Technical white paper

AQL Developer Guide

Operations Analytics 2.20

Document Release Date: December 2014

Software Release Date: May 2015

Contents

What is AQL

The primary objective of the Analytics Query Language (AQL) is to simplify your ad hoc query experience. This applies to the process of building custom dashboards as well as troubleshooting problems using statistical algorithms.

AQL is a hierarchical language that provides layers of abstraction on analytic queries. The idea here is that the more abstract, the easier it is for you to write AQL in an ad hoc fashion. The layers of abstraction in AQL are:

- Built-in analytics are defined as functions that become intrinsic in AQL
- A query language to provide SQL-like access to all collections
- Functions and expressions as abstractions of queries

As an example of the layers of abstraction consider the following query:

```
from i in (oa_sysperf_global)
let interval=300
let analytic_interval=between($starttime,$endtime)
where (i.host_name like "myhost")
select  moving_avg(i.cpu_util)
```

This query assumes a collection of system metrics (`oa_sysperf_global`) and will calculate a time series of the moving average of the CPU utilization for the system call "myhost". The time series data is every 300 seconds (5 minutes) and the time range is specified by the internal macros `$starttime` and `$endtime`. Note that 'moving_avg' is a built-in analytic that significantly simplifies this transformation over standard SQL.

It is clear that this query pattern is quite useful for all sorts of metrics. Suppose you have other metrics and other functions and want to calculate the time series of a particular metric using a particular function r for a particular host or hosts. You would clearly use a query pattern as above.

Operations Analytics AQL allows useful query patterns to be abstracted into AQL functions. Using the above example, suppose you want to generalize the query to generate a time series of any metric in `oa_sysperf_global` using any function for any set of hosts. Then you can define an AQL function as follows:

```
/* Returns the moving analytic of a specific HP Operations Agent metric
by host. Input parameters are the host filter, metric name, and moving
analytic function name. */
define oaSysperfMovingMetric(hostFilter, metric, function) =
    from i in (oa_sysperf_global)
    let analytic_interval = between($starttime,$endtime)
    let interval = $interval
    where i.host_name like hostFilter
    group by i.host_name
    select function(i.metric)
```

With this function defined, the following AQL expression:

`[oaSysperfMovingMetric("myhost", cpu_util, moving_avg)]`

is identical to the above AQL query.

Furthermore the following expression:

```
[oaSysperfMovingMetric("myhost", swap_util, moving_max)]
```
Would give you the time series of the moving maximum swap utilization on host "myhost".

Clearly the ability to define apecialized AQL provides a significant 'ease-of-use' factor in using Operations Analytics to do ad hoc analytics. As further examples, Operations Analytics includes several packages of useful AQL functions that can be seen by using the Operations Analytics console.

Analytics Query Language Syntax, Intrinsics and Examples

AQL Syntax

The basic structure of an AQL query is very similar to the standard 'Structured Query Language'. An AQL query is a sequence of clauses. The clauses you include depend on the type, organization, and order of the information you want Operations Analtyics to return. It also depends on the time range and type of analysis you want Operations Analytics to apply to the data.

The types of clauses supported by AQL are as follows:

- `from <row variable> in <collection>`
- `where <relational expression>`
- `let <name> = <value>`
- `order <expression> <ascending | descending>`
- `group by <list of columns>`
- `select <select expression>`

When positioning the clauses in an AQL query, note the following:

1. The `from` and `select` clauses are mandatory. The `from` clause must be the first clause and the `select` clause must be the last clause in the query.
2. All other clauses in the query may be in any order between the `from` and the `select` clauses. The following clauses filter and group the identified collection of metrics and attributes.

From Clause

The `from` clause defines the row variable and specifies the collection from which the rows will be selected. For example:

```
from i in (oa_sysperf_global)
```

defines the row variable to be `i` and the collection (table) to select from as `oa_sysperf_global`.

Where Clause

The `where` clause is any arbitrary relational expression. The `where` clause specifies the criteria for which rows are selected from the collection. The following is an example:

```
where (( i.hostinfo_dnsname like "myhost")
        && (( i.severity ilike "CRITICAL" )||(i.severity ilike "WARNING")
))
```

This `where` clause is restricting the selected rows to be only those events for host "`myhost`" with severity of either `CRITICAL` or `WARNING`.

Let Clause

The `let` clause is used to define a value for a specific control variable for the query. For example, to control the time interval of the query, use the `let` clause to define a value for the global control variable `analytic_interval` (for example, `analytic_interval=between($starttime, $endtime)` is where `$starttime` and `$endtime` are UI parameters.

The `let` clause can also be used to override dashboard pane parameters. For example it can override the limit setting that controls the number of results. The default for `Limit` is 100 and `let Limit = 50` would override the `Limit` dashboard pane parameter that is set to to return just 50 results.

Group By

The `group by` clause organizes the results in the query based on the column or columns specified in the `group by` clause. For example `group by i.hostname` displays the results of the query in distinct groups by the host name attribute.

You can specify multiple columns in the `group by` clause, meaning the results will be organized primarily by the first column then by the second column, and so forth.

Order Clause

The `order` clause sorts the results in the query based on the expression specified in the `order` clause. For example `order i.utilization ascending` displays the results of the query sorted by the utilization column in ascending order. Order can also sort by `descending` by using that keyword following the order expression.

You can specify multiple columns in the `group by` clause, meaning the results will be organized primarily by the first column then by the second column, and so forth.

Select Clause

The `select` clause explicitly specifies the values to be selected for the query results. If you specify just the row variable, all columns are selected by the query.

Examples: `select i`
Selects all columns in the table.

```
select i.hostname, i.timestamp, i.state, i.category, i.title
```
Selects only the `hostname`, `timestamp`, `state`, `category`, and `title` attributes from the table.

You can specify multiple columns in the `group by` clause, meaning the results will be organized primarily by the first column then by the second column, and so forth.

Intrinsic Statistical Functions in AQL

Operations Analytics provides a set of analytic functions to analyze the metrics, topology, inventory, event, and log file data that it collects.

Overall Aggregate (Summary) Functions Provided by Operations Analytics

The following table shows descriptions of the overall aggregate (summary) analytic functions provided by Operations Analytics.

Analytic Function Type	Description
`aggregate_avg`	Identifies the average value for the metric or metrics selected.
`aggregate_min`	Identifies the minimum value for the metric or metrics selected.

aggregate_max	Identifies the maximum value for the metric or metrics selected.
aggregate_total	Identifies the total value or cumulative sum for the metric or metrics selected.
aggregate_count	Computes the total count of rows with values of an attribute or total count of all rows in a collection table.
aggregate_distinct_count	Computes the total count of distinct values of an attribute.

Moving Aggregates (Time Series) Functions Provided by Operations Analytics

See the following table for descriptions of the moving aggregate (time series) functions provided by Operations Analytics.

Function	Description
moving_avg	Computes the average values at each time interval within the specified time window for one or more metrics.
moving_min	Computes the minimum values at each time interval within the specified time window for one or more metrics.
moving_max	Computes the maximum values at each time interval within the specified time window for one or more metrics.
moving_total	Computes the totals at each time interval within the specified time window for one or more metrics.
moving_count	Computes the total counts of rows with values of an attribute or total count of all rows within a collection table at each time interval within the specified time window.
moving_distinct_count	Computes the total counts of distinct values of an attribute at each time interval within the specified time window.

Analytic Statistical Functions applied to Overall Aggregate and Moving Aggregate Functions

The following table describes the analytic statistical functions provided by Operations Analytics.

bottomN	Computes the lowest N values in the expressions; returns the bottomN values with their associated rank.
inverse_pctile	Calculates the inverse percentile distribution values for the set of values in the expression. For example, if you specify 50 as the <pctile> value, inverse_pctile finds the 50th percentile value (or median value) for the data in the expression.
pctile	Calculates the percentile rank value for the values in the expressions

	For example, if you specify 75 as the <pctile> value, pctile returns all values greater than the 75th percentile value for the data in the expression.
rank	Calculates the overall rank for all values in the expression, where the results include an integer (indiciating rank) for each value along with the value itself.
topN	Uses the rank (descending order) analytic function to identify the highest N values. Operations Analytics returns the top N values with their associated rank. Note the following: • If you do not specify an N value in the AQL query, Operations Analytics displays the top five values. • The topN analytic function is not permitted in the where clause.

AQL Query Examples

Return the average CPU utilization and CPU run queue size

The following AQL query returns the average CPU utilization and CPU run queue size for each host matching the filter criteria.

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where (i.host_name like "*.mydomain.com") group by i.host_name
select aggregate_avg(i.cpu_util), aggregate_avg(i.cpu_run_queue)
```

Return the average for each of the metrics collected by the oa_sysperf_global collection

The following AQL query returns the average for each of the metrics collected by the oa_sysperf_global collection for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where (i.host_name like "*.mydomain.com") group by i.host_name
select aggregate_avg(i)
```

Return the maximum, minimum, and average values for CPU utilization and CPU run queue size

The following AQL query returns the maximum, minimum, average for CPU utilization and CPU run queue size for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where (i.host_name like "*.mydomain.com") group by i.host_name
select aggregate_min(i.cpu_util), aggregate_max(i.cpu_util),
aggregate_max(i.cpu_util),
aggregate_min(i.cpu_run_queue), aggregate_max(i.cpu_run_queue),
aggregate_avg(i.cpu_run_queue)
```

Return the minimum, maximum, and average for each of the metrics collected by the oa_sysperf_global collection

The following AQL query returns the minimum, maximum and average for each of the metrics collected by the `oa_sysperf_global` collection for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where (i.host_name like "*.mydomain.com") group by i.host_name
select aggregate_min(i), aggregate_max(i), aggregate_avg(i)
```

Return Summary Information on Events (Example AQL Queries)

Note: Each of the examples queries data from the `omi_events_omievents` collection. This collection uses HP Operations Manager i (OMi) to collect OMi events. Each example queries data for only the hosts in the `mydomain.com` domain.

Return the total count of OMi events for a specified host and severity combination

The following AQL query calculates the total count of OMi events for each host and severity combinations matching the filter criteria:

```
from i in (omi_events_omievents)
let analytic_interval= between($starttime, $endtime)
where ( ( i.hostinfo_dnsname like "*mydomain.com" ) && ( ( i.severity
ilike "CRITI*" ) || (i.severity
ilike "WARN*") ))
group by i.hostinfo_dnsname, i.severity select aggregate_count(i)
```

Return the total count of OMi events for a specified host and severity combination and for which the event count exceeds 100

The following AQL query does the same as the previous AQL query, but returns the counts for only those host name and severity combinations for which the event count exceeds 100

```
from i in (omi_events_omievents)
let analytic_interval= between($starttime, $endtime)
where (( i.hostinfo_dnsname like "*mydomain.com" ) && ( ( i.severity ilike
"CRITI*" ) || (i.severity
ilike "WARN*") ) && ( aggregate_count(i) > 100 ))
group by i.hostinfo_dnsname, i.severity select aggregate_count(i)
```

Return the number of distinct applications monitored by HP Business Process Monitor (BPM) per location

Note: The following AQL query uses the `bpm_application_performance` collection. This collection uses HP Business Process Monitor (BPM) to gather application performance information.

The following AQL query calculates the number of distinct applications monitored by BPM on a location by location basis.

```
from i in (bpm_application_performance)
let analytic_interval = between($starttime, $endtime)
group by i.location
select aggregate_distinct_count(i.application)
```

Return the total count of distinct database instances reporting oracle metrics

Note: The following AQL query uses the `oa_oraperf_graph` collection. The `oa_oraperf_graph` collection uses HP Operations Smart Plug-in for Oracle to gather Oracle performance information.

The following AQL query returns a distinct counts of database instances reporting oracle metrics:

```
from i in (oa_oraperf_graph)
let analytic_interval= between($starttime,$endtime)
where ( i.host_name like "*mydomain.com" )
group by i.host_name select aggregate_distinct_count(i.db_instance_name)
```

Return the moving average CPU utilization and CPU run queue size

Note: Each of the examples queries data from the `oa_sysperf_global` collection. This collection uses HP Performance Agent to collect system metrics. Each example queries data for only the hosts in the mydomain.com domain.

The following AQL query returns the moving average CPU utilization and CPU run queue size for each host matching the filter criteria.

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime) let interval=$interval
where (i.host_name like "*.mydomain.com") group by i.host_name
select moving_avg(i.cpu_util), moving_avg(i.cpu_run_queue)
```

Return the moving average for each of the metrics collected by the oa_sysperf_global collection

The following AQL query returns the moving average for each of the metrics collected by the `oa_sysperf_global` collection for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime) let interval=$interval
where (i.host_name like "*.mydomain.com") group by i.host_name
select moving_avg(i)
```

Return the moving maximum, minimum, and average values for CPU utilization and CPU run queue size

The following AQL query returns the moving maximum, minimum, and average for CPU utilization and CPU run queue size for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime) let interval=$interval
where (i.host_name like "*.mydomain.com") group by i.host_name
select moving_min(i.cpu_util), moving_max(i.cpu_util),
moving_max(i.cpu_util), moving_min
(i.cpu_run_queue), moving_max(i.cpu_run_queue),
moving_avg(i.cpu_run_queue)
```

Return the moving minimum, maximum, and average for each of the metrics collected by the oa_sysperf_global collection

The following AQL query returns the moving minimum, maximum and average for each of the metrics collected by the `oa_sysperf_global` collection for each host matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime) let interval=$interval
where (i.host_name like "*.mydomain.com") group by i.host_name
select moving_min(i), moving_max(i), moving_avg(i)
```

Note: Each of the following examples queries data from the `omi_events_omievents` collection. This collection uses HP Operations Manager i (OMi) to collect OMi events. Each example queries data for only the hosts in the mydomain.com domain.

Return the moving total count of OMi events for a specified host and severity combination

The following AQL query calculates the moving total count of OMi events for each host and severity combinations matching the filter criteria:

```
from i in (omi_events_omievents)
let analytic_interval=between($starttime,$endtime) let interval=$interval
where (( i.hostinfo_dnsname like "*mydomain.com" ) && ( ( i.severity ilike
"CRITI*" ) || (i.severity
ilike "WARN*") ))
group by i.hostinfo_dnsname, i.severity select moving_count(i)
```

Return the moving total count of OMi events for a specified host and severity combination and for which the event count exceeds 100

The following AQL query does the same as the previous AQL query, but returns the moving counts for only those host name and severity combinations at only those intervals at which the event count exceeds 100:

```
from i in (omi_events_omievents)
let analytic_interval=between($starttime,$endtime) let interval=$interval
where (( i.hostinfo_dnsname like "*mydomain.com" ) && ( ( i.severity ilike
"CRITI*" ) || (i.severity
ilike "WARN*") ) && ( moving_count(i) > 100 ))
group by i.hostinfo_dnsname, i.severity select moving_count(i)
```

Return the moving number of distinct applications monitored by HP Business Process Monitor (BPM) per location.

Note: The following AQL query uses the `bpm_application_performance` collection. This collection uses HP Business Process Monitor (BPM) to gather application performance information**.**

The following AQL query calculates the moving number of distinct applications monitored by BPM on a location by location basis.

```
from i in (bpm_application_performance)
let analytic_interval = between($starttime, $endtime) let interval =
$interval
group by i.location
select moving_distinct_count(i.application)
```

Return the moving total count of distinct database instances reporting Oracle metrics.

Note: The following AQL query uses the `oa_oraperf_graph` collection. The `oa_oraperf_graph` collection uses HP Operations Smart Plug-in for Oracle to gather Oracle performance information.

The following AQL query returns moving total counts of the distinct database instances reporting Oracle metrics:

```
from i in (oa_oraperf_graph)
let analytic_interval= between($starttime,$endtime) let interval =
$interval where ( i.host_name like"*mydomain.com" )
group by i.host_name
select moving_distinct_count(i.db_instance_name)
```

Returns the percentile distribution of overall cpu utilization by host

The following AQL query determines the hosts and their overall aggregate average values of CPU utilization along with the percentile rank for the value among the overall aggregate average values for all hosts matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where( i.host_name like "*.mydomain.com" )
group by i.host_name select pctile(aggregate_avg(i.cpu_util) )
```

Note: The following example queries data from the `omi_events_omievents` collection. This collection uses HP Operations Manager i (OMi) to collect OMi events. Each example queries data for only the hosts in the mydomain.com domain.

Returns the percentile distribution of event count by host

The following AQL query determines the hosts and their overall aggregate count of events along with percentile ranks of the overall aggregate event count values for all hosts matching the filter criteria:

```
from i in (omi_events_omievents)
let analytic_interval= between($starttime,$endtime)
where( i.hostinfo_dnsname like "*.mydomain.com" )
group by i.hostinfo_dnsname
select pctile(aggregate_count(i))
```

Note: The following example queries data from the `bpm_application_performance` collection. This collection uses HP Business Process Monitor (BPM) to gather application performance information.

Return the Top N Values (Example AQL Queries)

Tip: Also use these examples to assist you in constructing AQL queries that use the `bottomN` analytic function.

The following examples use the `topN` analytic function to return the top n values for sets of data returned by the overall aggregate and moving aggregate analytic functions.

Note: The following examples query data from the `oa_sysperf_global` collection. This collection uses HP Operations Agent to collect system metrics. Each example queries data for only the hosts in the mydomain.com domain.

Return the top five hosts and their overall aggregate average values of CPU utilization. This query also returns the associated relative ranks.

The following AQL query determines the top five hosts and their overall aggregate average values of CPU utilization among the overall aggregate average values and relative ranks for all hosts matching the filter criteria:

```
from i in (oa_sysperf_global)
let analytic_interval= between($starttime,$endtime)
where ( i.host_name like "*.mydomain.com" )
group by i.host_name
select topN(aggregate_avg(i.cpu_util),5 )
```

Return the top 10 hosts with the highest overall aggregate count of events

The following AQL query determines the top 10 hosts with the highest overall aggregate count of events among the overall aggregate event count values for all hosts matching the filter criteria:

```
from i in (omi_events_omievents)
let analytic_interval= between($starttime,$endtime)
where ( i.hostinfo_dnsname like "*.mydomain.com" )
group by i.hostinfo_dnsname
select topN(aggregate_count(i), 10 )
```

Analytics Query Language Functions and Expressions

You have seen in the first section, Analytics Query Language Syntax, Intrinsics and Examples, how to define an AQL function. This section explains how AQL functions are used and the style of AQL functions that are expected to be written.

Collection-Specific AQL Functions

Operations Analytics has a notion of a content package which is how additional functionality is added to the product. Basically, a content pack consists of the following:

- A collection or collections.
- AQL functions to analyze the new collection(s).
- Dashboards to present the analytics for the new collection or collections.

AQL functions that are specific to a particular collection are usually specialized to certain types of metrics and analytics. They provide a very easy way for the user to ad –hoc analysis on the data. The following examples show Oracle-specific AQL funtions.

```
/* Returns the top N of an aggregate analytic on an HP Operations Oracle
SPI metric. Input parameters are the host filter, database instance
filter, metric name, aggregate analytic, and N. */

define oaOraperfTopNAggregateMetric

(hostFilter,instanceFilter,metric,aggregate_analytic,N)  =

from i in (oa_oraperf_graph)

let analytic_interval = between($starttime, $endtime)

let interval=$interval

let aggregate_playback=$aggregate_playback_flag

where ( ( i.host_name like hostFilter ) && ( i.db_instance_name like
instanceFilter ) )

group by i.host_name, i.db_instance_name

select topN(aggregate_analytic(i.metric), N);


/* Returns the aggregate analytic above a specified threshold on an HP
Operations Oracle SPI metric. Input parameters are the host filter,
database instance filter, metric name, aggregate analytic, and threshold
percentage. */

define oaOraperfAggregateMetricAbovePctile
(hostFilter,instanceFilter,metric,aggregate_analytic,upper_limit_pctile) =

from i in (oa_oraperf_graph)

let analytic_interval = between($starttime, $endtime)

let interval=$interval let aggregate_playback=$aggregate_playback_flag

where ( ( ( i.host_name like hostFilter ) && ( i.db_instance_name like
instanceFilter ) ) && ( aggregate_analytic(i.metric) >
inverse_pctile(aggregate_analytic(i.metric), upper_limit_pctile ) ) )

group by i.host_name, i.db_instance_name

select aggregate_analytic(i.metric);
```

Generic AQL functions

Generic functions are more generalized and can be used on any type of collection. There are two primary generic functions:

- `metricQuery`
- `attributeQuery`

The generic functions are mostly a template or shorthand for composing a complete query. These functions are used by PQL in the process of generating the AQL for dashboard panes. Because of their succinctness they are also frequently used in the out-of-box dashboards.

`metricQuery` takes four parameters using the following syntax:

```
metricQuery(<table name>, {<where clause>), {<group by>}, {<select>})
```

`metricQuery` (as the name suggests) is intended as a generalized approach to formulate a query on metrics that yields either time series metric data or aggregated metric data. Note the '{' delimeters are used instead of normal '(' to group the clauses. An example AQL expression use of `metricQuery is:`

```
[metricQuery(oa_sysperf_global, {(i.host_name ilike "*")}, {i.host_name},
{moving_avg(i.active_processes), moving_avg(i.cpu_util)}]
```

(that is, select time series data of active_processes and cpu_utilization for all hosts in the oa_sysperf_global collection).

`attributeQuery` takes three parameters using the following syntax:

```
metricQuery(<table name>, {<where clause>), {<select>})
```

`attributeQuery` is intended as a generalized approach to formulate a query on attributes that yields single or aggregated attribute data. Note the '{' delimeters are used instead of normal '(' to group the clauses. An example AQL expression use of `attributeQuery is:`

```
[attributeQuery(oneview_rest_inventory, {(i.category_name ==
"enclosures")}, {i.name}]
```

(that is, select the name of all enclosures from the oneview_rest_inventory collection).

Bucket Functions

The `bucket` function is used to group the counts of items in a data set that fall into various partitions. An example use scenario is that given the overall cpu_utilization of a set of hosts we'd like to see how many fall into the 0-10 percent, 10-20 percent, and so forth.

The parameters to the bucket function are:

- `AQL expression – This is the aql query that yields the data set to partition.`
- `numbuckets (optional) – This is an integer to define the number of partitions.`
- `min & max (optional) – These two numbers specify the range of values to partition.`
- `aliasforbucketmemberscount (optional) – This is a label to provide a meaningful name for the count of items in each partition.`

Consider the following example use case of the bucket function:

```
[bucket[metricQuery(oneview_rabbitmq_metrics,{i.category=="server-
hardware"},{i.resource_uri},{aggregate_avg(i.cpu_utilization)})](numbucket
s=5,min=0,max=100,aliasforbucketmemberscount="Number Of Servers")]
```

- `numbuckets`: Change the value of this parameter to the number of partitions you want to display. The default value is 5 if you do not assign a value.

- `min`: Change the value of this optional parameter to the minimum data value you want partitioned. `min` is an optional parameter. If f you use it, you must use it along with the `max` parameter for this parameter to function correctly.

- `max`: Change the value of this optional parameter to the maximum data value you want partitioned. `max` is an optional parameter. Iif you use it, you must use it along with the `min` parameter for this parameter to function correctly.
  Note: If you do not specify the `min` and `max` parameters, the range (`min`/`max`) is automatically calculated and the entire range of values are partitioned into buckets.

- `Aliasforbucketmemberscount`: Change the value of this optional parameter if you need a meaningful name for the count of items in each partition. If this parameter is not specified, then the `"bucketmemberscount"` string is used as the label.

AQL Expressions

AQL expressions include multiple AQL functions. Use AQL expressions when you want the results of multiple queries to be combined into a single query pane in a dashboard.

You can use AQL functions in an AQL expression in any of the following ways:

Use a single AQL function

Syntax: `[<aql_function_invocation>]`

Concatenate multiple AQL functions

Concatenating multiple AQL functions enables you to concatenate the results from each AQL function as if they were run individually.

Syntax: `[<aql_function1>,<aql_function2>, ….<aql_functionn>]`

The following AQL function returns the concatenation of the results from the following:

- moving averages of CPU utilization
- outlier values for the data set of moving averages of CPU utilization

```
[oaSysperfMovingMetric("*.mydomain.com", cpu_util, moving_avg),
oaSysperfOutlierMovingMetric("*.mydomain.com", cpu_util, moving_avg)]
```

```
/* Returns the moving aggregation analytic function results for the
specified metric. Input parameters are host filter, metric, and analytic
function. */
```

```
define oaSysperfMovingMetric(hostFilter, metric, moving_analytic) =
from i in (oa_sysperf_global)
let analytic_interval = between($starttime, $endtime) let interval =
$interval
where i.host_name like hostFilter
group by i.host_name
select moving_analytic(i.metric);
```

```
/* Returns the outlier values for the results from a moving aggregate
analytic function on a metric. Input parameters are host filter, metric, &
analytic. */
define oaSysperfOutlierMovingMetric(hostFilter, metric, moving_analytic) =
from i in (oa_sysperf_global)
let analytic_interval = between($starttime, $endtime)
let interval = $interval where i.host_name like hostFilter group by
i.host_name
select outlier(moving_analytic(i.metric));
```

Use multiple AQL functions so that the results from one AQL function is an input filter for another AQL function

This type of AQL expression is known as an AQL composition.

Syntax: `[do <target_function> filter by <filter_function> with <filter_criteria>]`

`<target_function>` is the AQL function to execute.

`<filter_function>` is the AQL function used to filter the results.

`<filter_criteria>` is the criteria to use for filtering the results of target function. The syntax of `<filter_criteria>` is:

`(<filter_criteria_element1>, <filter_criteria_element2>, ….)`

Each `<filter_criteria_element>` specifies a metric or attribute column name with its associated collection. Values for the column name specified must be returned in the `target_function` and `filter_Function` results.

Note: All of the filter criteria elements must be met to successfully filter the target function results.

The syntax for any filter criteria element is:

```
<target_function_name>.<target_function_resultcolumn> ==
<filter_function_name>.<filter_function_resultcolumn>
```

The `<target_function_resultcolumn>` can be any of the expected result columns from the results of `<target_function>`.

`<target_function_name>` is the name of the target function.

Similarly, `<filter_function_resultcolumn>` can be any of the expected result columns from the results of `<filter_function>`. The `<filter_function_name>` is the name of the filter function.

The following example AQL expression returns the `moving_avg`, `moving_max`, and `moving min` of CPU utilization for the top five hosts with the `highest aggregate_avg cpu_util` values.

```
[do oaSysperfMovingMetricAvgMaxMin("*", cpu_util) filter by
oaSysperfTopNAggregateMetric ("*.mydomain.com",cpu_util,aggregate_avg,5)
with (oaSysperfMovingMetricAvgMaxMin.host_name==
oaSysperfTopNAggregateMetric.host_name)]
```

```
/* Returns the moving average, maximum, and minimum values of a specific
metric by host. Input parameters are the host filter and the metric. */
define oaSysperfMovingMetricAvgMaxMin(hostFilter, metric) =
from i in (oa_sysperf_global)
let analytic_interval = between($starttime,$endtime) let interval =
$interval
where i.host_name like hostFilter
```

```
group by i.host_name
select moving_avg(i.metric), moving_max(i.metric), moving_min(i.metric);

/* Returns the topN of a moving aggregate analytic function on a metric.
Input parameters are the host filter, metric, moving aggregate analytic
function, and N. */

define oaSysperfTopNMovingMetric(hostFilter, metric, moving_analytic, N) =
from i in (oa_sysperf_global)
let analytic_interval = between($starttime, $endtime) let interval =
$interval
where i.host_name like hostFilter group by i.host_name
select topN(moving_analytic(i.metric), N);
```

The following AQL expression returns the aggregate_avg CPU utilization for all server nodes in the Operations Analytics topology. These servers include the database server nodes. This example uses topology data to filter and return metric analysis for important entities in your topology:

```
[do oaSysperfAggregateMetric("*",cpu_util,aggregate_avg) filter by
opsaNodes()
with (
oaSysperfAggregateMetric.host_name== opsaNodes.opsa_server_name,
oaSysperfAggregateMetric.host_name== opsaNodes.collector_server_name,
oaSysperfAggregateMetric.host_name== opsaNodes.logger_server_name,
oaSysperfAggregateMetric.host_name== opsaNodes.vertica_node
)]

/* Returns the results of the overall aggregate analytic function applied
to the specified metric. Input parameters are host filter, metric, and
overall aggregate analytic function. */

define oaSysperfAggregateMetric(hostFilter,metric,aggregate_analytic) =
from i in (oa_sysperf_global) let analytic_interval = between($starttime,
$endtime)
where i.host_name like hostFilter
group by i.host_name
select aggregate_analytic(i.metric);

/* Returns the host names of Operations Analytics application servers,
logger servers, collector servers, and vertica nodes in an Operations
Analytics deployment */
define opsaNodes() = from i in (opsa_topology) select i.opsa_server_name,
i.logger_server_name, i.collector_server_name, i.vertica_node;
```

AQL Modules: Importing Analytic Query Language Functions

By default, Operations Analytics provides several AQL functions to assist you with creating AQL queries, AQL functions, and associated dashboards. The concepts in this manual help you write your own AQL functions using a text editor.

Once you write your AQL functions, you can import these functions into Operations Analytics. Each text file you create can contain any number of AQL functions. A set of AQL functions that reside in a single file are known as an AQL module.

Tip: Tip: Use the `bpm_functions.aql` module as an example. This AQL module contains several AQL functions that can be used as a template for creating your own. They reside in the `$OPSA_HOME/inventory/lib/hp/aql` directory.

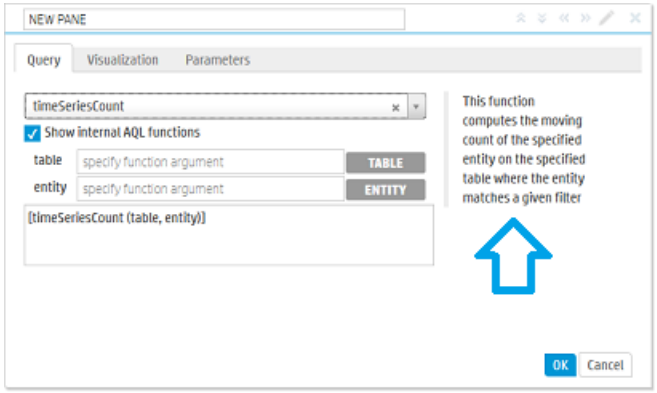
You can also view these AQL functions when you use the **Add A Query Pane** option from an

Operations Analytics dashboard. See *Dashboards and Query Panes* in the *Operations Analytics Help* for more information.

Note: To view the AQL query associated with each AQL function provided by Operations Analytics, look at the `.aql` files in `$OPSA_HOME/inventory/lib/hp/aql` or use the `opsa-aqlmodule- manager.sh` command.

When creating AQL functions to be imported, note the following:

- The comment preceding each AQL function is displayed as the description for the AQL function selected as shown in the following example:



- As a best practice, name your file using an `.aql` extension.
- As a best practice, use the validate option in the opsa`-aql-module-manager.sh` script to ensure your module will import.
- As a best practice, place your file in the `$OPSA_HOME/inventory/lib/user/aql` directory before it is imported. This helps to ensure that the file is not overwritten when upgrading to a new Operations Analytics version.
- To make your AQL functions available to your user community, use the `opsa-aql-modulemanager.sh` script. This script imports the AQL functions defined in your module into the Operations Analytics database and makes them available to your user community by default.

By default, Operations Analytics provides several AQL functions to assist you with creating AQL queries, AQL functions, and associated dashboards.

You can write your own AQL functions using a text editor and then import these functions into Operations Analytics. Each text file you create can contain any number of AQL functions. Each set of AQL functions contained in a single file is known as a **module**.

Use the `opsa-aql-module-manager.sh` script to manage the AQL functions that you create. When using the `opsa-aql-module-manager.sh` script, note the following:

- You must specify the tenant name for which the AQL functions should be available.
- Use file names that identify the types of AQL functions contained in each file.
- You define the `<module_name>` in the first line of each file; for example: `module <my_new_module>;`
- You validate, list, and delete modules using the module name.

Use the `opsa-aql-module-manager.sh` script to perform the following tasks:

Validate the AQL functions included in n module file

Run the following command: `opsa-aql-module-manager.sh -t <tenant_name> -v <file_name>`

Note: The `opsa-aql-module-manager.sh` script does not currently detect some syntax errors, such as unbound variables referenced within the body of an AQL function. Take extra care when creating and editing your AQL functions.

Import an AQL Module

Enter the following command: `opsa-aql-module-manager.sh -t <tenant_name> -i <file_name>`

When importing AQL functions, note the following:

- After importing your AQL functions, all functions are available to the user community in the specified tenant.
- To replace or redefine AQL functions, you must make the appropriate changes to the `.aql` module, then re-import the file.

List all AQL modules that have been imported into Operations Analytics

Run the following command: `opsa-aql-module-manager.sh -t <tenant_name> -l modules`

List the AQL functions contained in a module that has been imported into Operations Analytics

Run the following command: `opsa-aql-module-manager.sh -t <tenant_name> -l <module_name>`

See *opsa-aql-module-manager.sh reference page* (or the Linux manpage) for more information.

Analytics Query Language for Log Data

The information in this section explains how to use AQL functions to search log file information. Examples in this topic use AQL to return the information collected by log files configured using HP ArcSight Logger.

Note: The queries in this section do not apply to structured log files. Structured log files are fragments of log file data that are stored as collections in HP Operations Analytics. Structured logs are log files that are configured as collections. These collections are created so that users can perform analytics on the log file contents. For example, you might want to query for all outliers by host name and application for a particular time range.

You can use three types of AQL functions to search log file information:

- To search for text strings use the `aqlrawlog` function.
- To count the number of log file entries, use the `aqlrawlogcount` function.
- To enter a query supported by HP ArcSight Logger, use the `aqlrawlogarbitrary` function.

Using the aqlrawlog Function to Search for Text Strings

Use the `aqlrawlog` function to search the log file entries stored in HP ArcSight Logger servers.

The `aqlrawlog` query returns the following attributes for each matching log file message entry: `timestamp`, `message text`, `host name`, and `source host name`.

Syntax: `aqlrawlog(<aqllit><text_to_search></aqllit>, <starttime_as_seconds_since_epoch>, <end_time_as_seconds_since_epoch>, ""|"<comma_separated_list_of_logger_host_names>"[,<limit>])`

aqlrawlog arguments

`[let timeout=<timeout_in_seconds>]`
`[let limit=<limit>]`
<*text_to_search*> is the text string that must match in the log file entries.

Note: The <*text_to_search*> argument must be enclosed by the `<aqllit>` keyword. For example `<aqllit>severity</aqllit>`.

<*starttime_as_seconds_since_epoch*> is the start time of the time window within which to look for matching log file entries.

Note: To use the value selected in the Operations Analytics console, enter `$starttime` as the value for this argument.

<*endtime_as_seconds_since_epoch*> is the end time of the time window within which to look for log file entries.

Note: To use the value selected in the Operations Analytics console, enter `$endtime` as the value for this argument.

<*comma_separated_list_of_logger_host_names*> is a comma separated list of host names that identify the HP ArcSight Logger servers to query.

Tip: To query all of the HP ArcSight Logger servers configured for the current tenant, specify "" as this parameter value.

<*limit*> is an optional parameter that overrides the default maximum number of log file entries to return.

Note: If you do not use this parameter or the optional `let limit=`<*limit*> clause, Operations Analytics returns up to a maximum of 2000 log file messages matching the search text. You can also specify `$limit` for this value.

<*timeout_in_seconds*> is the timeout for the search operation. This parameter is specified when using the optional `let timeout=`… clause.

Note: If you do not specify this parameter, Operations Analytics uses the default timeout value.

Examples:

```
/* Returns a maximum of 500 log file entries that include "error" */

aqlrawlog(<aqllit>error</aqllit>, $starttime, $endtime, "", 500)

/*Returns the default maximum number of log file entries that include
"error". This query searches log file entries only on the following
servers: mylogger1.mydomain.com and mylogger2.mydomain.com logger
servers*/

aqlrawlog(<aqllit>error</aqllit>, $starttime, $endtime,

"mylogger1.mydomain.com,mylogger2.mydomain.com")

/* Returns the default maximum number of log file entries that include
"error". It uses the timeout value of 5 minutes */

aqlrawlog(<aqllit>error</aqllit>, $starttime, $endtime, "") let
timeout=300

/* Returns a maximum number of 500 log file entries that include "error".
It uses the timeout value of 5 minutes */

aqlrawlog(<aqllit>error</aqllit>, $starttime, $endtime, "") let
timeout=300 let limit=500
```

Use the aqlrawlogcount Function to Count the Number of Log File Entries

Use the `aqlrawlogcount` function to count the log file entries stored in HP ArcSight Logger servers that contain the search text string.

Syntax: `aqlrawlogcount(<aqllit>`<*text_to_search*>`</aqllit>,` <*starttime_as_seconds_since_epoch*>, <*end_time_as_seconds_since_epoch*>, ""

```
| "<comma_separated_list_of_logger_host_names>","" |
"<comma_separated_list_of_group_by_fields>" [,<granularity_in_seconds>])
```

Description of each of the aqlrawlogcount arguments

`[let timeout=<timeout_in_seconds>]`

`[let limit=<limit>]`

`<text_to_search>` is the text string that must match in each log file entry returned.

Note: The `<text_to_search>` argument must be enclosed by the `<aqllit>` keyword, for example `<aqllit>severity</aqllit>`.

`<starttime_as_seconds_since_epoch>` is the start time of the time window within which to look for matching log file entries.

Note: To use the value selected in the Operations Analytics console, enter `$starttime` as the value for this argument.

`<endtime_as_seconds_since_epoch>` is the end time of the time window within which to look for matching log file entries.

Note: To use the value selected in the Operations Analytics console, enter `$endtime` as the value for this argument.

`<comma_separated_list_of_logger_host_names>` is a comma separated list of host names that identify the HP ArcSight Logger servers to query.

Tip: To query all of the HP ArcSight Logger servers configured for the current tenant, specify `""` as this parameter value.

`<limit>` is an optional parameter that overrides the default maximum number of log file entries to return.

Note: If you do not use this parameter or the optional `let limit=<limit>` clause, Operations Analytics returns up to a maximum of 2000 log file messages matching the search text. You can also specify `$limit` as the value.

`<timeout_in_seconds>` is the timeout for the search operation specified using the optional `let timeout=…` clause.

Note: If you do not specify this parameter, Operations Analytics uses the default timeout value.

The aqlrawlog query returns the following attributes for each matching log file entry: `timestamp`, `message text`, `host name`, and `source host name`.

`<comma_separated_list_of_group_by_fields>` is a comma separated list of the HP ArcSight Logger attributes in which to group the results.

Tip: If you do not want Operations Analytics to group the results, specify `""` as the parameter value.

Note: If you specify `""` as this parameter and do not specify `<granularity_in_seconds>`, Operations Analytics computes the moving counts without any group by criteria.

The window of time between `<starttime_as_seconds_since_epoch>` and `<endtime_as_seconds_ since_epoch>` is divided into multiple intervals. Operations Analytics calculates counts at each of these intervals. Operations Analytics automatically computes the optimal length of time for each interval.

`<time_interval_in_seconds>` specifies the value Operations Analytics should use to subdivide the window of time between `<starttime_as_seconds_since_epoch>` and `<endtime_as_seconds_since_epoch>`. Operations Analytics computes the moving counts at each of these intervals.

Note: To use the value selected in the Operations Analytics console, enter `$interval` as the value for this argument. See *Dashboards and Query Panes* in the *Operations Analytics Help* for more information about how to specify the `$interval` parameter value in the Operations Analytics console.

`<limit>` is an optional parameter that overrides the default maximum number of log file entries to return.

Note: If you do not use this parameter or the optional `let limit=<limit>` clause, Operations Analytics returns up to a maximum of 2000 log file messages matching the search text. You can also specify `$limit` as the value.

`<timeout_in_seconds>` is the timeout for the search operation specified using the optional `let timeout=...` clause.

Note: If you do not specify this parameter, Operations Analytics uses the default timeout value.

Examples

```
/* Returns the time series counts of log file entries that contain "error" at 5 minuteintervals*/

aqlrawlogcount(<aqllit>error</aqllit>, $starttime, $endtime, "", "", 300)

/*Returns the time series counts of log file entries that contain "error" for each combination of deviceHostName and agentSeverity at 5 minute intervals. The function queries only the mylogger1.mydomain.com server*/

aqlrawlogcount(<aqllit>error</aqllit>, $starttime, $endtime,
"mylogger1.mydomain.com", "deviceHostName,agentSeverity", 300)

/*Returns overall aggregate counts of log file entries that contain "error" for each combination of deviceHostName and agentSeverity. This AQL function queries only themylogger1.mydomain.com server*/

aqlrawlogcount(<aqllit>error</aqllit>, $starttime, $endtime,
"mylogger1.mydomain.com", "deviceHostName,agentSeverity")

/*Returns the time series of counts of log file entries that contain "error" for each combination of deviceHostName and agentSeverity at 5 minute intervals. This AQL function queries only mylogger1.mydomain.com, uses the timeout value of 10 minutes, and queries a maximum of 1000 entries */

aqlrawlogcount(<aqllit>error</aqllit>, $starttime, $endtime,
"mylogger1.mydomain.com", "deviceHostName,agentSeverity", 300) let
timeout=600 let limit=1000
```

Use the aqlrawlogarbitrary Function to Enter a Query Supported by HP ArcSight Logger

Note: A supported query is any query that is configured for use on an HP ArcSight Logger server.

Use the `aqlrawlogarbitrary` function to run any other query supported by your HP ArcSight Logger server.

Operations Analytics displays the `aqlrawlogarbitrary` results table format.

Syntax: `aqlrawlogarbitrary(<aqllit><query_string></aqllit>, <starttime_as_seconds_since_epoch>, <end_time_as_seconds_since_epoch>, ""|"<comma_separated_list_of_logger_host_names>" [,<limit>])`

Description of each of the aqlrawlogarbitrary function arguments.

`[let timeout=<timeout_in_seconds>]`

`[let limit=<limit>]`

`<query_string>` is the query string that is supported by your HP ArcSight Logger server.

Note: The <*query_string*> argument must be enclosed by the <aqllit> keyword, for example: <aqllit>severity</aqllit>.

<*starttime_as_seconds_since_epoch*> is the start time of the time window within which to look for matching log file entries.

Note: To use the value selected in the Operations Analytics console, enter $starttime as the value for this argument.

<*endtime_as_seconds_since_epoch*> is the end time of the time window within which to look for matching log file entries.

Note: To use the value selected in the Operations Analytics console, enter $endtime as the value for this argument.

<*comma_separated_list_of_logger_host_names*> is a comma separated list of host names of the HP ArcSight Logger servers to query.

Tip: To query all of the HP ArcSight Logger servers configured for the current tenant, specify "" as this parameter value.

<*limit*> is optional parameter and if specified it overrides the default maximum rows of information returned by logger to consider for returning back to the Operations Analytics console.

Note: If you do not use this parameter or the optional let limit=<limit> clause, Operations Analytics returns up to a maximum of 2000 log file messages matching the search text. You can also specify $limit for this value.

<*timeout_in_seconds*> is the timeout for the search operation specified using the optional let timeout=... clause.

Note: If you do not specify this parameter, Operations Analytics uses the default timeout value.

Examples

```
/* Returns a maximum of 500 log file entries that contain the text string "error" */

aqlrawlogarbitrary(<aqllit>error</aqllit>, $starttime, $endtime, "", 500)
```

/*Returns up to the default maximum number of log file entries that contain the text string "error". This AQL function queries only the mylogger1.mydomain.com and mylogger2.mydomain.com logger servers*/

```
aqlrawlogarbitrary(<aqllit>error</aqllit>, $starttime, $endtime, "mylogger1.mydomain.com,mylogger2.mydomain.com")

/* Returns the default maximum number of log file entries that contain "error". This AQL function uses a timeout value of 5 minutes */

aqlrawlogarbitrary(<aqllit>error</aqllit>, $starttime, $endtime, "") let timeout=300

/* Returns a maximum of 500 log file entries that contain "error". This AQL function uses a timeout value of 5 minutes */

aqlrawlogarbitrary(<aqllit>error</aqllit>, $starttime, $endtime, "") let timeout=300 let limit=500
```

You can add a let clause to your aqlrawlog, aqlrawlogcount, or aqlrawlogarbitrary query to define a variable that contains a list of entities returned from an AQL function. The variable can then be used in the <aqllit><*text_to_search*></aqllit> string. This feature is useful when you want to search for a set of entities, such as hosts, applications, Business Service Management transactions, or database instances without needing to enter the entire list of values.

- To add a let clause to your aqlrawlog, aqlrawlogcount or aqlrawlogarbitrary query, use the following syntax:

```
let <variable_name>=<AQL_function>
```

For example, you could define the variable `$myhosts` to contain the list of servers returned from the AQL function named `oaSysperfHosts`. The `oaSysperfHosts` AQL function uses the following arguments to return hosts that have performance metrics collected:

```
oaSysperfHosts (hostFilter, numHostsLimit)
```

To define a variable to store the results returned from the oaSysperfHosts AQL function, use the following syntax:

```
let <variable_name>=oaSysperfHosts (hostFilter, numHostsLimit)
```

For example, to pass the first 50 hosts that have performance metrics collected in the enterprise.com domain to the myhosts variable, add the following let clause to your `aqlrawlog`, `aqlrawlogcount`, or `aqlrawlogarbitrary` query:

```
let myhosts=oaSysperfHosts ("*enterprise.com", 50)
```

- The variable you define using the `let` clause can be used in a text search or with a Common Event Field (CEF) field that was configured using the Operations Analytics Log File Connector for ArcSight Logger. See *Configuring the Operations Analytics Log File Connector for ArcSight Logger* in the *HP Operations Analytics Configuration Guide for* more information.
- To use the variable in a text search, use the following syntax:
  `<aqllit><$variable></aqllit>`
  For example:
  `<aqllit><$myhosts></aqllit>`
- To use the variable with a CEF, use the following syntax in place of `<aqllit><$variable></aqllit>`:
  `<aqllit><CEF> in [$<variable_name>]</aqllit>`
  For example: `sourcehostName in [$myhosts]`

  The previous example searches for all log file messages that contain any of the host names stored in the `$myhosts` variable. These host names would be the first 50 hosts that have performance metrics collected in the enterprise.com domain.

Using R with AQL

Setting up the R Language Pack from Vertica

The purpose of this section is to document the steps that custom analytics developers can take to register custom analytics written using R and make use of them on data being collected by Operations Analytics. Using Operations Analytics 2.10 or newer, Operations Analytics users can execute R functions on top of an Analytics Query Language (AQL) function that fetches entities and some measurements done for them based on data collected by Operations Analytics.

Operations Analytics uses Vertica's R language runtime environment as the runtime environment for any R function you register with both Operations Analytics and Vertica. It is mandatory that you have the Vertica R Language Pack set up on each node of the Vertica cluster used by your Operations Analytics deployment. You must install the following packages on each node of the Vertica cluster in order to set up the Vertica R language pack:

compat-libgfortran-41-4.1.2- 52.el5_8.1.x86_64.rpm

vertica-R-lang- 6.1.3-12.x86_64.RHEL5.rpm

The former is a pre-requisite for the latter.

Complete the instructions shown in the *Use an Existing Vertica Installation and use the R Language Pack from Vertica* section of the *Operations Analytics Installation Guide.*

Creating the R Functions that Integrate with Operations Analytics

Operations Analytics expects all R functions to conform to the Vertica R UDX framework (R UDX). In order to have a valid R UDX, Vertica expects the following:

1. R functions must have a corresponding UDX factory function written in R. This function must capture input, output frame descriptions, and descriptions of optional input parameters to the core R function.
2. If an R function's output frame does not contain a fixed number of columns with fixed types, then the factory function needs to specify an output type callback R function that is written by the user. The output callback function describes the output frame structure to Vertica at runtime.
3. If an R function expects parameters for configuring the behavior of the computation done in the core R function in question, then those parameters need to be described using a parameter callback R function and the factory function must specify the same parameters.
4. It is expected that the core R function, the UDX factory R function, any optional output callback R function and any optional parameter callback R function must all be present in a single .R file that is used for registering the R function as a valid Vertica R UDX.

Identifying the Statistical Random Variables in an Input Frame for an R function

An R function's integration with Operations Analytics currently assumes that the R function is written so that it identifies the statistical random variables in the Operations Analytics domain from the input data frame that is fed to the R function at runtime. The following information helps you better understand the concept of Operations Analytics random variables and how to write R code to identify these variables in the input frames.

As noted earlier, one can use Operations Analytics dashboard panes to invoke R functions on an AQL function that results in Operations Analytics timeseries data.

At runtime, an AQL function is translated to a Vertica SQL. In addition, Operations Analytic's AQL wraps the Vertica SQL inside of another Vertica SQL involving the registered R UDX invocation.

The Vertica SQL translated from the AQL function represents the query that Vertica would run to supply the results as an input data frame to the R function.

AQL also supplies three internal parameters to an R function call in the outer SQL involving the invocation of the R function: numentityparts, entitypart1columnindex, and timestampcolumnindex. These parameters should permit R function writers to identify the entities, measurement names combinations, and their corresponding timeseries data. Once entity, measurements, and their corresponding timeseries data are identified, each entity, measurement combination could be considered a valid random variable backed by the timeseries data for itself being the observations for the random variable.

Operations Analytics provides some example .R files containing core R functions, their UDX factory R functions, output callback functions, and parameter callback functions, in the following location: */opt/HP/opsa/inventory/lib/hp/r-udx-examples*

See the example named `MVCorr.R` that attempts to do statistical correlation between pairs of such random variables.

Note: You must provide a `parametertypecallback` R function in your `.R` file in order to achieve smooth passing of these parameters to the R function at runtime. The `parametertypecallback` R function is recognized by Vertica UDX framework as an optional function that allows specifying the

description of parameters to UDX framework. For integration with Operations Analytics, this callback is mandatory.

The following snippet from MVCorr.R example, demonstrates the boiler plate code that needs to be written for the `parametertypecallback` R function. This snippet also demonstrate how to specify the `parametertypecallback` function (in this example the function named *`mvCorrParamType`*) in the UDX factory function (in this example its function named *`mvCorrFactory`*).

```
mvCorrParamType<-function(){
        params <- data.frame(datatype=rep(NA, 3), length=rep(NA,3),
scale=rep(NA,3),name=rep(NA,3) )

        params[1,1] = "int"
        params[1,4] = "numentityparts"

        params[2,1] = "int"
        params[2,4] = "entitypart1columnindex"

        params[3,1] = "int"
        params[3,4] = "timestampcolumnindex"
        params
}

mvCorrFactory<-function()
{
        list(name=mvCorr,udxtype=c("transform"),intype=c("any"),
outtype=c("any"),
outtypecallback=mvCorrOutType,parametertypecallback=mvCorrParamType)
}
```

The various other helper methods in `MVCorr.R` also illustrates one could write code to identify the random variables in the input frame based on the parameters passed by AQL.

Take a look at the optional Vertica R UDX framework recognized `outtypecallback` R function that attempts to establish the contract with Vertica for the outgoing result or output frame columns. If you want the frame columns output by specific names or want to specify specialized types for some of these columns, then you must code the `outtypecallback` R function and register the same in UDX factory R function.

```
mvCorrOutType<-function(x,y){

    entityStartIndex<-y[['entitypart1columnindex']]
    numEntityParts<-y[['numentityparts']]
```

```
    ret <-
data.frame(datatype=rep(NA,(numEntityParts*2+2+1)),lenth=rep(NA,(numEntityParts*2+2+1)),scale=rep(NA,(n
umEntityParts*2+2+1)),name=rep(NA,(numEntityParts*2+2+1)))

    for ( i in 0: numEntityParts ){
        ret[i+1, 1]="varchar"
        ret[i+1, 4]=paste("rv1",x[entityStartIndex+i, 4],sep="_")
    }

    ret[numEntityParts+1, 1]="varchar"
    ret[numEntityParts+1, 4]="rv1_metric"

    for ( i in 0: numEntityParts ){
        ret[i + numEntityParts + 2, 1]="varchar"
        ret[i + numEntityParts + 2, 4]=paste("rv2",x[entityStartIndex+i,4],sep="_")
    }

    ret[numEntityParts*2+2, 1]="varchar"
    ret[numEntityParts*2+2, 4]="rv2_metric"

    ret[numEntityParts*2+2+1, 1]="float"
    ret[numEntityParts*2+2+1, 4]="correlation_coeff"
    ret
}
```

Note: Notice how the input parameters are used by the `mvCorrOutType` callback function to describe the output column names and types.

The names used above in the `outtypecallback` function are directly processed by AQL in its result processing and sent to the dashboard pane in the Operations Analytics console.

Registering an R Function

You must register a newly created R Function with both Vertica and Operations Analytics.

Registering your R function with Vertica

1. Prepare the `.R` file so that it contains the following:
    - The core R function implementing your custom analytics logic.
    - The Vertica R UDX factory function.
    - The parameter type callback R function.
    - Any optional output type callback R function.

- Any other helper R functions used by the core R function.

2. Run the Vertica R UDX load commands to load the R function into Vertica. At this stage the R function becomes available as a valid UDX that can be invoked from Vertica SQL.
Note: You must complete these steps as a valid Vertica database user who has the privileges to run SQL commands and who can create UDX functions in the Vertica database system.

   The following are example Vertica SQL commands that load the example R UDX provided by MVCorr.R:

   ```
   create library mvCorrLib as '/home/dbadmin/functions/MVCorr.R'
   language 'R';

   create transform function mvCorr as language 'R' name
   'mvCorrFactory' library mvCorrLib;
   ```

   You can also review Vertica documentation on how to load Vertica R UDX functions.

Registering your R function with Operations Analytics

Once the R function is loaded and available in Vertica, you need to tell Operations Analytics about it by registering an R function module into Operations Analtyics.

1. Create an R module specification file. The following example shows the contents of one such module definition file that defines the R module for the multi-variate correlation R UDX function example from the `/opt/HP/opsa/inventory/lib/hp/r-udx-examples/mvCorr.R` file.

   ```
   module MultiVariate;


   /* Does multivariate correlation */

   define mvCorr input(any, integer, integer) output(any);
   ```

   Save the content in a text file. For example, see `/opt/HP/opsa/inventory/lib/hp/r/multivariate.rpsec`

2. Load the R module specification into OPERATIONS ANALYTICS by running the following command:

```
/opt/HP/opsa/bin/opsa-rspec-module-manager.sh -?
```

   You should see an output similar to the following:

```
OPSA_HOME is set to /opt/HP/opsa


-t <tenant name>              Name of Tenant (mandatory argument except when using -v
option)
-v <file>                     Validate File
-l modules                    List Summary of Loaded Modules
-l all                        List Contents of All Loaded Modules
-l <modulename>               List Contents of Module
-i <file>                     Import File
-a <authorname>               Specify Author for Import File
-d <modulename>               Delete Module
-?                            This help message
```

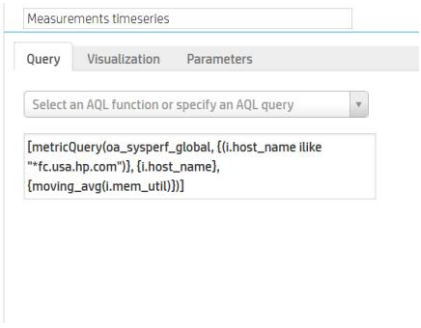
For example, you could load the R module named MultiVariate previously defined in the */opt/HP/opsa/inventory/lib/hp/r/multivariate.rpsec* file by running the following command:

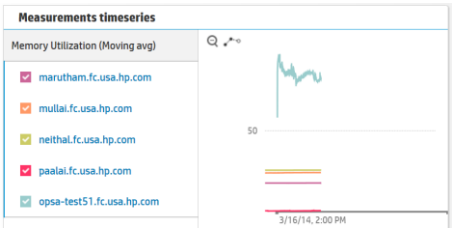
```
/opt/HP/opsa/bin/opsa-rspec-module-manager.sh – t opsa_default -i
/opt/HP/opsa/inventory/lib/hp/r/multivariate.rpsec
```

Using your R Function in an Operations Analytics Dashboard

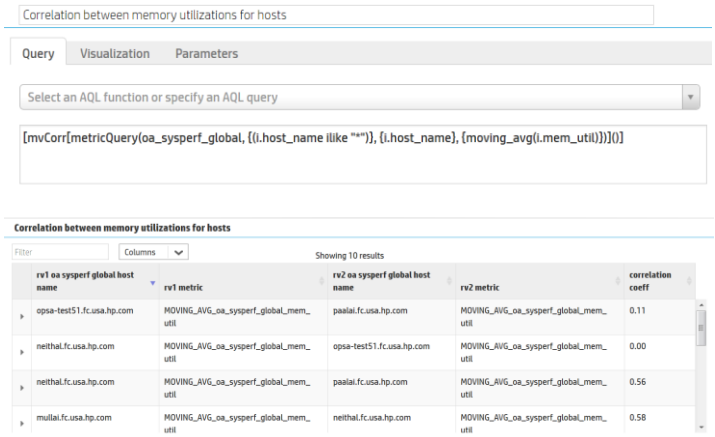
You can create your AQL function that returns time series data that you can visualize using an Operations Analytics line chart, heatmap chart, or bar chart UI elements.

In a dashboard pane, you can visualize the results of the AQL function by using it in the query edit box for the pane as shown below:





Now surround the AQL function call with a call to an R function as shown below to trigger the invocation of the registered R function:



Limitations

Only a table visualization of the invoked R function is supported in Operations Analytics 2.20.

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Microsoft and Windows are trademarks of the Microsoft group of companies.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:
https://softwaresupport.hp.com/group/softwaresupport/search-result?keyword=

This site requires an HP Passport account. If you do not have one, click the **Create an account** button on the HP Passport Sign in page.

Support

Visit the HP Software Support Online web site at: **https://softwaresupport.hp.com**

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software Support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to https://softwaresupport.hp.com and click **Register**.

To find more information about access levels, go to: https://softwaresupport.hp.com/web/softwaresupport/access-levels