and an upsize. It completes the calculations until the last requested day which is Feb 17, 2021. This chart demonstrates a typical case where not all instances are tagged into the environment of study, hence the "zero" value for a large portion of the studied time period.

The only way we can avoid such incorrect calculations is to manually find which instances belong to the environment of study (For example: the site's Production), and then create a New Relic query which specifically targets these instances and calculates vCPU. The steps for this approach are as follows:

1. **Find all instances of the site**. To accomplish this, you can use the following query:

   ```
   SELECT uniques(entityName) FROM SystemSample  SINCE 10 days ago
   ```

   Again the timeframe of 10 days is purely empirical and up to the customer. The result will look similar to the below image:



2. **Manually find which of the instances belong to the environment that the customer wishes to study**. There are many different ways to accomplish this, and it is not in this article's goal to explore them. From the example above, let's assume that the instances belonging to Production are the following: i-080, i-010, and i-094.

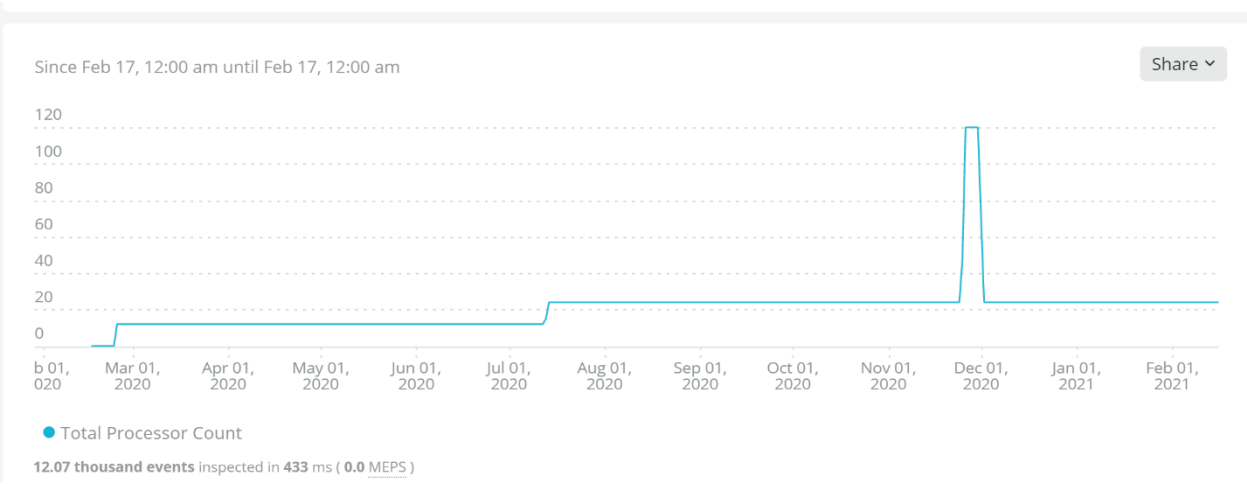
3. Once the customer **knows**, or otherwise **has decided**, which instances to include in the study, **they may use the below query**:

   ```
   SELECT sum(processorCount) FROM (SELECT
   average(numeric(processorCount)) as processorCount FROM
   ```

```
SystemSample facet `entityName` where (`entityName` in ('i-
080','i-010','i-094') ) timeseries 1 day ) timeseries 1 day SINCE
'2020-02-17 00:00:00 UTC' UNTIL '2021-02-17 00:00:00 UTC'
```

In the above query, we use the keyword **`entityName`** which targets the instances as a **`facet`**, which we constrain by using the **`where`** expression and the set of instances we have found on our earlier step. The outcome of the constraint is that we receive only the three values of i-080, i-010, and i-094. Note that the keyword **`entityName`** must be used exactly as indicated on the above query, including the quotes. Any other keyword or variable name of the customer's choice will lead to "4XX" New Relic errors, or entirely incorrect results.

The result of the above query targets the same site and the same timeframe of the chart demonstrated at the beginning of this chapter. The revised query result is demonstrated on the chart below:



From the above chart, we observe that New Relic correctly depicts the time period of Feb 17, 2020, up to mid-July, 2020, with a vCPU value of *12*. Then, the chart describes an upsize in approximately mid-July, 2020, to a value of *24*, another brief upsize to a value of *120*, and finally a downsize back to the value of *24* for the rest of the time period. The only **zero** values observed are at the very beginning of the chart, where we reach the threshold of New Relic's storage range.

Finally, note that to calculate the total vCPU-days of the above 1-year time period (not the graph), you only need to omit the second **`timeseries`** keyword, defined in **bold**.

The above method and corresponding queries are, unfortunately, **the only queries that truly guarantee a correct calculation of a vCPU-days graphs and a total vCPU-days usage**. Most of the time, the New Relic **`apmApplicationNames`** contains all relevant instances for the vCPU-days calculation. However, if the customer has the suspicion that the variable, for whatever reason, has not been defined properly, they should use only the Manual method, briefly described in this last chapter.