Engineering
Simplicity

Juniper Secure Analytics Ariel Query Language (AQL) Guide

Published
2022-05-13

RELEASE
7.5.0

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Secure Analytics Ariel Query Language (AQL) Guide
7.5.0

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

Table of Contents

About This Guide

Use this guide to understand AQL functions which are built into Ariel database with AQL statements.

1

CHAPTER

Ariel Query Language

Ariel Query Language

The Ariel Query Language (AQL) is a structured query language that you use to communicate with the Ariel databases. Use AQL to query and manipulate event and flow data from the Ariel database.

Sample AQL Queries

Use Ariel Query Language (AQL) queries to retrieve data from the Ariel database based on specific criteria.

Use the following query syntax, and adhere to the clause order, when you build an AQL query:

- `[SELECT *, column_name, column_name]`
- `[FROM table_name]`
- `[WHERE search clauses]`
- `[GROUP BY column_reference*]`
- `[HAVING clause]`
- `[ORDER BY column_reference*]`
- `[LIMIT numeric_value]`
- `[TIMEFRAME]`

NOTE: When you use a `GROUP BY` or `ORDER BY` clause to sort information, you can reference column_names from your existing `SELECT` statement only.

NOTE: By default, if the `TIMEFRAME` value is not specified, the query runs against the last five minutes of Ariel data.

Remember to use single quotation marks to specify literal values or variables and use double quotation marks for column names that contain spaces or non-ASCII characters:

Single quotation marks

Use single quotation marks when you reference the beginning and end of a string, as shown in these examples:

```
username LIKE ’%User%'
```

```
sourceCIDR= ’192.0.2.0’
```

```
TEXT SEARCH = ’VPN Authenticated user'
```

```
QIDNAME(qid) AS 'Event Name'
```

Double quotation marks

Use double quotation marks when column names contain spaces or non-ASCII characters, as shown in these examples:

Custom property names with spaces, such as “Account Security ID”.

Values that have non-ASCII characters.

Simple AQL Queries

Table 1: Simple AQL Queries

Basic AQL Commands	Comments
`SELECT * FROM events LAST 10 MINUTES`	Returns all the fields from the events table that were sent in the last 10 minutes..
`SELECT sourceip,destinationip FROM events LAST 24 HOURS`	Returns the `sourceip` and `destinationip` from the events table that were sent in the last 24 hours.
`SELECT * FROM events START ’2017 01 01 9:00:00’ STOP ’2017 01 01 10:20:00’`	Returns all the fields from the events table during that time interval.
`SELECT * FROM events limit 5 LAST 24 HOURS`	Returns all the fields in the events table during the last 24 hours, with output limited to five results.
`SELECT * FROM events ORDER BY magnitude DESC LAST 24 HOURS`	Returns all the fields in the events table sent in the last 24 hours, sorting the output from highest to lowest magnitude.

Table 1: Simple AQL Queries *(Continued)*

Basic AQL Commands	Comments
`SELECT * FROM events WHERE magnitude >= 3 LAST 24 HOURS`	Returns all the fields in the events table that have a magnitude that is less than three from the last 24 hours.
`SELECT * FROM events WHERE sourceip = ’192.0.2.0’ AND destinationip = ’198.51.100.0’ START ’2017 01 01 9:00:00’ STOP ’2017 01 01 10:20:00’`	Returns all the fields in the events table that have the specified source IP and destination IP within the specified time period.
`SELECT * FROM events WHERE INCIDR(’192.0.2.0/24’, sourceip)`	Returns all the fields in the events table where the source IP address is within the specified CIDR IP range.
`SELECT * FROM events WHERE username LIKE ’%roul%’`	Returns all the fields in the events table where the user name contains the example string. The percentage symbols (%) indicate that the user name can match a string of zero or more characters.
`SELECT * FROM events WHERE username ILIKE ’%ROUL%’`	Returns all the fields in the events table where the user name contains the example string, and the results are case-insensitive. The percentage symbols (%) indicate that the user name can match a string of zero or more characters.
`SELECT sourceip,category,credibility FROM events WHERE (severity > 3 AND category = 5018)OR (severity < 3 AND credibility > 8)`	Returns the `sourceip`, `category`, and `credibility` fields from the events table with specific severity levels, a specific category, and a specific credibility level. The AND clause allows for multiple strings of types of results that you want to have.
`SELECT * FROM events WHERE TEXT SEARCH ’firewall’`	Returns all the fields from the events table that have the specified text in the output.
`SELECT * FROM events WHERE username ISNOT NULL`	Returns all the fields in the events table where the username value is not null.

Ariel Query Language in the JSA User Interface

Using AQL can help enhance advanced searches and provide specific results.

When you use AQL queries, you can display data from all across JSA in the Log Activity or Network Activity tabs.

To use AQL in the search fields, consider the following functions:

- In the search fields on the **Log Activity** or **Network Activity** tabs, type Ctrl + Space to see the full list of AQL functions, fields (properties), and keywords.
- Ctrl + Enter helps you create multiline AQL queries in the user interface, which makes the queries more readable.
- By using the copy (Ctrl + C) and paste (Ctrl + V) keyboard commands, you can copy directly to and from the **Advanced search** field.

NOTE: Ensure that you use appropriate quotation marks when you copy queries to the search field.

The AQL categories are listed with the entered component in the user interface.

The following table lists and explains the different categories:

Table 2: Ariel Query Language categories

Category	Definition
Database	The name of an Ariel database, or table, that you can query. The database is either `events` or `flows`.
Keyword	Typically core SQL clauses. For example, `SELECT`, `OR`, `NULL`, `NOT`, `AS`, `ASC` (ascending), and more.
Field	Indicates basic information that you can query from the database. Examples include `Access intent`, `VPC ID`, and `domainid`.

Table 2: Ariel Query Language categories *(Continued)*

Category	Definition
Function	The name of a function that is used to call in more information. Functions work on all fields and databases. Examples of functions include `DATEFORMAT`, `HOSTNAME`, and `LOWER`.

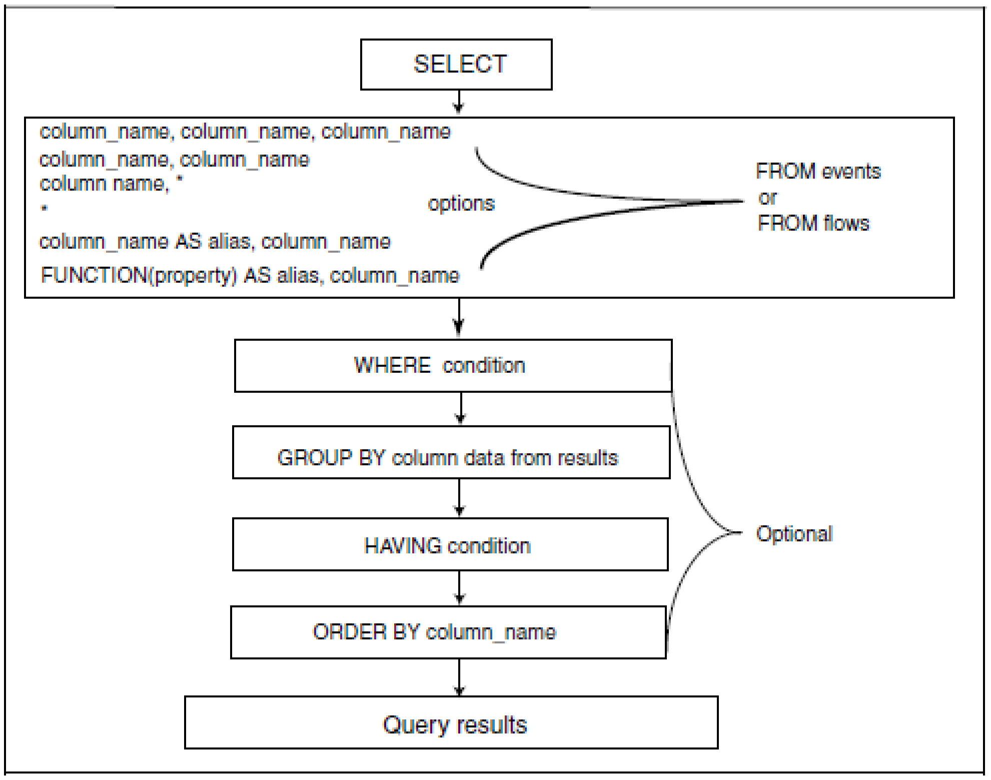
Ariel Query Structure

IN THIS SECTION

Use AQL to extract, filter, and perform actions on event and flow data that you extract from the Ariel database in JSA. You can use AQL to get data that might not be easily accessible from the user interface.

The following diagram shows the flow of an AQL query.

Figure 1: AQL Query Flow



Structure Of an AQL Statement

Use the `SELECT` statement to select fields from events or flows in the Ariel database, which are displayed as columns. For example, the following query returns the results that are shown in the following table:

```
SELECT sourceip, destinationip, username, protocolid, eventcount FROM events
```

Table 3: AQL Query Results

`sourceip`	`destinationip`	`Username`	`Protocolid`	`eventcount`
192.0.2.21	198.51.100.21	Joe Ariel	233	1

Table 3: AQL Query Results *(Continued)*

`sourceip`	`destinationip`	`Username`	`Protocolid`	`eventcount`
192.0.2.22	198.51.100.24	Jim Ariel	233	1

AQL queries begin with a SELECT statement to select event or flow data from the Ariel database. You can refine the data output of the SELECT statement by using the WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, and LAST clauses.

- **SELECT**--Use the `SELECT` statement to select fields from events or flows. For example, select all fields from events or flows by typing:

  `SELECT * FROM events`, or `SELECT * FROM flows`

Use the following clauses to filter and manipulate the data that is returned by the SELECT statement:

- **WHERE**--Use the `WHERE` clause to insert a condition that filters the output, for example, `WHERE logsourceid='65'`.
- **GROUP BY**--Use the `GROUP BY` clause to group the results by one or more columns that you specify in the query, for example, `GROUP BY logsourceid`.
- **HAVING**--Use the `HAVING` clause to specify a condition after the `GROUP BY` clause, for example, `HAVING MAG > 3`.
- **ORDER BY**--Use the `ORDER BY` clause to order the results for a column in the AQL query in an ascending or descending order, for example, `ORDER BY username DESC`.
- **LIMIT** --Use a `LIMIT` clause to limit the number of results that are returned to a specific number, for example `LIMIT 50` to limit the output to 50 results.
- **LAST**--Use a LAST clause to specify a time frame for the query, for example `LAST 1 HOURS`.

The following example incorporates all of the clauses that are described in the list:

```
SELECT sourceip, destinationip, username
FROM events
WHERE username = ’test name’
GROUP by sourceip, destinationip
ORDER BY sourceip DESC
```

```
LIMIT 10
LAST 2 DAYS
```

SELECT Statement

Use the SELECT statement to define the criteria that you use to retrieve event or flow data.

Use the `SELECT` statement to define the columns (fields) that you want to output from your query. You can use the SELECT statement to output data from an AQL function by using a column alias. Typically, you refer to events or flows in your SELECT statement but you can also use the `SELECT` statement with the `GLOBALVIEW` database, or any other database that you might have access to.

Use the `SELECT` statement to select the columns that you want to display in the query output.

A `SELECT` statement can include the following elements:

- Fields from the events or flows databases
- Custom properties from the events or flows databases
- Functions that you use with fields to represent specific data that you want to return.

  For example, the function `ASSETHOSTNAME(sourceip)` searches for the host name of an asset by source IP address at a specific time.

Use an asterisk (*) to denote all columns.

Field names and `SELECT` and `FROM` statements are not case-sensitive. For example, the following query uses different cases and it parses.

```
select Sourceip, DATEFORMAT(starTTime,'YYYY-MM-dd HH:mm') as startTime from events WHERE username is noT Null
GROUP BY sourceip ordER BY starttime lAsT 3 houRS
```

The following examples are queries that use SELECT statements:

- `SELECT * FROM flows`

  Returns all columns from the flows database.

- `SELECT sourceip, destinationip FROM events`

  Returns only the `sourceip` and `destinationip` columns from the events database.

- `SELECT sourceip, * FROM flows`

  Returns the `sourceip` column first, which is followed by all columns from the flows database.

- `SELECT sourceip AS 'MY Source IPs' FROM events`

  Returns the `sourceip` column as the alias or renamed column '`MY Source IPs`'.

- `SELECT ASSETHOSTNAME(sourceip) AS 'Host Name', sourceip FROM events`

  Returns the output of the function `ASSETHOSTNAME` as the column name `Host Name`, and the `sourceip` column from the events database.

WHERE Clause

Filter your AQL queries by using `WHERE` clauses. The `WHERE` clause describes the filter criteria that you apply to the query and filters the resulting view to accept only those events or flows that meet the specified condition.

You can apply the `WHERE` clause to add a condition to search criteria in AQL queries, which filters the search results.

A search condition is a combination of logical and comparison operators that together make a test. Only those input rows that pass the test are included in the result.

You can apply the following filters when you use `WHERE` clause in a query:

- Equal sign (=)
- Not equal to symbol (<>)
- Less than symbol (<)
- Greater than symbol (>)
- Less that or equal to symbol (<=)
- Greater than or equal to symbol (>=)
- `BETWEEN` between two values, for example (64 AND 512)
- `LIKE` case sensitive match
- `ILIKE` case insensitive match
- `IS NULL` is empty
- `AND / OR` combine conditions or either condition
- `TEXT SEARCH` text string match

Examples Of WHERE Clauses

The following query example shows events that have a severity level of greater than nine and are from a specific category.

SELECT sourceIP, category, credibility FROM events WHERE severity > 9 AND category = 5013

Change the order of evaluation by using parentheses. The search conditions that are enclosed in parentheses are evaluated first.

SELECT sourceIP, category, credibility FROM events WHERE (severity > 9 AND category = 5013) OR (severity < 5 AND credibility > 8)

Return events from the events database where the text '`typot`' is found.

SELECT QIDNAME(qid) AS EventName, * FROM events WHERE TEXT SEARCH 'typot'

The following query outputs events from the events database where health is included in the log source name.

SELECT logsourceid, LOGSOURCEGROUPNAME(logsourceid), LOGSOURCENAME(logsourceid) FROM events WHERE LOGSOURCENAME(logsourceid) ILIKE '%%health%%'

The following query outputs events where the device type ID is equal to 11 (Linux Server DSM), and where the QID is equal to 44250002, which is the identifier for Cron Status.

SELECT * FROM events WHERE deviceType= '11' AND qid= '44250002'

GROUP BY Clause

Use the GROUP BY clause to aggregate your data by one or more columns. To provide meaningful results of the aggregation, usually, data aggregation is combined with aggregatefunctions on remaining columns.

Examples Of GROUP BY Clauses

The following query example shows IP addresses that sent more than 1 million bytes within all flows in a specific time.

```
SELECT sourceIP, SUM(sourceBytes)
FROM flows where sourceBytes > 1000000
GROUP BY sourceIP
```

The results might look similar to the following output.

```
-----------------------------------
| sourceIP | SUM_sourceBytes |
-----------------------------------
| 192.0.2.0 | 4282590.0 |
| 198.51.100.0 | 4902509.0 |
| 203.0.113.0 | 2802715.0 |
| 203.0.113.1 | 3313370.0 |
| 198.51.100.1 | 2467183.0 |
| 198.51.100.2 | 8325356.0 |
| 203.0.113.2 | 1629768.0 |
-----------------------------------
```

However, if you compare this information to a non-aggregated query, the output displays all the IP addresses that are unique, as shown in the following output:

```
------------------------------
| sourceIP | sourceBytes |
------------------------------
| 192.0.2.0 | 1448629 |
| 198.51.100.0 | 2412426 |
| 203.0.113.0 | 1793095 |
| 203.0.113.1 | 1449148 |
| 198.51.100.1 | 1097523 |
| 198.51.100.2 | 4096834 |
| 192.0.2.1 | 2833961 |
| 198.51.100.3 | 2490083 |
| 203.0.113.2 | 1629768 |
| 203.0.113.3 | 1009620 |
| 198.51.100.4 | 1369660 |
| 203.0.113.4 | 1864222 |
| 198.51.100.5 | 4228522 |
------------------------------
```

To view the maximum number of events, use the following syntax:

```
SELECT MAX(eventCount) FROM events
```

To view the number of average events from a source IP, use the following syntax:

```
SELECT AVG(eventCount), PROTOCOLNAME(protocolid)
FROM events
GROUP BY sourceIP
```

The output displays the following results:

```
---------------------------------
| sourceIP | protocol |
---------------------------------
| 192.0.2.0 | TCP.tcp.ip |
| 198.51.100.0 | UDP.udp.ip |
| 203.0.113.0 | UDP.udp.ip |
| 203.0.113.1 | UDP.udp.ip |
| 198.51.100.1 | TCP.tcp.ip |
| 198.51.100.2 | TCP.tcp.ip |
| 192.0.2.1 | TCP.tcp.ip |
| 198.51.100.3 | ICMP.icmp.ip |
---------------------------------
```

HAVING Clause

Use the HAVING clause in a query to apply more filters to specific data by applying filters to the results after the GROUP BY clause.

The HAVING clause follows the GROUP BY clause.

You can apply the following filters when you use a `HAVING` clause in a query:

- Equal sign (=)
- Not equal to symbol (<>)
- Less than symbol (<)
- Greater than symbol (>)
- Less that or equal to symbol (<=)
- Greater than or equal to symbol (>=)

- BETWEEN between two values, for example (64 AND 512)
- LIKE case-sensitive match
- ILIKE case insensitive match
- SUM/AVG total or average values
- MAX/MIN maximum or minimum values

Examples Of HAVING Clauses

The following query example shows results for users who triggered VPN events from more than four IP addresses (HAVING 'Count of Source IPs' > 4) in the last 24 hours.

```
SELECT username, UNIQUECOUNT(sourceip) AS ’Count of Source IPs’
FROM events
WHERE LOGSOURCENAME(logsourceid) ILIKE ’%vpn%’
AND username IS NOT NULL
GROUP BY username
HAVING "Count of Source IPs" > 4
LAST 24 HOURS
```

NOTE: When you type an AQL query, use single quotation marks for a string comparison, and use double quotation marks for a property value comparison.

The following query example shows results for events where the credibility (HAVING credibility > 5) is greater than five.

```
SELECT username, sourceip, credibility
FROM events
GROUP BY sourceip
HAVING credibility > 5
LAST 1 HOURS
```

The following query groups results by source IP but displays only results where the magnitude (HAVING magnitude > 5) is greater than five.

```
SELECT sourceIP, magnitude
FROM events
```

```
GROUP BY sourceIP
HAVING magnitude > 5
```

ORDER BY Clause

Use the ORDER BY clause to sort the resulting view that is based on expression results. The result is sorted by ascending or descending order.

NOTE: When you type an AQL query, use single quotation marks for a string comparison, and use double quotation marks for a property value comparison.

You can use the ORDER BY clause on one or more columns.

Use the GROUP BY and ORDER BY clauses in a single query.

Sort in ascending or descending order by appending the ASC or DESC keyword to the ORDER BY clause.

Examples Of ORDER BY Clauses

To query AQL to return results in descending order, use the following syntax:

```
SELECT sourceBytes, sourceIP
FROM flows
WHERE sourceBytes > 1000000
ORDER BY sourceBytes DESC
```

To display results in ascending order, use the following syntax:

```
SELECT sourceBytes, sourceIP
FROM flows
WHERE sourceBytes > 1000000
ORDER BY sourceBytes ASC
```

To determine the top abnormal events or the most bandwidth-intensive IP addresses, you can combine GROUP BY and ORDER BY clauses in a single query. For example, the following query displays the most traffic intensive IP address in descending order:

```
SELECT sourceIP, SUM(sourceBytes)
FROM flows
GROUP BY sourceIP
ORDER BY SUM(sourceBytes) DESC
```

NOTE: When you use the `GROUP BY` clause with a column name or AQL function, only the first value is returned for the `GROUP BY` column, by default, even though other values might exist.

When you use a time field in the `ORDER BY` clause, use a simple datetime field, such as starttime. Using a formatted datetime field can impact the performance of the search.

LIKE Clause

Use the LIKE clause to retrieve partial string matches in the Ariel database.

You can search fields by using the LIKE clause.

The following table shows the wildcard options are supported by the Ariel Query Language (AQL).

Table 4: Supported Wildcard Options for LIKE Clauses

Wildcard character	Description
%	Matches a string of zero or more characters
_	Matches any single character

Examples Of LIKE Clauses

To match names such as Joe, Joanne, Joseph, or any other name that begins with Jo, type the following query:

```
SELECT * FROM events WHERE userName LIKE 'Jo%'
```

To match names that begin with Jo that are 3 characters long, such as, Joe or Jon, type the following query:

```
SELECT * FROM events WHERE userName LIKE 'Jo_'
```

You can enter the wildcard option at any point in the command, as shown in the following examples.

```
SELECT * FROM flows WHERE sourcePayload LIKE '%xyz'
SELECT * FROM events WHERE UTF8 (payload) LIKE '%xyz%'
SELECT * FROM events WHERE UTF8 (payload) LIKE '_yz'
```

Examples Of String Matching Keywords

The keywords, ILIKE and IMATCHES are case-insensitive versions of LIKE and MATCHES.

```
SELECT qidname(qid) as test FROM events WHERE test LIKE 'Information%'
SELECT qidname(qid) as test FROM events WHERE test ILIKE 'inForMatiOn%'
SELECT qidname(qid) as test FROM events WHERE test MATCHES '.*Information.*'
SELECT qidname(qid) as test FROM events WHERE test IMATCHES '.*Information.*'
```

COUNT Function

The COUNT function returns the number of rows that satisfy the WHERE clause of a SELECT statement.

If the SELECT statement does not have a WHERE clause, the COUNT function returns the total number of rows in the table.

Examples Of the Count Function

The following query returns the count of all events with credibility that is greater than or equal to 9.

```
SELECT COUNT(*) FROM events WHERE credibility >= 9
```

The following query returns the count of assets by location and source IP address.

```
SELECT ASSETPROPERTY('Location',sourceip)
AS location, COUNT(*)
FROM events
GROUP BY location
LAST 1 days
```

The following query returns the user names, source IP addresses, and count of events.

```
SELECT username, sourceip,
COUNT(*) FROM events
GROUP BY username
LAST 600 minutes
```

The `sourceip` column is returned as `FIRST_sourceip`.

One `sourceip` is returned only per `username`, even if another `sourceip` exists.

NOTE: When you use the `GROUP BY` clause with a column name or AQL function, only the first value is returned for the `GROUP BY` column, by default, even though other values might exist.

Quotation Marks

In an AQL query, query terms and queried columns sometimes require single or double quotation marks so that JSA can parse the query.

The following table defines when to use single or double quotation marks.

Table 5: Type Of Quotation Marks to Use in a Query

Type of quotation marks	When to use

Single	To specify any American National Standards Institute (ANSI) VARCHAR string to SQL such as parameters for a LIKE or equals (=) operator, or any operator that expects a VARCHAR string.
	
	`SELECT * from events WHERE sourceip = ’192.0.2.0’`
	`SELECT * from events WHERE userName LIKE ’%james%’`
	`SELECT * from events WHERE userName = ’james’`
	`SELECT * FROM events`
	`WHERE INCIDR(’10.45.225.14’, sourceip)`
	`SELECT * from events WHERE TEXT SEARCH ’my search term’`

Table 5: Type Of Quotation Marks to Use in a Query *(Continued)*

Type of quotation marks	When to use

Double	Use double quotation marks for the following query items to specify table and column names that contain spaces or non-ASCII characters, and to specify custom property names that contain spaces or non-ASCII characters.
	
	`SELECT "username column" AS ’User name’ FROM events`
	`SELECT "My custom property name"`
	`AS ’My new alias’ FROM events`
	
	Use double quotation marks to define the name of a system object such as field, function, database, or an existing alias.
	
	`SELECT "Application Category", sourceIP,`
	`EventCount AS ’Count of Events’`
	`FROM events GROUP BY "Count of Events"`
	
	Use double quotation marks to specify an existing alias that has a space when you use a WHERE, GROUP BY, or ORDER BY clause
	
	`SELECT sourceIP, destinationIP, sourcePort,`
	`EventCount AS ’Event Count’,`
	`category, hasidentity, username, payload, UtF8(payLoad),`
	`QiD, QiDnAmE(qid) FROM events`
	`WHERE (NOT (sourcePort <= 3003 OR hasidentity = ’True’))`
	`AND (qid = 5000023 OR qid = 5000193)`
	`AND (INCIDR(’192.0.2.0/4’, sourceIP)`
	`OR NOT INCIDR(’192.0.2.0/4’, sourceIP)) ORDER BY "Event Count"`
	`DESC LAST 60 MINUTES`
	
	`SSELECT sourceIP, destinationIP, sourcePort, EventCount`
	`AS ’Event Count’,`
	`category, hasidentity, username, payload, UtF8(payLoad),`
	`QiD, QiDnAmE(qid)`
	`FROM events ORDER BY "Event Count"`
	`DESC LAST 60 MINUTES`

Table 5: Type Of Quotation Marks to Use in a Query *(Continued)*

Type of quotation marks	When to use

Single or double	Use single quotation marks to specify an alias for a column definition in a query.
	
	`SELECT username AS 'Name of User', sourceip`
	`AS 'IP Source' FROM events`
	
	Use double quotation marks to specify an existing alias with a space when you use a WHERE, GROUP BY, or ORDER BY clause.
	
	`SELECT sourceIP AS 'Source IP Address',`
	`EventCount AS 'Event Count', QiD, QiDnAmE(qid)`
	`FROM events`
	`GROUP BY "Source IP Address"`
	`LAST 60 MINUTES`

Copying Query Examples from the AQL Guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

RELATED DOCUMENTATION

AQL Logical and Comparison Operators | 23

AQL Data Calculation and Formatting Functions | 27

AQL Data Aggregation Functions | 34

AQL Logical and Comparison Operators

IN THIS SECTION

Operators are used in AQL statements to determine any equality or difference between values. By using operators in the **WHERE clause of an AQL statement, the results are filtered by those results that match the conditions in the WHERE clause.**

The following table lists the supported logical and comparison operators.

Table 6: Logical and Comparison Operators

Operator	Description	Example
`*`	Multiplies two values and returns the result.	**SELECT * FROM flows WHERE sourceBytes * 1024 < 1**
`=`	The equal to operator compares two values and returns true if they are equal.	**SELECT * FROM EVENTS WHERE sourceIP = destinationIP**
`!=`	Compares two values and returns true if they are unequal.	**SELECT * FROM events WHERE sourceIP != destinationip**
`< AND <=`	Compares two values and returns true if the value on the left side is less than or equal to, the value on the right side.	**SELECT * FROM flows WHERE sourceBytes < 64 AND destinationBytes <= 64**

Table 6: Logical and Comparison Operators *(Continued)*

Operator	Description	Example
`> AND >=`	Compares two values and returns true if the value on the left side is greater than or equal to the value on the right side.	SELECT * FROM flows WHERE sourceBytes > 64 AND destinationBytes >= 64
`/`	Divides two values and returns the result.	SELECT * FROM flows WHERE sourceBytes / 8 > 64
`+`	Adds two values and returns the result.	SELECT * FROM flows WHERE sourceBytes + destinationBytes < 64
`-`	Subtracts one value from another and returns the result.	SELECT * FROM flows WHERE sourceBytes - destinationBytes > 0
`^`	Takes a value and raises it to the specified power and returns the result.	SELECT * FROM flows WHERE sourceBytes ^ 2 < 256
`%`	Takes the modulo of a value and returns the result.	SELECT * FROM flows WHERE sourceBytes % 8 == 7
`AND`	Takes the left side and right side of a statement and returns true if both are true.	SELECT * FROM events WHERE (sourceIP = destinationIP) AND (sourcePort = destinationPort)
`BETWEEN (X,Y)`	Takes in a left side and two values and returns true if the left side is between the two values.	SELECT * FROM events WHERE magnitude BETWEEN 1 AND 5
`COLLATE`	Parameter to order by that allows a BCP47 language tag to collate.	SELECT * FROM EVENTS ORDER BY sourceIP DESC COLLATE 'de-CH'

Table 6: Logical and Comparison Operators *(Continued)*

Operator	Description	Example
`INTO`	Creates a named cursor that contains results that can be queried at a different time.	SELECT * FROM EVENTS INTO 'MyCursor' WHERE....
`NOT`	Takes in a statement and returns true if the statement evaluates as false.	SELECT * FROM EVENTS WHERE NOT (sourceIP = destinationIP)
`ILIKE`	Matches if the string passed is LIKE the passed value and is not case sensitive. Use `%` as a wildcard.	SELECT * FROM events WHERE userName ILIKE '%bob%'
`IMATCHES`	Matches if the string matches the provided regular expression and is not case sensitive.	SELECT * FROM events WHERE userName IMATCHES '^.bob.$'
`LIMIT`	Limits the number of results to the provided number.	SELECT * FROM events LIMIT 100 START '2015-10-28 10:00' STOP '2015-10-28 11:00' **NOTE**: Place the LIMIT clause in front of a `START` and `STOP` clause.
`LIKE`	Matches if the string passed is `LIKE` the passed value but is case sensitive. Use `%` as a wildcard.	SELECT * FROM events WHERE userName LIKE '%bob%'
`MATCHES`	Matches if the string matches the provided regular expression.	SELECT * FROM events WHERE userName MATCHES '^.bob.$'
`NOT NULL`	Takes in a value and returns true if the value is not null.	SELECT * FROM events WHERE userName IS NOT NULL

Table 6: Logical and Comparison Operators *(Continued)*

Operator	Description	Example
`OR`	Takes the left side of a statement and the right side of a statement and returns true if either side is true.	**SELECT * FROM events WHERE (sourceIP = destinationIP) OR (sourcePort = destinationPort)**
`TEXT SEARCH`	Full-text search for the passed value. TEXT SEARCH is valid with `AND` operators. You can't use TEXT SEARCH with `OR` or other operators; otherwise, you get a syntax error. Place TEXT SEARCH in the first position of the WHERE clause. You can also do full-text searches by using the Quick filter in the JSA user interface. For information about Quick filter functions, see the *Juniper Secure Analytics Users Guide*.	**SELECT * FROM events WHERE TEXT SEARCH 'firewall' AND sourceip='192.168.1.1' SELECT sourceip,url FROM events WHERE TEXT SEARCH 'download.cdn.mozilla.net' AND sourceip='192.168.1.1' START '2015-01-30 16:10:12' STOP '2015-02-22 17:10:22'**

The following table lists the supported logical and comparison operators.

Examples Of Logical and Comparative Operators

- To find events that are not parsed, type the following query:

  SELECT * FROM events WHERE payload = 'false'

- To find events that return an offense and have a specific source IP address, type the following query:

  SELECT * FROM events WHERE sourceIP = '192.0.2.0' AND hasOffense = 'true'

- To find events that include the text "firewall", type the following query:

```
SELECT QIDNAME(qid) AS EventName, * FROM events WHERE TEXT SEARCH 'firewall'
```

RELATED DOCUMENTATION

AQL Data Calculation and Formatting Functions

IN THIS SECTION

Use Ariel Query Language (AQL) calculation and formatting functions on search results that are retrieved from the Ariel databases.

This list describes the AQL functions that are used for calculations and data formatting:

BASE64

- **Purpose--**Returns a Base64 encoded string that represents binary data.
- **Example--SELECT BASE64(payload) FROM events**

  Returns the payloads for events in BASE64 format.

CONCAT

- **Purpose--**Concatenates all passed strings into one string.

- **Example--SELECT CONCAT(username, ':', sourceip, ':', destinationip) FROM events LIMIT 5**

DATEFORMAT

- **Purpose**--Formats time in milliseconds since 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970 to a user-readable form.
- **Examples**--SELECT DATEFORMAT(startTime, 'yyyy-MM-dd hh:mm:ss') AS StartTime FROM events **SELECT DATEFORMAT(starttime,'yyyy-MM-dd hh:mm') AS 'Start Time', DATEFORMAT(endtime, 'yyyy-MM-dd hh:mm') AS Storage_time, QIDDESCRIPTION(qid) AS 'Event Name' FROM events**

  "AQL Date and Time Formats" on page 60

DOUBLE

- **Purpose**--Converts a value that represents a number into a double.
- **Example--DOUBLE('1234')**

LONG

- **Purpose**-- Converts a value that represents a number into a long integer.
- **Examples** --SELECT destinationip, LONG(SUM(sourcebytes+destinationbytes)) AS TotalBytes FROM flows GROUP BY sourceip

  The example returns the destination IP address, and the sum of the source and destination bytes in the TotalBytes column.

  SELECT LONG(sourceip) AS long_ip FROM events INTO <*cursor_name*> WHERE (long_ip & 0x<*ff*>000000) = 0x<*hexadecimal value of IP address*>000000 GROUP BY long_ip LIMIT 20

  In JSA 7.3.0, you can use the LONG function to convert IP addresses into a long integer. JSA uses long integers with bitwise operators to do IP address arithmetic and filtering in AQL queries. In the example, the source IP is returned as an integer, which is used by the bitwise AND operator.

  In the example, the <*ff*> corresponds with <*hexadecimal value of IP address*>, which is in the first octet position for an IP address. The <*cursor_name*> can be any name that you want to use.

For example, if you want to return all source IP addresses with the number 9 in the first octet, then substitute the hexadecimal value 9, which is the same as the decimal value, in <*hexadecimal value of IP address*>.

"Bitwise Operators in AQL Queries" on page 80

PARSEDATETIME

- **Purpose--**Pass a time value to the parser, for example, `PARSEDATETIME('time reference')`. The *time reference* indicates the parse time for the query.
- **Example--SELECT * FROM events START PARSEDATETIME('1 hour ago')**

  "Time Criteria in AQL Queries" on page 57

PARSETIMESTAMP

- **Purpose--**Parse the text representation of date and time and convert it to UNIX epoch time.

  For example, parse the following text date format:

  ```
  Thursday, August 24, 2017 3:30:32 PM GMT +01:00 and convert it to the following epoch
  timestamp: 1503588632.
  ```

  This function makes it easier to issue calls from the API that are based on scripts.

- **Example of how the time format conversion works--**The following example demonstrates how the DATEFORMAT function converts epoch time to a text timestamp by using the specified date format, and then the PARSETIMESTAMP function is used to convert the text timestamp to an epoch time format.

  ```
  SELECT starttime, DATEFORMAT(starttime,'EEE, MMM d, "yyyy"')
  AS "text time format",
  PARSETIMESTAMP('EEE, MMM d, "yyyy"', "text time format")
  AS 'epoch time returned' from events limit 5
  ```

The following example displays an extract of the output from the query:

```
starttime text time format epoch time returned
1503920389888 Mon, M08 28, "2017" 1503920389888
```

- **Example of how PARSETIMESTAMP might be used to convert times to epoch time so that time calculations can be made--**In the following example, events are returned when the time difference between logout and login times is less that 1 hour.

  The EEE, d MMM yyyy HH:mm:ss.SSSZ time format is just one example of a time format that you might use, and my_login and my_logout are custom properties in a known time format, for example, EEE, MMM d, "yy".

  ```
  SELECT * from events
  WHERE
  PARSETIMESTAMP('EEE, d MMM yyyy HH:mm:ss.SSSZ', my_logout)
  - PARSETIMESTAMP('EEE, d MMM yyyy HH:mm:ss.SSSZ', my_login)
  < 3600000 last 10 days
  ```

NOW

- **Purpose--**Returns the current time that is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.
- **Example--SELECT ASSETUSER(sourceip, NOW()) AS 'Asset user' FROM events**

  Find the user of the asset at this moment in time (NOW).

LOWER

- **Purpose--**Returns an all lowercase representation of a string.
- **Example--SELECT LOWER(username), LOWER(LOGSOURCENAME(logsourceid)) FROM events**

  Returns user names and log source names in lowercase.

REPLACEALL

- **Purpose**--Match a regex and replace all matches with text.

  Replaces every subsequence (*arg2*) of the input sequence that matches the pattern (*arg1*) with the replacement string (*arg3*).

- **Example--REPLACEALL('\d{16}', username, 'censored')**

REPLACEFIRST

- **Purpose**--Match a regex and replace the first match with text.

  Replaces the first subsequence (*arg2*) of the input sequence that matches the pattern (*arg1*) with the replacement string (*arg3*).

- **Example--REPLACEFIRST('\d{16}', username, 'censored')**

STR

- **Purpose**--Converts any parameter to a string.
- **Example--STR(sourceIP)**

STRLEN

- **Purpose**--Returns the length of this string.
- **Example--SELECT STRLEN(sourceIP), STRLEN(username) from events**

  Returns the string length for `sourceip` and `username`.

STRPOS

- **Purpose**--Returns the position (index - starts at zero) of a string in another string. Searches in string for the index of the specified substring. You can optionally specify an extra parameter to indicate at what position (index) to start looking for the specified pattern.

  The search for the string starts at the specified offset and moves towards the end of string.

  ```
  STRPOS(string, substring, index)
  ```

  Returns `-1` if the substring isn't found.

- **Examples**--SELECT STRPOS(username, 'name') FROM events **SELECT STRPOS(sourceip, '180', 2) FROM events)**

SUBSTRING

- **Purpose**--Copies a range of characters into a new string.
- **Examples**--SELECT SUBSTRING(userName, 0, 3) FROM events SELECT SUBSTRING(sourceip, 3, 5) FROM events

UPPER

- **Purpose**--Returns an all uppercase representation of a string.
- **Example--SELECT UPPER(username), UPPER(LOGSOURCENAME(logsourceid)) FROM events Returns user names and log source names in uppercase.**

UTF8

- **Purpose**--Returns the UTF8 string of a byte array.
- **Example--SELECT UTF8(payload) FROM events WHERE sourceip='192.0.2.0' Returns the UTF8 payload for events where the source IP address is 192.0.2.0**

RELATED DOCUMENTATION

AQL Data Aggregation Functions

IN THIS SECTION

Ariel Query Language (AQL) aggregate functions help you to aggregate and manipulate the data that you extract from the Ariel database.

Data Aggregation Functions

Use the following AQL functions to aggregate data, and to do calculations on the aggregated data that you extract from the AQL databases:

AVG

- **Purpose--**Returns the average value of the rows in the aggregate.
- **Example--SELECT sourceip, AVG(magnitude) FROM events GROUP BY sourceip**

COUNT

- **Purpose--**Returns the count of the rows in the aggregate.
- **Example--SELECT sourceip, COUNT(*) FROM events GROUP BY sourceip**

"COUNT Function" on page 18

DISTINCTCOUNT

- **Purpose**--Returns the unique count of the value in the aggregate. Uses the HyperLogLog+ approximation algorithm to calculate the unique count. Operates with a constant memory requirement and supports unlimited data sets.
- **Example--SELECT username, DISTINCTCOUNTCOUNT(sourceip) AS CountSrcIP FROM events GROUP BY username**

FIRST

- **Purpose**--Returns the first entry of the rows in the aggregate.
- **Example--SELECT sourceip, FIRST(magnitude) FROM events GROUP BY sourceip**

GROUP BY

- **Purpose**-- Creates an aggregate from one or more columns.

  To return values other than the default first value, use functions such as COUNT, MAX, AVG.
- **Examples** --SELECT sourceip, COUNT(*) FROM events GROUP BY sourceip, destinationip **SELECT username, sourceip, COUNT(*) FROM events GROUP BY username LAST 5 minutes**The `sourceip` column is returned as `FIRST_sourceip`. Only one `sourceip` is returned per `username`, even if another `sourceip` exists.**SELECT username, COUNT(sourceip), COUNT(*) FROM events GROUP BY username LAST 5 minutes**

  The `sourceip` column is returned as `COUNT_sourceip`. The count for `sourceip` results is returned per `username`.

  "GROUP BY Clause" on page 12

HAVING

- **Purpose**--Uses operators on the result of a grouped by column.

- **Example--SELECT sourceip, MAX(magnitude) AS MAG FROM events GROUP BY sourceip HAVING MAG > 5**

  "HAVING Clause" on page 14

  Saved searches that include the having clause and that are used for scheduled reports or time-series graphs are not supported.

LAST

- **Purpose**--Returns the last entry of the rows in the aggregate.
- **Example--SELECT sourceip, LAST(magnitude) FROM events GROUP BY sourceip**

MIN

- **Purpose**--Returns the minimum value of the rows in the aggregate.
- **Example--SELECT sourceip, MIN(magnitude) FROM events GROUP BY sourceip**

MAX

- **Purpose**--Returns the maximum value of the rows in the aggregate.
- **Example--SELECT sourceip, MAX(magnitude) FROM events GROUP BY sourceip**

STDEV

- **Purpose**--Returns the Sample Standard Deviation value of the rows in the aggregate.
- **Example--SELECT sourceip, STDEV(magnitude) FROM events GROUP BY sourceip**

STDEVP

- **Purpose**--Returns the Population Standard Deviation value of the rows in the aggregate.
- **Example**--SELECT sourceip, STDEVP(magnitude) FROM events GROUP BY sourceip

SUM

- **Purpose**--Returns the sum of the rows in the aggregate.
- **Example**--SELECT sourceip, SUM(sourceBytes) FROM flows GROUP BY sourceip

UNIQUECOUNT

- **Purpose**--Returns the unique count of the value in the aggregate.
- **Example**--SELECT username, UNIQUECOUNT(sourceip) AS CountSrcIP FROM events GROUP BY sourceip

RELATED DOCUMENTATION

AQL Data Retrieval Functions | 38
Time Criteria in AQL Queries | 57
AQL Date and Time Formats | 60

AQL Data Retrieval Functions

IN THIS SECTION

Use the Ariel Query Language (AQL) built-in functions to retrieve data by using data query functions, and field ID properties from the Ariel database.

Use the following AQL functions to extract data from the AQL databases:

Data Retrieval Functions

APPLICATIONNAME

- **Purpose**--Returns flow application names by application ID
- **Parameters**--Application ID
- **Example** --SELECT APPLICATIONNAME(applicationid) AS 'Name of App' FROM flows

  Returns the names of applications from the flows database. These application names are listed in the **Name of App** column, which is an alias.

ARIELSERVERS4EPID

- **Purpose--**Use the ARIELSERVERS4EPID function to specify the Event Processor ID when you use it with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS.

  The domain can optionally be specified to target an asset on a particular domain.

- **Parameters**--ARIELSERVERS4EPID(processor_ID)

The following examples show how to use the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(processor_ID)
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(processor_ID)
```

- **Examples--**In the following example, only the search results from ARIELSERVERS4EPID(8) are included in the output. If the processor ID that you specify as a parameter for the ARIELSERVERS4EPID function is not in your JSA deployment, then the query does not run.

```
SELECT ARIELSERVERS4EPID(8), ARIELSERVERS4EPID(11), processorid,
PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
LAST 20 MINUTES
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(8)
```

You can also use the ARIELSERVERS4EPID function to returns the Ariel servers that are connected to a specific Event Processor that is identified by ID, as shown in the following example:

```
SELECT processorid, PROCESSORNAME(processorid),
ARIELSERVERS4EPID(processorid)
FROM events GROUP BY processorid
```

ARIELSERVERS4EPNAME

- **Purpose--**You use the ARIELSERVERS4EPNAME function to specify the Event Processor name when you use it with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS.
- **Parameters--**ARIELSERVERS4EPNAME('eventprocessor_name')

The following examples show how you use ARIELSERVERS4EPNAME PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor255')
```

- **Example** --In the following example, records from servers that are associated with eventprocessor104 are excluded from the search.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

  You can also use the function to return Ariel servers that are associated with an Event Processor that is identified by name.

```
SELECT PROCESSORNAME(processorid),
ARIELSERVERS4EPNAME(PROCESSORNAME(processorid))
FROM events GROUP BY processorid
```

Returns Ariel servers for the named Event Processor.

ASSETHOSTNAME

- **Purpose--**Searches for the host name of an asset at a point in time.

  The domain can optionally be specified to target an asset on a particular domain.

```
ASSETHOSTNAME(sourceip)
ASSETHOSTNAME(sourceip, NOW())
ASSETHOSTNAME(sourceip, domainid)
```

- **Parameters--**IP address, (timestamp and domain ID are optional)

  If the time stamp is not specified, the current time is used.

- **Example**

```
SELECT ASSETHOSTNAME(destinationip, NOW())
AS 'Host Name'
FROM events
SELECT ASSETHOSTNAME(sourceip, NOW())
```

```
AS 'Host Name'
FROM events
```

Returns the host name of the asset at the time of the query.

ASSETPROPERTY

- **Purpose**--Looks up a property for an asset.

  The domain can optionally be specified to target an asset on a particular domain.

  ASSETPROPERTY ('Unified Name', sourceIP, domainId)

- **Parameters**--Property name, IP address

  Domain ID is optional

- **Example** --SELECT ASSETPROPERTY('Location',sourceip) AS Asset_location, COUNT(*) AS 'event count' FROM events GROUP BY Asset_location LAST 1 days

  Returns the asset location that is affiliated with the source IP address.

ASSETUSER

- **Purpose**--Searches for the user of an asset at a point in time.

  Domain can optionally be specified to target an asset in a specific domain.

  ASSETUSER(sourceIP,NOW(), domainId)

- **Parameters**--IP address, (timestamp and domain ID are optional)If the time stamp is not specified, the current time is used.

- **Example** --SELECT ASSETUSER(sourceip, now()) AS 'Username of Asset' FROM events

  Returns the user name that is affiliated with the source IP address.

CATEGORYNAME

- **Purpose**--Searches for the name of a category by the category ID.

CATEGORYNAME(Category)

- **Parameters**--Category
- **Example** --SELECT sourceip, category, CATEGORYNAME(category) AS 'Category name' FROM events

  Returns the source IP, category ID, and category name

DOMAINNAME

- **Purpose--**Searches for the domain name by the domain ID.

  DOMAINNAME(domainID)
- **Parameters**--Domain ID
- **Example** --SELECT sourceip, username, DOMAINNAME(domainid) AS 'Domain name' FROM events

  Returns source IP, user name, and domain names from events database

GLOBALVIEW

- **Purpose--**Returns the GLOBALVIEW database results for a given saved search name based on the time range that is input.

  This query can be run only by using API.

  For more information about accessing a GLOBALVIEW database, see the *Juniper Secure Analytics Administration Guide*.
- **Parameters**--Saved search, time range (DAILY, NORMAL, HOURLY)
- **Example** --SELECT * FROM GLOBALVIEW ('Top Log Sources','DAILY') LAST 2 days

GEO::LOOKUP

- **Purpose--**Returns location data, provided by MaxMind, for a selected IP address.
- **Parameters**--IP address (required)

Strings (at least one required):

```
city, continent, physical_country, registered_country, represented_country, location, postal,
subdivisions, traits, geo_json
```

- **Example**

```
SELECT sourceip, GEO::LOOKUP(sourceip, 'city')
AS GEO_CITY
FROM events last 10 minutes
```

GEO::DISTANCE

- **Purpose**--Returns the distance, in kilometers, of two IP addresses.
- **Parameters**--IP address (two required)
- **Example**

```
SELECT GEO::DISTANCE(sourceip, destinationip)
AS GEO_DISTANCE
FROM events last 10 minutes
```

HOSTNAME

- **Purpose**--Returns the host name of an event processor with a certain processor ID.

  HOSTNAME (processorId)
- **Parameters**--Processor ID
- **Example**--SELECT HOSTNAME (processorId) FROM events

INCIDR

- **Purpose**--Filters the output of the SELECT statement by referencing the source/destination CIDR IP address that is specified by INCIDR.
- **Parameters**--IP/CIDR, IP address
- **Example** --SELECT sourceip, username FROM events WHERE INCIDR('172.16.0.0/16', sourceip)

  Returns the source IP and user name columns from the flows database where the source CIDR IP address is from the 172.16.0.0/16 subnet.

  "CIDR IP Addresses in AQL Queries" on page 86

INOFFENSE

- **Purpose**--If an event or flow belongs to the specified offense, it returns `true`.
- **Parameters**--Offense ID
- **Example** --SELECT * FROM events WHERE InOffense(123)**SELECT * FROM flows WHERE InOffense(123)**

LOGSOURCENAME

- **Purpose**--Looks up the name of a log source by its log source ID.

  LOGSOURCENAME(logsourceid)
- **Parameters**--Log source ID
- **Example** --SELECT * FROM events WHERE LOGSOURCENAME(logsourceid) ILIKE '%mylogsourcename%'

  Returns only results that include `mylogsourcename` in their log source name.

  SELECT LOGSOURCENAME(logsourceid) AS Log_Source FROM events

  Returns the column alias **Log_source**, which shows log source names from the events database.

LOGSOURCEGROUPNAME

- **Purpose**--Searches for the name of a log source group by its log source group ID.

  ```
  LOGSOURCEGROUPNAME(deviceGroupList)
  ```

- **Parameters**--Device group list
- **Example** --SELECT sourceip, logsourceid FROM events WHERE LOGSOURCEGROUPNAME(devicegrouplist) ILIKE '%other%'

  Returns the source IP address and log source IDs for log source groups that have 'other' in their name.

LOGSOURCETYPENAME

- **Purpose**--Searches for the name of a log source type by its device type.

  ```
  LOGSOURCETYPENAME(deviceType)
  ```

- **Parameters**--Device type
- **Example** --SELECT LOGSOURCETYPENAME(logsourceid) AS 'Device names', COUNT(*) FROM events GROUP BY "Device names" LAST 1 DAYS

  Returns device names and the event count.

  All log sources functions example:

  SELECT logsourceid, LOGSOURCENAME(logsourceid) AS 'Name of log source', LOGSOURCEGROUPNAME(devicegrouplist) AS 'Group Names', LOGSOURCETYPENAME(devicetype) AS 'Devices' FROM events GROUP BY logsourceid

  Returns log source names, log source group names, and log source device names.

  When you use the `GROUP BY` function, the first item only in the `GROUP BY` list is shown in the results.

MATCHESASSETSEARCH

- **Purpose**--If the asset is returned in the results of the saved search, it returns true. **MATCHESASSETSEARCH ('My Saved Search', sourceIP)**
- **Parameters**--Saved Search Name, IP address

- **Example --MATCHESASSETSEARCH ('My Saved Search', sourceIP)**

NETWORKNAME

- **Purpose**--Searches for the network name from the network hierarchy for the host that is passed in.

  NetworkName(sourceip)

  The domain can optionally be specified to target a network in a particular domain.

  NETWORKNAME(sourceip, domainId)

- **Parameters**--Host property (domain is optional)
- **Examples--SELECT NETWORKNAME(sourceip) ILIKE 'servers' AS 'My Networks' FROM flows**

  Returns any networks that have the name servers.

  SELECT NETWORKNAME(sourceip, domainID) ILIKE 'servers' AS 'My Networks' FROM flows

  Returns any networks that have the name servers in a specific domain.

  SELECT NETWORKNAME(sourceip) AS 'Src Net', NETWORKNAME(destinationip) AS Dest_net FROM events

  Returns the network name that is associated with the source and destination IP addresses.

OFFENSE_TIME

- **Purpose**--Limits the query to applicable times that an offense could be active.

  This function increases the speed of the query.

- **Parameters**--Offense ID.
- **Example--SELECT * FROM events WHERE INOFFENSE(1) times OFFENSE_TIME(1)**

PARAMETERS EXCLUDESERVERS

- **Purpose**--Filters search criteria by excluding the specified servers.
- **Parameters**--[Server IP address:Port number]

Use port 32006 for an Event Processor, and port 32011 for a Console.

Parameters accept a comma-separated list of arguments. For example,

```
"host1:port1,host2:port2,host3:port3".
```

- **Example** --In the following example, search results from 192.0.2.0 are excluded. To exclude a Console, you must use localhost or 127.0.0.1. Do not use the IP address of the Console in this query.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid)
from events
GROUP BY logsourceid
PARAMETERS EXCLUDESERVERS='192.0.2.0:32006'
```

In the following example, search results from the Console are excluded:

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) FROM events
GROUP BY logsourceid start '2017-03-15 10:26'
STOP '2017-03-15 10:30'
PARAMETERS EXCLUDESERVERS='127.0.0.1:32011'
```

In the following example, search results from the Console are excluded. The Console is referred to as localhost in this example.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid start '2017-03-15 10:25'
STOP '2017-03-15 10:30'
PARAMETERS EXCLUDESERVERS='localhost:32011'
```

The following example uses multiple arguments to exclude search results from the Console and two other servers.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid start '2017-04-15 10:25'
```

```
STOP '2017-04-15 10:30'
PARAMETERS EXCLUDESERVERS='127.0.0.1:32011,192.0.2.0:32006,172.11.22.31:32006'
```

Specify the ID of the Event Processor in your query by using the following function:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(processor_ID)'
```

Refine your query by using ARIELSERVERS4EPID with PARAMETERS EXCLUDESERVERS to specify the Event Processor ID that you want to exclude from your search. You can specify one or more Event Processor IDs.

- **Example** --In the following example, all results from ARIELSERVERS4EPID(8) are excluded in the search.

```
SELECT processorid,
PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
LAST 20 MINUTES
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(8)
```

Specify the name of the Event Processor in your query by using the following function:

```
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('processor_name')
```

Refine your query by using ARIELSERVERS4EPNAME with PARAMETERS EXCLUDESERVERS to specify the Event Processor by name. You can specify one or more Event Processor names.

- **Example** -- In the following example, records from servers that are associated with eventprocessor104 are excluded from the search.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

PARAMETERS REMOTESERVERS

- **Purpose--**Use the PARAMETERS REMOTESERVERS function to narrow your search to specific servers, which speeds up your search by not searching all hosts.
- **Parameters--**[Server IP address:Port number]

  Use port 32006 for an Event Processor, and port 32011 for a Console.

  Use a comma-separated list for multiple arguments, for example,

  ```
  "host1:port1,host2:port2,host3:port3".
  ```

- **Example --**In the following example, only the specified server is searched.

  ```
  SELECT * FROM EVENTS START '2016-09-08 16:42'
  STOP '2016-09-08 16:47'
  PARAMETERS REMOTESERVERS='192.0.2.0:32006'
  ```

  In the following example, multiple servers are specified, which includes search results from the Console and two other servers.

  ```
  SELECT processorid,PROCESSORNAME(processorid),
  LOGSOURCENAME(logsourceid) from events
  GROUP BY logsourceid start '2017-04-15 10:25'
  STOP '2017-04-15 10:30'
  PARAMETERS REMOTESERVERS='127.0.0.1:32011,192.0.2.0:32006,172.11.22.31:32006'
  ```

  Specify the ID of the Event Processor in your query by using the following function:

  ```
  PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(processor_ID)
  ```

  Refine your query by using ARIELSERVERS4EPID with PARAMETERS REMOTESERVERS to specify the ID of the Event Processor that you want to include in your search. You can specify one or more Event Processor IDs.

- **Example** --In the following example, only the search results from ARIELSERVERS4EPID(8) are included in the output.

```
SELECT ARIELSERVERS4EPID(8), ARIELSERVERS4EPID(11), processorid,
PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid) from events
GROUP BY logsourceid
LAST 20 MINUTES
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(8)
```

NOTE: If the processor ID that you specify as a parameter for the ARIELSERVERS4EPID function is not in your JSA deployment, then the query does not run.

Specify the name of the Event Processor in your query by using the following function:

```
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor_name')
```

Refine your query by using ARIELSERVERS4EPNAME and PARAMETERS REMOTESERVERS to specify the name of the Event Processor that you want to include in your search. You can specify one or more Event Processor names.

- **Example** -- In the following example, only search records that are associated with eventprocessor104 are included in the search results.

```
SELECT processorid,PROCESSORNAME(processorid),
LOGSOURCENAME(logsourceid)
FROM events
GROUP BY logsourceid
PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104')
```

PROCESSORNAME

- **Purpose**--Returns the name of a processor by the processor ID.

```
PROCESSORNAME(processorid)
```

- **Parameters**--Processor ID number
- **Example**

```
SELECT sourceip, PROCESSORNAME(processorid)
AS ’Processor Name’
FROM events)
```

  Returns the source IP address and processor name from the events database.

- **Example**

```
SELECT processorid, PROCESSORNAME(processorid)
FROM events WHERE processorid=104
GROUP BY processorid LAST 5 MINUTES
```

  Returns results from the Event Processor that has a processor ID equal to 104.

PROTOCOLNAME

- **Purpose**--Returns the name of a protocol by the protocol ID
- **Parameters**--Protocol ID number
- **Example --SELECT sourceip, PROTOCOLNAME(protocolid) AS 'Name of protocol' FROM events**

  Returns the source IP address and protocol name from the events database.

QIDNAME

- **Purpose**--Searches for the name of a QID by its QID.

  QIDNAME(qid)

- **Parameters**--QID
- **Example** --SELECT QIDNAME(qid) AS 'My Event Names', qid FROM events

  Returns QID name and QID number.

QIDESCRIPTION

- **Purpose--**Searches for the QID description by its QID.

  QIDDESCRIPTION(qid)

- **Parameters--**QID

- **Example --SELECT QIDDESCRIPTION(qid) AS 'My_Event_Names', QIDNAME(qid) AS 'QID Name' FROM events**

  Returns QID description and QID name.

REFERENCEMAP

- **Purpose--**Searches for the value for a key in a reference map.

  ReferenceMap('Value',Key,domainID)

  Although the `domainID` is optional, in a domain-enabled environment, the search is limited to only shared reference data when the `domainID` is excluded.

- **Parameters--**String, String, Integer

- **Example --SELECT REFERENCEMAP('Full_name_lookup', username, 5) AS Name_of_User FROM events**

  Searches for the `userName` (key) in the `Full_name_lookup` reference map in the specified domain, and returns the full name (value) for the user name (key).

REFERENCEMAPSETCONTAINS

- **Purpose--**If a value exists for a key in a reference map of sets, for a domain, it returns `true`.

  REFERENCEMAPSETCONTAINS(MAP_SETS_NAME, KEY, VALUE)

- **Parameters--**String, String, String

- **Example --ReferenceMapSetContains('RiskyUsersForIps','sourceIP','userName')**

REFERENCETABLE

- **Purpose**--Searches for the value of a column key in a table that is identified by a table key in a specific reference table collection.

  REFERENCETABLE ('testTable','value','key', domainID) or REFERENCETABLE ('testTable','value','key' domainID)

  Although the `domainID` is optional, in a domain-enabled environment, the search is limited to only shared reference data when the `domainID` is excluded.

- **Parameters**--String, String, String (or IP address), Integer

- **Example --SELECT REFERENCETABLE('user_data','FullName',username, 5) AS 'Full Name', REFERENCETABLE('user_data','Location',username, 5) AS Location, REFERENCETABLE('user_data','Manager',username, 5) AS Manager FROM events**

  Returns the full name (value), location (value), and manager (value) for the `username` (key) from `user_data`.

  "Reference Data Query Examples " on page 94

REFERENCESETCONTAINS

- **Purpose**--If a value is contained in a specific reference set, it returns `true`.

  REFERENCESETCONTAINS ('Ref_Set', 'value', domainID)

  Although the `domainID` is optional, in a domain-enabled environment, the search is limited to only shared reference data when the `domainID` is excluded.

- **Parameters**--String, String, Integer

- **Example --SELECT ASSETUSER(sourceip, NOW()) AS 'Source Asset User' FROM flows WHERE REFERENCESETCONTAINS('Watchusers', username, 5) GROUP BY "Source Asset User" LAST 24 HOURS**

  Returns the asset user when the `username` (value) is included in the `Watchusers` reference set.

RULENAME

- **Purpose**--Returns one or more rule names that are based on the rule ID or IDs that are passed in.

RULENAME(creeventlist)

RULENAME(3453)

- **Parameters--**A single rule ID, or a list of rule IDs.
- **Example --SELECT * FROM events WHERE RULENAME(creEventList) ILIKE '%my rule name%'**

  Returns events that trigger a specific rule name.

  SELECT RULENAME(123) FROM events

  Returns rule name by the rule ID.

RELATED DOCUMENTATION

Time Criteria in AQL Queries

IN THIS SECTION

Define time intervals in your AQL queries by using START and STOP clauses, or use the LAST clause for relative time references.

Define the Time Settings That Are Passed to the AQL Query

The SELECT statement supports an arieltime option, which overrides the time settings.

You can limit the time period for which an AQL query is evaluated by using the following clauses and functions:

- "START" on page 58
- "STOP" on page 59
- "LAST" on page 59
- "NOW" on page 60
- "PARSEDATETIME" on page 60

START

You can pass a time interval to `START` selecting data (from time), in the following formats:

yyyy-MM-dd HH:mm yyyy-MM-dd HH:mm:ss yyyy/MM/dd HH:mm:ss yyyy/MM/dd-HH:mm:ss yyyy:MM:dd-HH:mm:ss

The *timezone* is represented by '`z` or `Z`' in the following formats:

yyyy-MM-dd HH:mm'Z' yyyy-MM-dd HH:mm'z'Use `START` in combination with `STOP`.

- **Examples--SELECT * FROM events WHERE userName IS NULL START '2014-04-25 15:51' STOP '2014-04-25 17:00'**

  Returns results from: 2014-04-25 15:51:00 to 2014-04-25 16:59:59

  SELECT * FROM events WHERE userName IS NULL START '2014-04-25 15:51:20' STOP '2014-04-25 17:00:20' Returns results from: 2014-04-25 15:51:00 to 2014-04-25 17:00:59**SELECT * from events START PARSEDATETIME('1 hour ago') STOP PARSEDATETIME('now')**`STOP` is optional. If you don't include it in the query, the STOP time is = `now`

STOP

You can pass a time interval to `STOP` selecting data (end time), in the following formats: **yyyy-MM-dd HH:mm yyyy-MM-dd HH:mm:ss yyyy/MM/dd HH:mm:ss yyyy/MM/dd-HH:mm:ss yyyy:MM:dd-HH:mm:ss**

The *timezone* is represented by '`z` or `Z`' in the following formats:

yyyy-MM-dd HH:mm'Z' yyyy-MM-dd HH:mm'z'Use `STOP` in combination with `START`.

- Examples--

  SELECT * FROM events WHERE username IS NULL START '2016-04-25 14:00' STOP '2016-04-25 16:00'

  SELECT * FROM events WHERE username IS NULL START '2016-04-25 15:00:30' STOP '2016-04-25 15:02:30'

  Use any format with the `PARSEDATETIME` function, for example,**SELECT * FROM events START PARSEDATETIME('1 day ago')** Even though `STOP` is not included in this query, the STOP time is = `now`.

  Select * FROM events START PARSEDATETIME('1 hour ago') STOP PARSEDATETIME('now')

  SELECT * FROM events START PARSEDATETIME('1 day ago')

  Select * FROM events WHERE logsourceid = '69' START '2016-06-21 15:51:00' STOP '2016-06-22 15:56:00'

LAST

You can pass a time interval to the `LAST` clause to specify a specific time to select data from.

The valid intervals are MINUTES, HOURS, and DAYS

- Examples--

  SELECT * FROM events LAST 15 MINUTES

  SELECT * FROM events LAST 2 DAYS

  SELECT * from events WHERE userName ILIKE '%dm%' LIMIT 10 LAST 1 HOURS

NOTE: If you use a LIMIT clause in your query, you must place it before START and STOP clauses, for example,

```
SELECT * FROM events LIMIT 100 START '2016-06-28 10:00' STOP '2016-06-28 11:00'
```

Time Functions

Use the following time functions to specify the parse time for the query.

NOW

- **Purpose--**Returns the current time that is expressed as milliseconds since the time 00:00:00 Coordinated Universal Time (UTC) on January 1, 1970.
- **Example--SELECT ASSETUSER(sourceip, NOW()) AS 'Asset user' FROM events**Find the user of the asset at this moment in time (NOW).

PARSEDATETIME

- **Purpose--**Pass a time value to the parser, for example, `PARSEDATETIME('time reference')`. This `'time reference'` is the parse time for the query.
- **Example--SELECT * FROM events START PARSEDATETIME('1 hour ago')**

RELATED DOCUMENTATION

AQL Date and Time Formats | **60**

AQL Subquery | **65**

Grouping Related Events Into Sessions | **66**

AQL Date and Time Formats

Use Ariel Query Language (AQL) date and time formats to represent times and dates in queries.

The following table lists the letters that represent date and time in AQL queries. This table is based on the *SimpleDateFormat*.

Table 7: Date and Time Formats

Letter	Date or time parameter	Presentation	Examples

			`DATEFORMAT(starttime,'yy-MM-dd')`
			
			Returns date format: 16-06-20
			
			`DATEFORMAT(starttime,'yyyy-MM-dd')`
			
			Returns date format: 2016-06-20
			
			SELECT DATEFORMAT(devicetime,'yyyy-MM-dd') AS Log_Src_Date, QIDDESCRIPTION(qid) AS 'Event Name' FROM events

			`DATEFORMAT(starttime,'YY-MM-dd')`
			
			Returns date format: 16-06-20
			
			`DATEFORMAT(starttime,'YYYY-MM-dd')`
			
			Returns date format: 2016-06-20
			
			SELECT DATEFORMAT(starttime,'YYYY-MM-dd hh:mm') AS 'Start Time', DATEFORMAT(endtime,'YYYY-MM-dd hh:mm') AS Storage_time, QIDDESCRIPTION(qid) AS 'Event Name' FROM events
			
			Returns start time, storage time, and event name columns

			`DATEFORMAT(starttime,'yyyy-MMMM-dd')`
			
			Returns date format: 2016-June-20
			
			`DATEFORMAT(starttime,'yyyy-MMM-dd')`
			
			Returns date format: 2016-Jun-20
			
			`DATEFORMAT(starttime,'yyyy-MM-dd')`
			
			Returns date format: 2016-06-20

Table 7: Date and Time Formats *(Continued)*

Letter	Date or time parameter	Presentation	Examples
			`DATEFORMAT(starttime,'yyyy-ww-dd')` Returns date format: 2016-26-20 **NOTE**: 26 is week 26 in year
			`DATEFORMAT(starttime,'yyyy-WW-dd')` Returns date format: 2016-04-20 **NOTE**: 04 is week 4 in month
			`DATEFORMAT(starttime,'yyyy-mm-DD')` Returns date format: 2016-06-172 **NOTE**: 172 is day number 172 in year
			`DATEFORMAT(starttime,'yyyy-mm-dd')` Returns date format: 2016-06-20
			`DATEFORMAT(starttime,'yyyy-MM-FF')` Returns date format: 2016-06-03 **NOTE**: 03 is day 3 of week in month
			`DATEFORMAT(starttime,'yyyy-MM-EE')` Returns date format: 2016-06-Mon
			`DATEFORMAT(starttime,'yyyy-MM-dd h a')` 2016-06-20 06 PM

Table 7: Date and Time Formats *(Continued)*

Letter	Date or time parameter	Presentation	Examples
			`DATEFORMAT(starttime,'yyyy-MM-dd H')` Returns date format: 2016-06-20 18 **NOTE**: 18 is 18:00 hours
			`DATEFORMAT(starttime,'yyyy-MM-dd k')` Returns date format: 2016-06-20 18 **NOTE**: 18 is 18:00 hours
			`DATEFORMAT(starttime,'yyyy-MM-dd K a')` Returns date format: 2016-06-20 6 PM **NOTE**: K = 6 and a = PM
			`DATEFORMAT (starttime,'yyyy-MM-dd h a')` Returns date format: 2016-06-20 6 PM **NOTE**: h = 6 and a = PM
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m a')` Returns date format: 2016-06-20 6:10 PM **NOTE**: colon added in query to format time
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m:s a')` Returns date format: 2016-06-20 6:10:56 PM **NOTE**: colons added in query to format time

Table 7: Date and Time Formats *(Continued)*

Lette r	Date or time parameter	Presentation	Examples
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m:ss:SSS a')` Returns date format: 2016-06-20 6:10:00:322 PM **NOTE**: colons added in query to format time
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m a z')` Returns date format: 2016-06-20 6:10 PM GMT + 1 **NOTE**: colon added in query to format time
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m a Z')` Returns date format: 2016-06-20 6:10 PM + 0100 **NOTE**: colon added in query to format time
			`DATEFORMAT(starttime,'yyyy-MM-dd h:m a X')` Returns date format: 2016-06-20 6:10 PM + 01 **NOTE**: colon added in query to format time

RELATED DOCUMENTATION

AQL Subquery | 65
Grouping Related Events Into Sessions | 66
Conditional Logic in AQL Queries | 79

AQL Subquery

IN THIS SECTION

Use an AQL subquery as a data source that is referred to, or searched by the main query. Use the FROM or IN clause to refine your AQL query by referring to the data that is retrieved by the subquery.

A *subquery* is a nested or inner query that is referenced by the main query. A subquery is accessible only by using API and is not yet available for use in searches from the **Log Activity** or **Network Activity** tabs. The subquery is available in the following formats:

- `SELECT` <*field/s*> `FROM` (<*AQL query expression*>)

  This query uses the `FROM` clause to search the output (cursor) of the subquery.

- `SELECT` <*field/s*> `FROM` events `WHERE` <*field*> `IN` (<*AQL query expression*>)

  This query uses the `IN` clause to specify the subquery results that match values from the subquery search. This subquery returns only one column. You can specify the results limit but the maximum is 10,000 results.

Subquery Examples

The nested SELECT statement in parenthesis is the subquery. The subquery is run first and it provides the data that is used by the main query. The main query SELECT statement retrieves the user names from the output (cursor) of the subquery.

SELECT username FROM (SELECT * FROM events WHERE username IS NOT NULL LAST 60 MINUTES)

The following query returns records where the user name from the Ariel database matches values in the subquery. **SELECT * FROM events WHERE username IN (SELECT username FROM events LIMIT 10 LAST 5 MINUTES) LAST 24 HOURS**

The following query returns records where the source IP address from the Ariel database matches the destination IP address in the subquery. **SELECT * FROM EVENTS WHERE sourceip IN (SELECT destinationip FROM events)**

The following query returns records where the source IP address from the Ariel database matches the source IP addresses that are returned in the subquery. The subquery filters the data for the main select statement by locating internal hosts that interacted with high-risk entities. The query returns hosts that communicated with any hosts that interacted with high-risk entities. **SELECT sourceip AS 'Risky Hosts' FROM events WHERE destinationip IN (SELECT sourceip FROM events WHERE eventdirection = 'L2R' AND REFERENCESETCONTAINS('CriticalWatchList', destinationip) GROUP BY sourceip) GROUP BY sourceip last 24 hours**

RELATED DOCUMENTATION

Grouping Related Events Into Sessions | 66

Conditional Logic in AQL Queries | 79

Bitwise Operators in AQL Queries | 80

Grouping Related Events Into Sessions

IN THIS SECTION

Group events that are contextually related into sessions where you can observe event sequences and the outcomes of those event sequences. Gain insight into user activity and network activity by observing the sequence of events that occur in a session.

You can use events to tell you what a user did at a specific time, but you can use transactional sessions to tell you what the user did before and after an event. Transactions give you full detail such as a purchase on the Internet, or an unauthorized login attempt.

The session ID is unique and is assigned to events in the same session. You define the session based on parameters such as time, user name, login, or any other criteria. You use the SESSION BY clause to create the unique sessions.

For example, use the transactional sessions to do these tasks:

- Define a user activity based on web-access events that includes a unique combination of activities.
- Group events by a specific user behavior session such as website visits, downloads, or emails sent.

- Record when users login to and logout of your network, and how long they log in for. The logout closes the related transaction that is initiated by the login.
- Pick an activity that you want to track and define the criteria for the session activity.

1. To create sessions, use the SESSION BY clause by using the following format.

   SESSION BY <*TimeExpression*> <*AQL_expression_list*> BEGIN <*booleanExpression*> END <booleanExpression>

   The following table describes the session parameters.

   Table 8:

Session parameters	Description
Time <*TimeExpression*>	Time
<*AQL_expression_list*>	AQL expression list
BEGIN <*booleanExpression*>	Starts a new session
END <*booleanExpression*>	The END clause is optional, and is used to finish the session.

   The **SessionId** changes when any AQL expression value changes or when the BEGIN or END *booleanExpression* is TRUE.

2. To test an example, take the following steps:

   a. To go to the JSA **API documentation** page, from the **Help** menu, click **Interactive API for Developers**.

   b. Click **8.0** or the highest version to expand the menu.

   c. Click **/ariel >/searches**.

   d. Click the **Post** tab.

   e. Enter your AQL query in the **Value** field for the **query_expression** parameter.

      For example,

```
Select sessionID, DATEFORMAT(starttime, 'YYYY-MM-dd HH:mm:ss')
start_time, username, sourceip, category from events
```

```
into <your_Cursor_Name> where username is not null
SESSION BY starttime username, sourceip
BEGIN category=16001
start '2016-09-14 14:20' stop '2016-09-14 14:50'
```

The <*your_cursor_name*> is any name that you want to use for the results output.

f. Click **Try it out**.

   If the query runs without errors, the response code is 201.

g. Click **/ariel >/ searches > >/{search_id} >/results**

   The **8.0 - GET - /ariel/searches/{search_id}/results page** opens.

h. In the **Value** field for the **search_id** parameter, type <*your_cursor_name*>.

i. Select **text/table** for the Mime Type.

j. Click **Try it out**.

Table 9: Query Results

sessionID	start_time	username	sourceip	category
1	2016-09-14 14:42:03	admin	9.23.121.97	16003
1	2016-09-14 14:42:09	admin	9.23.121.97	16003
2	2016-09-14 14:42:10	admin	127.0.0.1	16003
2	2016-09-14 14:42:11	admin	127.0.0.1	16003
3	2016-09-14 14:42:27	joe_blogs	9.23.121.98	16001
4	2016-09-14 14:44:11	joe_blogs	9.23.121.98	16001
5	2016-09-14 14:44:35	root	127.0.0.1	4017
5	2016-09-14 14:44:35	root	127.0.0.1	3014

Table 9: Query Results *(Continued)*

sessionID	start_time	username	sourceip	category
5	2016-09-14 14:44:55	root	127.0.0.1	4017
5	2016-09-14 14:44:55	root	127.0.0.1	3014

The categories represent specific activities in your event logs. A new session is started for every change of user name and source IP address values, for example, see **sessionid** 2 and **sessionid** 5.

Also, a new session is created for category 16001, which occurs in **sessionid** 3 and **sessionid** 4.

Example

In this example events are returned and grouped by unique session ID, where the user joe_blogs logs in and starts a process between 4 PM and 11:30 PM on November 25.

```
select sessionId,DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
start_time,username,sourceip,category from events into <cursor_name>
where username='joe_blogs'
SESSION BY starttime username,_sourceip
BEGIN category=16001
END category=16003
start '2016-11-25 16:00'
stop '2016-11-25 23:30'
```

A session is started when you get an event where the BEGIN expression is met OR the previous event ends the session.

A session is ended when you get an event where the END expression is true OR the next event starts a new session.

Event category 16001 indicates a user login or logout event on the Console, and event category 16003 indicates that a user initiated a process, such as starting a backup or generating a report. For a list of event categories, see the *Juniper Secure Analytics Administration Guide*.

Transactional Query Refinements

Refine transactional AQL queries by using the EXPLICIT expression with the BEGIN and END expressions. Also, use the TIMEOUT and TIMEWINDOW expressions to specify time intervals.

Use the EXPLICIT expression with the BEGIN and END expressions to apply more precise filtering to your transactional queries.

For example, you might use the BEGIN expression with the EXPLICIT END expression to capture several (BEGIN) unsuccessful login attempts, which are followed by an (EXPLICIT END) successful login.

Use the TIMEOUT and TIMEWINDOW expressions to apply time filters for the sessions in your transactional queries.

Expressions

Use the expressions that are described in the following to refine your transactional AQL query:

Table 10: AQL Transactional Query Expressions

Query expressions	Description
BEGIN	A session is started when you get an event where the BEGIN expression is met or the previous event ends the session.
EXPLICIT BEGIN	Starts a new session only if the EXPLICIT BEGIN expression is true.
END	A session is ended when you get an event where the END expression is true or the next event starts a new session.
EXPLICIT END	Closes the current session only if the EXPLICIT END expression is true.
TIMEOUT	Closes the session when the specified TIMEOUT period elapses from the time that the previous event occurred to the time that the current event happened.
TIMEWINDOW	Tracks the session time. Closes the session when the specified TIMEWINDOW period elapses from the time that the first event occurred to the time that the current event happened.

- Syntax --SESSION BY <*TimeExpression*> <*ExpressionList*> [EXPLICIT] BEGIN <*booleanExpression*> [EXPLICIT] END <*booleanExpression*> TIMEOUT <*IntegerLiteral millieseconds*> TIMEWINDOW <*IntegerLiteral SECONDS|MINUTES|HOURS|DAYS*>

The following examples show the examples of results that you get by using different combinations of the available query expressions:

BEGIN and END Expressions

A BEGIN expression starts a session when an event matches the BEGIN expression or the previous event ends the session.

An END expression ends a session when the END expression is true for an event or the next event starts a new session.

By using the EXPLICIT expression with the BEGIN and END expressions, you apply a more precise filter that refines the result set.

See the following examples of queries and results.

The following query example uses BEGIN and END expressions.

```
Select sessionId,
DATEFORMAT(starttime,’YYYY-MM-dd HH:mm:ss’)
start_time, username, sourceip,
category from events into TR1
where username = ’user_x’
SESSION BY starttime username, sourceip
BEGIN category=16001
END category=16003
start ’2016-12-10 16:00’ stop ’2016-12-10 23:30’
```

Event category 16001 indicates a user login or logout event on the Console, and event category 16003 indicates that a user initiated a process, such as starting a backup or generating a report.

The following table shows the results for the query that uses BEGIN and END.

Table 11: BEGIN and END Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16001

Table 11: BEGIN and END Query Results *(Continued)*

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16003
2	2016-12-10 16:00:06	user_x	10.2.2.10	16003
3	2016-12-10 16:00:10	user_x	10.2.2.10	16001
3	2016-12-10 16:00:10	user_x	10.2.2.10	16003
4	2016-12-10 16:00:11	user_x	10.2.2.10	16003
3	2016-12-10 16:00:11	user_x	10.2.2.10	16001
3	2016-12-10 16:00:11	user_x	10.2.2.10	16003

NOTE: **Sessionid** 2 consists of only one event that closes it (category 16003). A session that has one event is an exception and can happen.

EXPLICIT BEGIN and END Expressions

Events are skipped when a session is not started and an event is not an `EXPLICIT BEGIN` event.

```
Select sessionId,
DATEFORMAT(starttime,’YYYY-MM-dd HH:mm:ss’)
start_time, username, sourceip,
category from events into TR2
where username=’user_x’
SESSION BY starttime username,
sourceip EXPLICIT BEGIN category=16001
END category=16003 start ’2016-12-10 16:00’
stop ’2016-12-10 23:30’
```

The following table shows the results for the query that uses EXPLICIT BEGIN and END.

Table 12: EXPLICIT BEGIN and END Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16001
1	2016-12-10 16:00:06	user_x	10.2.2.10	16003
2	2016-12-10 16:00:07	user_x	10.2.2.10	16001
2	2016-12-10 16:00:07	user_x	10.2.2.10	16003
3	2016-12-10 16:00:11	user_x	10.2.2.10	16001
3	2016-12-10 16:00:11	user_x	10.2.2.10	16003
3	2016-12-10 16:00:11	user_x	10.2.2.10	16003
4	2016-12-10 16:00:14	user_x	10.2.2.10	16001
5	2016-12-10 16:00:15	user_x	10.2.2.10	16001
5	2016-12-10 16:00:15	user_x	10.2.2.10	16003

Only events that satisfy the `EXPLICIT BEGIN` expression are returned.

Sessionid 2 and **Sessionid** 4 in the EXPLICIT BEGIN and END don't satisfy the `EXPLICIT BEGIN` expression.

BEGIN and EXPLICIT END

Close current session only if the EXPLICIT END expression is true. There are no more checks for BEGIN events in the session when the EXPLICIT END expression is true.

Multiple BEGIN events in a single session can be associated with one `EXPLICIT END` expression. For example, you might use the `EXPLICIT END` expression for counting multiple failed login attempts that are followed by a successful login during a specific time interval (session timeout).

The following query example uses BEGIN and EXPLICIT END expressions.

```
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
start_time, username, sourceip,
category from events into TR3
where username = 'user_x'
SESSION BY starttime username, sourceip
BEGIN category=16001
EXPLICIT END category=16003
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses BEGIN and EXPLICIT END expressions.

Table 13: BEGIN and EXPLICIT END Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16001
1	2016-12-10 16:00:06	user_x	10.2.2.10	16003
2	2016-12-10 16:00:07	user_x	10.2.2.10	16003
2	2016-12-10 16:00:10	user_x	10.2.2.10	16001
2	2016-12-10 16:00:10	user_x	10.2.2.10	16003
3	2016-12-10 16:00:11	user_x	10.2.2.10	16001
3	2016-12-10 16:00:11	user_x	10.2.2.10	16003
4	2016-12-10 16:00:12	user_x	10.2.2.10	16003
4	2016-12-10 16:00:12	user_x	10.2.2.10	16001

Table 13: BEGIN and EXPLICIT END Query Results *(Continued)*

sessionID	start_Time	user name	sourceip	category
4	2016-12-10 16:00:12	user_x	10.2.2.10	16003
5	2016-12-10 16:00:13	user_x	10.2.2.10	16001
4	2016-12-10 16:00:11	user_x	10.2.2.10	16003

EXPLICIT BEGIN and EXPLICIT END

Events are ignored when a session is not started and an event is not an EXPLICIT BEGIN event.

Close current session only if the EXPLICIT END expression is true. There are no more checks for BEGIN events in the session when the EXPLICIT END expression is true.

The following query example uses both EXPLICIT BEGIN and EXPLICIT END expressions.

```
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss')
start_time, username, sourceip,
category from events into TR4
where username = 'user_x'
SESSION BY starttime username, sourceip
EXPLICIT BEGIN category=16001
EXPLICIT END category=16003
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses both EXPLICIT BEGIN and EXPLICIT END expressions.

Table 14: EXPLICIT BEGIN and EXPLICIT END Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16001

Table 14: EXPLICIT BEGIN and EXPLICIT END Query Results *(Continued)*

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06	user_x	10.2.2.10	16003
2	2016-12-10 16:00:10	user_x	10.2.2.10	16001
2	2016-12-10 16:00:10	user_x	10.2.2.10	16003
3	2016-12-10 16:00:11	user_x	10.2.2.10	16001
3	2016-12-10 16:00:12	user_x	10.2.2.10	16001
3	2016-12-10 16:00:12	user_x	10.2.2.10	16003
4	2016-12-10 16:00:13	user_x	10.2.2.10	16001
4	2016-12-10 16:00:14	user_x	10.2.2.10	16001
4	2016-12-10 16:00:14	user_x	10.2.2.10	16003
5	2016-12-10 16:00:15	user_x	10.2.2.10	16001
5	2016-12-10 16:00:15	user_x	10.2.2.10	16003

TIMEOUT

Closes the session when the specified TIMEOUT period elapses from the time that the previous event occurred to the time that the current event happened. The current event becomes part of a new session. The TIMEOUT value is specified in milliseconds.

The following query example uses the TIMEOUT expression.

```
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss.SSS')
```

```
start_time, username, sourceip,
category from events into TR5
where username='user_x'
SESSION BY starttime username, sourceip
BEGIN category=16001
EXPLICIT END category=16003
TIMEOUT 3600
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses the TIMEOUT expression.

Table 15: TIMEOUT Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06.716	user_x	10.2.2.10	16003
2	2016-12-10 16:00:10.328	user_x	10.2.2.10	16001

Sessionid 1 is ended and **sessionid** 2 is started because the TIMEOUT of 3600 is exceeded.

TIMEWINDOW

Tracks the session time. Closes the session when the specified TIMEWINDOW period elapses from the time that the first event occurred to the time that the current event happened. The current event becomes part of a new session. The TIMEWINDOW value can be specified in seconds, minutes, hours, or days.

The following query example uses the TIMEWINDOW expression.

```
Select sessionId,
DATEFORMAT(starttime,'YYYY-MM-dd HH:mm:ss.SSS')
start_time, username, sourceip,
category from events into TR6
where username='user_x'
SESSION BY starttime
_username, sourceip
BEGIN category=16001
EXPLICIT END category=16003
```

```
TIMEWINDOW 3000
start '2016-12-10 16:00'
stop '2016-12-10 23:30'
```

The following table shows the results for the query that uses the TIMEWINDOW expression.

Table 16: TIMEWINDOW Query Results

sessionID	start_Time	user name	sourceip	category
1	2016-12-10 16:00:06.415	user_x	10.2.2.10	16001
1	2016-12-10 16:00:06.433	user_x	10.2.2.10	16003
2	2016-12-10 16:00:06.716	user_x	10.2.2.10	16003
3	2016-12-10 16:00:10.328	user_x	10.2.2.10	16001
3	2016-12-10 16:00:06.328	user_x	10.2.2.10	16003

Sessionid 1 is within the TIMEWINDOW expression time but **sessionid** 2 is ended because the TIMEWINDOW of 3600 is exceeded.

RELATED DOCUMENTATION

Conditional Logic in AQL Queries | **79**

Bitwise Operators in AQL Queries | **80**

CIDR IP Addresses in AQL Queries | **86**

Conditional Logic in AQL Queries

IN THIS SECTION

Use conditional logic in AQL queries by using IF and CASE expressions.

Use conditional logic in your AQL queries to provide alternative options, depending on whether the clause condition evaluates to true or false.

CASE Statements

CASE expressions return a Boolean true or false result. When an expression is returned as true, the value of that CASE expression is returned and processing is stopped. If the Boolean result is false, then the value of the ELSE clause is returned.

In the following example, when the user name is root, the value of the CASE expression that is returned is `Admin root`. When the user name is admin, the value of the CASE expression that is returned is `Admin user`. If the CASE expressions return a Boolean false, the value of the ELSE clause is returned.

```
SELECT CASE username
WHEN 'root'
THEN 'Admin root'
WHEN 'admin'
THEN 'Admin user'
ELSE 'other' END FROM events
```

When the WHEN statement is true, the THEN statement is processed, otherwise processing finishes.

IF, THEN, ELSE Statements

Statements between THEN and ELSE are processed when the IF statement is true.

In this example, when the IF condition is true, `'ADMIN'` is returned when the user name is `'root'`, otherwise the user name is returned from the events log.

```
SELECT sourceip,
IF username = ’root’
THEN ’ADMIN’
ELSE username AS user FROM events
```

In the following example, if the log has no user name, then get it from the asset model. Otherwise, the user name is returned from the events log.

```
SELECT sourceip,
IF username IS NULL
THEN ASSETUSER(sourceip)
ELSE username AS username FROM events
GROUP BY username
LAST 2 DAYS
```

RELATED DOCUMENTATION

Bitwise Operators in AQL Queries

IN THIS SECTION

Enhance the filtering capability and performance of your AQL queries that include IP addresses by using bitwise operators. Specify filters at the IP address octet level to return specific results.

By filtering on octets in an IP address, you can refine the IP address search criteria.

For example, to search for specific device types whose last octet in a source IP address ends in 100, such as x.y.z.100, you can use the following query:

```
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into <cursor_name>
WHERE (long_ip & 0x000000ff)=0x00000064
GROUP BY long_ip
ORDER BY long_ip
```

In the example, the <*sourceip*> is returned as an integer. The integer is used by the bitwise AND operator. The hexadecimal value <*ff*> in the last octet position for the source IP address specifies a filter in the corresponding IP address octet position of 0x000000<*IP address octet hexidecimal value*>. In this case, the hexadecimal value <*64*> is substituted for the decimal value 100 in the IP address.

The result is all source IP addresses that end in 100. The results can be a list for a specific device type for a company, if the last octet of all of the IP addresses is 100.

The following examples outline scenarios to use when you search with bitwise operators.

Bitwise AND (&) Examples

Returns all IP addresses that match 10.xxx.xxx.xxx

```
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into t1
WHERE (long_ip & 0xff000000)=0x0a000000
GROUP BY long_ip
LIMIT 50
```

Returns all IP addresses that match xxx.100.xxx.xxx

```
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into t2
WHERE (long_ip & 0x00ff0000)=0x0064000
GROUP BY long_ip
ORDER BY long_ip
```

Returns all IP addresses that match xxx.xxx.220.xxx

```
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events into t3
WHERE (long_ip & 0x0000ff00)=0x000dc00
GROUP BY long_ip
ORDER BY long_ip
```

Returns all IP addresses that match xxx.xxx.xxx.1

```
SELECT LONG(sourceip)AS long_ip,
sourceip
FROM events
WHERE (long_ip & 0x000000ff)=0x0000001
GROUP BY long_ip
ORDER BY long_ip
```

Bitwise NOT (~) Examples

Use the following examples to convert each 1-bit value to a 0-bit value, or each 0-bit value to a 1-bit value, in a given binary pattern.

```
SELECT ~123456789
FROM events
LIMIT 1
```

Returns 123456790

```
SELECT ~0
FROM events
LIMIT 1
```

Returns -1

```
Returns -1
SELECT ~2147483647
FROM events
LIMIT 1
```

Returns - 2147483648

Bitwise OR Examples

Use the following examples compare two bits. If both bits have a value of "1", then the query returns a 1. If both bits have a value of "0", then the query returns a 0.

```
SELECT destinationip,
LONG(destinationip),
sourceip,
LONG(sourceip)AS source_ip,
LONG(destinationip)|source_ip
FROM events
WHERE destinationip='127.0.0.1'
LIMIT 1
```

```
SELECT destinationip,
LONG(destinationip),
sourceip,
~LONG(sourceip)AS not_source_ip,
LONG(destinationip)|not_source_ip
FROM events
WHERE destinationip='127.0.0.1'
LIMIT 1
SELECT-2147483648|2147483647
FROM events
LIMIT 1
```

Returns -1

Bitwise XOR Examples

The following examples can be used to take 2-bit patterns, or a pair of bits from each position, and convert them to either a 1 or a 0. If the bits are different, the result in that position is 1. If the bits are identical, the result in that position is 0.

```
SELECT 2147483647#2147483647
FROM events
LIMIT 1
```

Returns 0

```
SELECT 12345#6789
AS A,
(~12345 & 6789)|(12345 & ~6789)
AS B
FROM events
LIMIT 1
```

Returns 10940, 10940

ShiftLeft Examples

The number of places to shift is given as the second argument to the shift operator.

```
SELECT -1<<1
AS A
FROMS events
LIMIT 1
```

Returns -2

```
SELECT 16<<1
AS A
FROMS events
LIMIT 1
```

Returns 128

ShiftRight Examples

The operator >> uses the sign bit, which is the left-most bit, to fill the trailing positions after the shift. If the number is negative, then 1 is used as a filter and if the number is positive, then 0 is used as a filter.

```
SELECT 16>>3
AS A
FROMS events
LIMIT 1
```

Returns 2

```
SELECT -32768>>15
AS A
FROMS events
LIMIT 1
```

ShiftRightUnsigned Example

Always fills 0 regardless of the sign of the number.

```
SELECT -1>>>33
FROM events
LIMIT 1
```

Returns 2147483647

Dividing by the power of 2.

```
SELECT (20+44)>>>1 A,
(20+44)>>>2 B,
(20+44)>>>3 C,
(20+44)>>>4 D,
(20+44)>>>5 E
FROM events
LIMIT 1
```

RELATED DOCUMENTATION

CIDR IP Addresses in AQL Queries | 86

Custom Properties in AQL Queries | 88

System Performance Query Examples | 89

CIDR IP Addresses in AQL Queries

IN THIS SECTION

You can insert CIDR IP addresses in your AQL statements to query by IP address range, source IP, destination IP, or you can exclude specific CIDR IP addresses.

Examples Of CIDR IP Addresses in AQL Queries

Query by source CIDR IP address, or by destination CIDR IP address.

```
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',sourceip)
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',destinationip)
SELECT * FROM flows
WHERE INCIDR('ff02:0:0:0:0:1:ff2f:29d6',destinationv6)
```

Query for flows that have a source or destination CIDR IP address of 10.100.100.0/24

```
SELECT * FROM flows
WHERE INCIDR('10.100.100.0/24',sourceip)
OR INCIDR('10.100.100.0/24',destinationip)
```

Query for events where 192.168.222.0/24 is not the source CIDR IP address.

```
SELECT *
FROM events
WHERE NOT INCIDR('192.168.222.0/24',sourceip)
```

Query for flows where 192.168.222.0/24 is not the destination CIDR IP address.

```
SELECT *
FROM flows
WHERE NOT INCIDR('192.168.222.0/24',destinationip)
```

RELATED DOCUMENTATION

Custom Properties in AQL Queries | 88

System Performance Query Examples | 89

Events and Flows Query Examples | 91

Custom Properties in AQL Queries

IN THIS SECTION

You can call a custom property directly in your AQL statements. If the custom property contains spaces you must use double quotation marks to encapsulate the custom property.

You must enable a custom property before you can use it in an AQL statement.

If the custom property is not enabled, you will be able to run your AQL query but you will not get results.

Custom Property Example

```
SELECT Bluecoat-cs-host, sourceip, Bluecoat-cs-uri
FROM events
WHERE LOGSOURCEGROUPNAME(devicegrouplist)
ILIKE '%Proxies%'
AND Bluecoat-cs-host ILIKE '%facebook.com%'
GROUP BY sourceip
```

`Bluecoat-cs-host` is the host name from the client's URL that is requested.

`Bluecoat-cs-uri` is the original URL that is requested.

RELATED DOCUMENTATION

System Performance Query Examples | 89

Events and Flows Query Examples | 91

System Performance Query Examples

IN THIS SECTION

You can use or edit examples of system performance AQL queries to run in your network.

Use the following query examples to get information about system performance in your network or edit these examples to build your own custom queries.

Disk Utilization and CPU Usage

```
SELECT Hostname, "Metric ID", AVG(Value)
AS Avg_Value, Element
FROM events
WHERE LOGSOURCENAME(logsourceid)
ILIKE '%%health%%'
AND
"Metric ID"='SystemCPU'
OR
"Metric ID"='DiskUtilizationDevice'
GROUP BY Hostname, "Metric ID", Element
ORDER BY Hostname last 20 minutes
```

This query outputs the **Hostname**, **MetricID**, **Avg_Value**, and **Element** columns.

The **Avg_Value** column returns an average value for CPU usage and disk utilization.

Disk Utilization by Partition

```
SELECT Hostname, AVG(Value) AS Disk_Usage, Element
FROM events
where LOGSOURCENAME(logsourceid)
ILIKE '%%health%%'
and "Metric ID"='DiskUsage'
GROUP BY Hostname, Element
ORDER BY Hostname
LAST 2 HOURS
```

This query outputs the **Hostname**, **Disk_Usage**, and **Element** columns

The **Disk_Usage** column returns a value for disk usage for the directories that are listed in the **Element**column.

Disk Usage in Gigabytes (GB) Per Partition

```
SELECT element
AS Partiton_Name,
MAX(value/(1024*1024*1024))
AS 'Gigabytes_Used'
FROM events
WHERE "Metric ID"='DiskSpaceUsed'
GROUP BY element
ORDER BY Gigabytes_Used DESC
LAST 2 DAYS
```

This query outputs the **Partition_Name** and the **Gigabytes_Used** columns from the events database.

The **Gigabytes_Used** column returns a value for the gigabytes that are used by each partition that is listed in the **Gigabytes_Used** column for the last two days.

Copying Query Examples from the AQL Guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

RELATED DOCUMENTATION

Events and Flows Query Examples

IN THIS SECTION

Use or edit query examples to create events and flows queries that you can use for your AQL searches.

Use the following query examples to get information about events and flows in your network or edit these examples to build your own custom queries.

Event Rates and Flow Rates for Specific Hosts

```
SELECT AVG(Value), "Metric ID", Hostname
FROM events
WHERE LOGSOURCENAME(logsourceid)
ILIKE '%%health%%'
AND ("Metric ID"='FlowRate' OR "Metric ID"='EventRate')
GROUP BY "Metric ID", Hostname
LAST 15 minutes
```

This query outputs the **AVG_Value**, **Metric ID**, and **Hostname** columns from the events or flows database for the last 15 minutes.

The **AVG_Value** column returns a value for the average flow or event rate over the last 15 minutes for the host that is named in the **Hostname** column.

EPS Rates by Log Source

```
SELECT logsourcename(logsourceid)
AS 'MY Log Sources',
SUM(eventcount) / 2.0*60*60
AS EPS_Rates
FROM events
GROUP BY logsourceid
ORDER BY EPS_Rates DESC
LAST 2 HOURS
```

This query outputs **My Log Sources**, and **EPS_Rates** columns from events.

The **My Log Sources** column returns log source names and the **EPS_Rates** column returns the EPS rates for each log source in the last two hours.

Event Counts and Event Types Per Day

```
SELECT
DATEFORMAT( devicetime, 'dd-MM-yyyy')
AS 'Date of log source',
QIDDESCRIPTION(qid)
AS 'Description of event', COUNT(*)
FROM events
WHERE devicetime >( now() -(7*24*3600*1000) )
GROUP BY "Date of log source", qid
LAST 4 DAYS
```

This query outputs the **Date of log source**, **Description of event**, and **count** of event columns from events.

The date of the event, description of event, and count of events are returned for the last four days.

Monitoring Local to Remote Flow Traffic by Network

```
SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= ’L2R’
AND NETWORKNAME(sourceip)
ILIKE ’servers’
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the **sourceip** and **TotalBytes** columns.

The **TotalBytes** column returns the sum of the source and destination bytes that crosses from local to remote.

Monitoring Remote to Local Flow Traffic by Network

```
SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= ’R2L’
AND NETWORKNAME(sourceip)
ILIKE ’servers’
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the **sourceip** and **TotalBytes** columns.

The **TotalBytes** column returns the sum of the source and destination bytes from remote to local.

Copying Query Examples from the AQL Guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

RELATED DOCUMENTATION

Reference Data Query Examples

IN THIS SECTION

Use AQL queries to get data from reference sets, reference maps, or reference tables. You can create and populate reference data by using rules to populate reference sets, by using external threat feeds, for example, LDAP Threat Intelligence App, or by using imported data files for your reference set.

Use the following examples to help you create queries to extract data from your reference data.

Use Reference Tables to Get External Metadata for User Names That Show Up in Events

```
SELECT
REFERENCETABLE('user_data','FullName',username) AS 'Full Name',
REFERENCETABLE('user_data','Location',username) AS 'Location',
REFERENCETABLE('user_data','Manager',username) AS 'Manager',
UNIQUECOUNT(username) AS 'Userid Count',
UNIQUECOUNT(sourceip) AS 'Source IP Count',
COUNT(*) AS 'Event Count'
FROM events
WHERE qidname(qid)ILIKE '%logon%'
GROUP BY "Full Name", "Location", "Manager"
LAST 1 days
```

Use the reference table to get external data such as the full name, location, and manager name for users who logged in to the network in the last 24 hours.

Get the Global User IDs for Users in Events Who Are Flagged for Suspicious Activity

```
SELECT
REFERENCEMAP('GlobalID_Mapping',username) AS 'Global ID',
REFERENCETABLE('user_data','FullName', 'Global ID') AS 'Full Name',
UNIQUECOUNT(username),
COUNT(*) AS 'Event count'
FROM events
WHERE RULENAME(creEventlist)
ILIKE '%suspicious%'
GROUP BY "Global ID"
LAST 2 days
```

In this example, individual users have multiple accounts across the network. The organization requires a single view of a user's activity. Use reference data to map local user IDs to a global ID. The query returns the user accounts that are used by a global ID for events that are flagged as suspicious.

Use a Reference Map Lookup to Extract Global User Names for User Names That Are Returned in Events

```
SELECT
QIDNAME(qid) as 'Event name',
starttime AS Time,
sourceip AS 'Source IP',
destinationip AS 'Destination IP',
username AS 'Event Username',
REFERENCEMAP('GlobalID_Mapping', username) AS 'Global User'
FROM events
WHERE "Global User" = 'John Ariel'
LAST 1 days
```

Use the reference map to look up the global user names for user names that are returned in events. Use the `WHERE` clause to return only events for the global user John Ariel. John Ariel might have a few different user names but these user names are mapped to a global user, for example, in an external identity mapping system, you can map a global user to several user names used by the same global user.

Monitoring High Network Utilization by Users

```
SELECT
LONG(REFERENCETABLE('PeerGroupStats', 'average',
REFERENCEMAP('PeerGroup',username)))
AS PGave,
LONG(REFERENCETABLE('PeerGroupStats', 'stdev',
REFERENCEMAP('PeerGroup',username)))
AS PGstd,
SUM(sourcebytes+destinationbytes) AS UserTotal
FROM flows
WHERE flowtype = 'L2R'
GROUP BY UserTotal
HAVING UserTotal > (PGAve+ 3*PGStd)
```

Returns user names where the flow utilization is three times greater than the average user.

You need a reference set to store network utilization of peers by user name and total bytes.

Threat Ratings and Categories

```
SELECT
REFERENCETABLE('ip_threat_data','Category',destinationip)
AS 'Threat Category',
REFERENCETABLE('ip_threat_data','Rating', destinationip)
AS 'Threat Rating',
UNIQUECOUNT(sourceip)
AS 'Source IP Count',
UNIQUECOUNT(destinationip)
AS 'Destination IP Count'
FROM events
GROUP BY "Threat Category", "Threat Rating" LAST 24 HOURS
```

Returns the threat category and the threat rating.

You can look up reference table threat data and include it in your searches.

Copying Query Examples from the AQL Guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

RELATED DOCUMENTATION

- User and Network Monitoring Query Examples | 97
- Event, Flow, and Simarc Fields for AQL Queries | 101
- Events and Flows Query Examples | 91

User and Network Monitoring Query Examples

IN THIS SECTION

Use query examples to help you create your user and network monitoring query AQL queries.

Use the following examples to monitor your users and network, or you can edit the queries to suit your requirements.

Find Users Who Used the VPN to Access the Network from Three or More IP Addresses in a 24-hour Period

```
SELECT username,
UNIQUECOUNT(sourceip)
AS 'Source IP count'
FROM events
WHERE LOGSOURCENAME(logsourceid)
ILIKE '%VPN%'
AND username IS NOT NULL
GROUP BY username
HAVING "Source IP count" >= 3
ORDER BY "Source IP count"
DESC
LAST 24 HOURS
```

This query outputs the **username** and **Source IP count** columns.

The **username** column returns the names of users who used the VPN to access the network from three or more IP addresses in the last 24 hours.

Find Users Who Used the VPN from More That One Geographic Location in 24 Hours

```
SELECT username, UNIQUECOUNT(geographiclocation)
AS 'Count of locations'
FROM events
```

```
WHERE LOGSOURCENAME(logsourceid)
ILIKE '%VPN%'
AND geographiclocation <> 'other location'
AND username
IS NOT NULL
GROUP BY username
HAVING "Count of locations" > 1
ORDER BY "Count of locations"
DESC
LAST 3 DAYS
```

This query outputs the **username** and **Count of locations** columns.

The **username** column returns the names of users who used the VPN from more than one location that is not called 'other location' in the last 24 hours.

Monitoring Local to Remote Flow Traffic by Network

```
SELECT sourceip,
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
WHERE flowdirection= 'L2R'
AND NETWORKNAME(sourceip)
ILIKE 'servers'
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the **sourceip** and **TotalBytes** columns.

The **TotalBytes** column returns the sum of the source and destination bytes that crosses from local to remote.

Monitoring Remote to Local Flow Traffic by Network

```
LONG(SUM(sourcebytes+destinationbytes))
AS TotalBytes
FROM flows
```

```
WHERE flowdirection= ’R2L’
AND NETWORKNAME(sourceip)
ILIKE ’servers’
GROUP BY sourceip
ORDER BY TotalBytes
```

This query outputs the **sourceip** and **TotalBytes** columns.

The **TotalBytes** column returns the sum of the source and destination bytes from remote to local.

Application Usage by Application Name, Users, and Flows Traffic

```
SELECT sourceip
AS Source_IP,
FIRST(destinationip)
AS Destination_IP,
APPLICATIONNAME(applicationid)
AS Application,
DATEFORMAT(lastpackettime, ’dd-MM-yyyy hh:m:ss’)
AS ’Start Time’,
FIRST(sourcebytes)
AS Source_Bytes,
ASSETUSER(sourceip, NOW()) AS Src_Asset_User
FROM flows
GROUP BY Source_IP
ORDER BY Source_Bytes DESC
```

This query outputs data about your asset users, application names, and flow data. Use this query to report specific user activity or application usage, or to build a variation of this query to achieve your desired results.

Location Of Assets

```
SELECT ASSETPROPERTY(’Location’,sourceip)
AS asset_location,
COUNT(*)
FROM events
```

```
GROUP BY asset_location
LAST 1 days
```

This query outputs the **asset_location** and **count** columns.

The **asset location** column returns the location of the assets.

Copying Query Examples from the AQL Guide

If you copy and paste a query example that contains single or double quotation marks from the AQL Guide, you must retype the quotation marks to be sure that the query parses.

RELATED DOCUMENTATION

Event, Flow, and Simarc Fields for AQL Queries | **101**

Events and Flows Query Examples | **91**

Reference Data Query Examples | **94**

Event, Flow, and Simarc Fields for AQL Queries

IN THIS SECTION

Use the Ariel Query Language (AQL) to retrieve specific fields from the events, flows, and simarc tables in the Ariel database.

Supported Event Fields for AQL Queries

The event fields that you can query are listed in the following table.

Table 17: Supported Event Fields for AQL Queries

Field name	Description
adekey	Ade key
adevalue	Ade value
category	Low-level category
creEventList	Matched custom rule
credibility	Credibility
destinationMAC	Destination MAC
destinationPort	Destination port
destinationv6	IPv6 destination
destinationaddress	Destination address
destinationip	Destination IP
sourceaddress	Source address
deviceTime	Log source time
deviceType	Log source type
devicegrouplist	Device group list

Table 17: Supported Event Fields for AQL Queries ***(Continued)***

Field name	Description
domainID	Domain ID
duration	Duration
endTime	Storage time
eventCount	Event count
eventDirection	Event direction: **local-to-Local (L2L)** **local-to-remote (L2R)** **remote-to-local (R2L)** **remote-to-remote (R2R)**
geographiclocation	geographic location
sourcegeographiclocation	Source geographic location
destinationgeographiclocation	Destination geographic location
hasIdentity	Has identity
hasOffense	Associated with offense
highLevelCategory	High-level category
identityhostname	Identity host name
identityip	Identity IP address

Table 17: Supported Event Fields for AQL Queries *(Continued)*

Field name	Description
isduplicate	Is duplicate
isCREEvent	Is custom rule event
logsourceid	Log source ID
magnitude	Magnitude
pcappacket	PCAP packet
partialMatchList	Partial match list
payload	Payload
postNatDestinationIP	Destination IP after NAT
postNatDestinationPort	Destination port after NAT
postNatSourceIP	Source IP after NAT
postNatSourcePort	Source port after NAT
preNatDestinationIP	Destination IP before NAT
preNatDestinationPort	Destination port before NAT
preNatSourceIP	Source IP before NAT
preNatSourcePort	Source port before NAT

Table 17: Supported Event Fields for AQL Queries ***(Continued)***

Field name	Description
protocolid	Protocol
processorId	Event Processor ID
qid	Event name ID
relevance	Relevance
severity	Severity
sourceIP	Source IP
sourceMAC	Source MAC
sourcePort	Source port
sourcev6	IPv6 source
startTime	Start time
isunparsed	Event is unparsed
userName	User name

Supported Flow Fields for AQL Queries

The flow fields that you can query are listed in the following table.

Table 18: Supported Flow Fields for AQL Queries

Field name	Description
applicationId	Application ID
category	Category
credibility	Credibility
destinationASN	Destination ASN
destinationBytes	Destination bytes
destinationDSCP	Destination DSCP
destinationFlags	Destination flags
destinationIP	Destination IP
destinationIfIndex	Destination if index
destinationPackets	Destination packets
destinationPayload	Destination payload
destinationPort	Destination port
destinationPrecedence	Destination precedence
destinationv6	IPv6 destination
domainID	Domain ID

Table 18: Supported Flow Fields for AQL Queries *(Continued)*

Field name	Description
fullMatchList	Full match list
firstPacketTime	First packet time
flowBias	Flow bias
flowDirection	Flow direction local-to-local (L2L) local-to-remote (L2R) remote-to-local (R2L) remote-to-remote (R2R)
flowInterfaceID	Flow interface ID
flowSource	Flow Source
flowType	Flow type
geographic	Matches geographic location
hasDestinationPayload	Has destination payload
hasOffense	Has offense payload
hasSourcePayload	Has source payload
icmpCode	Icmp code
icmpType	ICMP type or code

Table 18: Supported Flow Fields for AQL Queries ***(Continued)***

Field name	Description
flowInterface	Flow interface
intervalId	Interval ID
isDuplicate	Duplicate event
lastPacketTime	Last packet time
partialMatchList	Partial match list
protocolId	Protocol ID
qid	Qid
processorID	Event processor ID
relevance	Relevance
retentionBucket	Retention bucket dummy
severity	Severity
sourceASN	Source ASN
sourceBytes	Source bytes
sourceDSCP	Source DSCP
sourceFlags	Source flags

Table 18: Supported Flow Fields for AQL Queries *(Continued)*

Field name	Description
sourceIP	Source IP
sourceIfIndex	Source if index
sourcePackets	Source packets
sourcePayload	Source payload
sourcePort	Source port
sourcePrecedence	Source precedence
sourcev6	IPv6 source
startTime	Start time
viewObjectPair	View object pair

Supported Simarc Fields for AQL Queries

The simarc fields that you can query are listed in the following table.

Table 19: Supported Simarc Fields for AQL Queries

Field name	Description
destinationPort	Destination port key creator
destinationType	Destination type key creator

Table 19: Supported Simarc Fields for AQL Queries *(Continued)*

Field name	Description
deviceId	Device key creator
direction	Direction key creator
eventCount	Event count key creator
eventFlag	Flag key creator
applicationId	Application ID key creator
flowCount	Flow count key creator
destinationBytes	Destination bytes key creator
flowSource	Flow source key creator
sourceBytes	Source bytes key creator
lastPacketTime	Time key creator
protocolId	Protocol key creator
source	Source key creator
sourceType	Source type key creator
sourceRemoteNetwork	Source remote network key creator
destinationRemoteNetwork	Destination remote network key creator

Table 19: Supported Simarc Fields for AQL Queries ***(Continued)***

Field name	Description
sourceCountry	Source geographic key creator
destinationCountry	Destination geographic key creator
destination	Destination key creator

RELATED DOCUMENTATION

Events and Flows Query Examples | **91**

Reference Data Query Examples | **94**

User and Network Monitoring Query Examples | **97**