Click to prove you're human

The QRadar Build Query and Search playbook creates an AQL query for the QRadar SIEM using the QRadarCreateAQLQuery automation queries. Complex queries take into consideration several inputs and allow including or excluding each of the values as well as performing a full or partial search. Each of the values can be searched across several fields.The playbook supports 3 separate conditions to be evaluated.For example, in the first condition, inputs will evaluate several user names that may or may not exist in several fields. The second input, can for example, evaluate for IP addresses in several fields that may or may not exist in several fields, and a third value can search for an event ID that may or may not exist in several fields. The results of all of the inputs will create an AQL query that covers all of the inputs combining all of the different conditions.Each of the inputs is validated so in case the inputs are not set correctly, the user can review and run them again.Also, populated inputs will be combined, meaning by populating the first and second values the resulting AQL query will be a combination of all of the values and not 3 separate searches. In addition, make sure to populate the inputs in order according to the indexed fields in QRadar (indexed fields should be provided before non indexed ones).Dependencies#This playbook uses the following sub-playbooks, integrations, and scripts.Sub-playbooks#QRadar Get Hunting ResultsIntegrations#Scripts#Commands#qradar-search-retrieve-eventsPlaybook Inputs#NameDescriptionDefault ValueRequiredBaseValuesToSearchThe values of the first field to search. This can be a single value or a comma-separated list of values. For example, admin1,admin2OptionalBaseFieldsToSearchThe field names of the first field to search. This can be a single value or a comma-separated list of values. For example, username,userOptionalBaseFieldStateThe state of the first field to search, meaning whether the values in the field be included or excluded. Valid options are include or exclude.includeOptionalBaseFieldMatchWhether the values of the first field should be an exact match or a partial match. Valid options are exact or partial. When choosing exact, the AQL query will use the = operator. When choosing partial, the AQL query will ILIKE and add '%%' to the values. Notice that if you choose exact you will have to specify the BaseFieldsToSearch as well.exactOptionalFirstAdditionalValuesThe values of the second field to search. This can be a single value or a comma-separated list of values. For example admin1,admin2OptionalFirstAdditionalFieldsThe field names of the second field to search. This can be a single value or a comma-separated list of values. For example admin1,admin2OptionalFirstAdditionalFieldStateThe state of the second field to search, meaning whether the values in the field should be included or excluded. Valid options are include or exclude. Notice that if you choose exact you will have to specify the FirstAdditionalFields as well.includeOptionalFirstAdditionalFieldMatchWhether the values of the second field should be an exact match or a partial match. Valid options are exact or partial.exactOptionalSecondAdditionalValuesThe values of the third field to search. This can be a single value or a comma-separated list of values. For example admin1,admin2OptionalSecondAdditionalFieldsThe field names of the third field to search. This can be a single value or a comma-separated list of values. For example username,userOptionalSecondAdditionalFieldStateThe state of the third field to search, meaning should the values in the field be included or excluded. Valid options are include or exclude.includeOptionalSecondAdditionalFieldMatchShould the values of the third field be an exact match or a partial match. Valid options are exact or partial. When choosing exact, the AQL query will use the = operator. When choosing partial, the AQL query will ILIKE and add '%%' to the values. Notice that if you choose exact you will have to specify the SecondAdditionalFields as well.exactOptionalSelectFieldsThe list of fields to select within the AQL query.The default fields areDATEFORMAT(devicetime,'dd-MM-yyyy hh:mm'),LOGSOURCENAME(logsourceid),CATEGORYNAME(category),QIDNAME(qid),sourceip,destinationip,usernameDATEFORMAT(devicetime,'dd-MM-yyyy hh:mm'),LOGSOURCENAME(logsourceid),CATEGORYNAME(category),QIDNAME(qid),sourceip,destinationip,usernameOptionalTimeFrameTime frame as used in AQL.For example:LAST 7 DAYSSTART '2019-09-25 15:51' STOP '2019-09-25 17:51'For more examples, see IBM's AQL documentation.LAST 1 HOURSOptionalUseHuntingResultsThe QRadar Get Hunting Results playbook outputs the detected hosts, users, and IP addresses detected in the QRadar search results.falseOptionalPlaybook Outputs#PathDescriptionTypeQRadar.SearchEventsThe result of the search.stringQRadar.DetectedUsersUsers detected based on the username field in your search.stringQRadar.DetectedInternalIPsInternal IP addresses detected based on fields and inputs in your search.stringQRadar.DetectedExternalIPsExternal IP addresses detected based on fields and inputs in your search.stringQRadar.DetectedInternalHostsInternal host names detected based on hosts in your assets table. Note that the data accuracy depends on how the asset mapping is configured in QRadar.stringQRadar.DetectedExternalHostsExternal host names detected based on hosts in your assets table. Note that the data accuracy depends on how the asset mapping is configured in QRadar.stringPlaybook Image# IBM QRadarVersion 7.3.2Ariel Query Language GuideIBM Note Before you use this information and the product that it supports, read the information in Notices on page 77.Product informationThis document applies to IBM QRadar Security Intelligence Platform V7.3.2 and subsequent releases unlesssuperseded by an updated version of this document. Copyright International Business Machines Corporation 2013, 2019.US Government Users Restricted Rights Use, duplication or disclosure restricted by GSA ADP Schedule Contract withIBM Corp. Contents About this guide ..... v Chapter 1. What's new for users in AQL ..... 1 Chapter 2. Ariel Query Language ..... 7 Ariel Query Language deprecated versions.....7 AQL fields changed in AQL V3..... 7 Introduction to AQL with sample queries..... 9 Ariel Query Language in the QRadar user interface.....10 Overview of Ariel Query Language..... 11 Best practices for using quotation marks in AQL queries..... 13 SELECT statement..... 16 WHERE clause..... 16 GROUP BY clause..... 18 HAVING clause..... 19 ORDER BY clause..... 20 LIKE clause..... 20 COUNT function.....21 AQL logical and comparison operators..... 22 AQL data calculation and formatting functions.....24 AQL data aggregation functions..... 30 AQL data retrieval functions..... 34 Time criteria in AQL queries..... 49 AQL date and time formats.....51 AQL subquery.....53 Grouping related events into sessions..... 54 Transactional query refinements..... 56 Conditional logic in AQL queries.....61 Bitwise operators in AQL queries..... 62 CIDR IP addresses in AQL queries..... 65 Custom properties in AQL queries.....66 System performance query examples..... 66 Events and flows query examples.....67 Reference data query examples .....68 User and network monitoring query examples.....70 Event, flow, and simarc fields for AQL queries.....72 Notices.....77 Trademarks..... 78 Terms and conditions for product documentation..... 78 IBM Online Privacy Statement..... 79 General Data Protection Regulation.....79 Index..... 81 iii About this guide The Ariel Query Language (AQL) Guide provides you with information for using the AQL advanced searching and API. Intended audience System administrators who view event or flow data that is stored in the Ariel database. Technical documentation To find IBM QRadar product documentation on the web, including all translated documentation, access the IBM Knowledge Center (. For information about how to access more technical documentation in the QRadar products library, see Accessing IBM Security Documentation Technical Note (www.ibm.com/support/docview.wss? rs=0&uid=swg21614644). Contacting customer support For information about contacting customer support, see the Support and Download Technical Note (. Statement of good security practices IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a lawful comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY. Please Note: Use of this Program may implicate various laws or regulations, including those related to privacy, data protection, employment, and electronic communications and storage. IBM QRadar may be used only for lawful purposes and in a lawful manner. Customer agrees to use this Program pursuant to, and assumes all responsibility for complying with, applicable laws, regulations and policies. Licensee represents that it will obtain or has obtained any consents, permissions, or licenses required to enable its lawful use of IBM QRadar. Copyright IBM Corp. 2013, 2019 v vi IBM QRadar : Ariel Query Language Guide Chapter 1. What's new for users in AQL IBM QRadar V7.3.1 introduces new Ariel Query Language (AQL) functions and enhancements. PARAMETERS REMOTESERVERS now includes the option to select servers in your search by specifying the ID or name of Event Processors By using the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS, you can specify an Event Processor by name in an AQL query; for example, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME('eventprocessor0', 'eventprocessor104') By using the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS; you can specify an Event Processor by ID in an AQL query, for example, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(102) By specifying an Event Processor, or servers that are connected to that Event Processor, you can run AQL queries faster and more efficiently. When you have multiple servers in your organization and you know where the data that you're looking for is saved, you can fine-tune the search to just the servers, clusters, or specific servers on Event Processors. In the following example, you search only the servers that are connected to 'eventprocessor104'. SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) FROM events GROUP BY logsourceid PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME ('eventprocessor104') You can significantly reduce the load on your servers, run the query regularly, and get your results faster when you filter your query to search fewer servers. Learn more about PARAMETERS REMOTESERVERS... Learn more about PARAMETERS REMOTESERVERS... For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query Language Guide. PARAMETERS EXCLUDESERVERS excludes servers from your AQL search Avoid having to search all AQL servers by using PARAMETERS EXCLUDESERVERS to exclude specific servers: IP address; for example, PARAMETERS EXCLUDESERVERS='177.22.123.246:32006,172.11.22.31:32006' Event Processor name; for example, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('') Event Processor ID; for example, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID() Searching only the servers that have the data that you require speeds up searches and uses less server resources. Copyright IBM Corp. 2013, 2019 1 Refine your query to exclude the servers that don't have the data that you're searching for. In the following example, you exclude servers that are connected to 'eventprocessorABC': SELECT processorid,PROCESSORNAME(processorid), LOGSOURCENAME(logsourceid) FROM events GROUP BY logsourceid PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessorABC') If you refine multiple queries by using PARAMETERS EXCLUDESERVERS, you can reduce the load on your servers and get your results faster. Learn more about PARAMETERS EXCLUDESERVERS... Learn more about PARAMETERS EXCLUDESERVERS... For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query Language Guide. Specify the Event Processor name in an AQL query by using the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS In an AQL query, you can include or exclude the servers that are connected to an Event Processor by using the ARIELSERVERS4EPNAME function to name an Event Processor in the query. For example, use the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS to include eventprocessor_ABC in the query. PARAMETERS REMOTESERVERS=ARIELSERVERS4EPNAME('eventprocessor_ABC') For example, you might want the search to exclude all servers on a named Event Processor by using the ARIELSERVERS4EPNAME function with PARAMETERS EXCLUDESERVERS. In the following example eventprocessor_XYZ is excluded in the query. PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPNAME ('eventprocessor_XYZ') Learn more about the ARIELSERVERS4EPNAME function... Learn more about the ARIELSERVERS4EPNAME function... For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query Language Guide. Specify the Event Processor ID in an AQL query by using the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS or PARAMETERS EXCLUDESERVERS In an AQL query, you can include or exclude servers connected to an Event Processor by using the ARIELSERVERS4EPID function to specify the ID of an Event Processor in the query. For example, include servers on the Event Processor that has the ID 101, PARAMETERS REMOTESERVERS=ARIELSERVERS4EPID(101) For example, exclude servers on the Event Processor that has the ID 102, PARAMETERS EXCLUDESERVERS=ARIELSERVERS4EPID(102) Learn more about the ARIELSERVERS4EPID function... Learn more about the ARIELSERVERS4EPID function... For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query Language Guide.2 IBM QRadar : Ariel Query Language Guide Filter your search by using the ARIELSERVERS4EPID function with the PARAMETERSREMOTESERVERS or PARAMETERS EXCLUDESERVERS to specify Event Processors by ID and theirAriel servers.You can use the ARIELSERVERS4EPID function with PARAMETERS REMOTESERVERS and PARAMETERSEXCLUDESERVERS to specify Ariel servers that you want to include or exclude from your search.You can also use the following query to list Ariel servers by Event Processor ID.SELECT processorid, ARIELSERVERS4EPNAME(PROCESSORNAME(processorid)) from eventsReturns Ariel servers that are associated with an Event Processor that is identified by ID.Here's an example of the output for the query, which shows the ID of the processor and the servers forthat processor: 22 localhost:32011,172.16.158.95:32006 Learn more about the ARIELSERVERS4EPID function... Learn more about the ARIELSERVERS4EPID function...For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query LanguageGuide.In an AQL query, you can specify Ariel servers that are connected to a named Event Processor byusing the ARIELSERVERS4EPNAME function.Use the ARIELSERVERS4EPNAME function with PARAMETERS REMOTESERVERS or PARAMETERSEXCLUDESERVERS to specify Ariel servers that you want to include or exclude from your search.You can also use the following query to list Ariel servers by Event Processor name.SELECT PROCESSORNAME(processorid),ARIELSERVERS4EPNAME(PROCESSORNAME(processorid)) from eventsHere's an example of the output for the query, which shows the name of the processor and the servers: eventprocessorABC localhost:32011,172.16.158.95:32006 Learn more about the ARIELSERVERS4EPNAME function... Learn more about the ARIELSERVERS4EPNAME function...For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query LanguageGuide.Use the COMPONENTID function to retrieve the ID for any named QRadar component and return datafor that component.For example, you can retrieve events for a named Event Processor. In the following example you retrieveevents from eventprocessor0: SELECT * from events where processorid = COMPONENTID('eventprocessor0') What's new for users in AQL 3 PARSETIMESTAMP function parses the text representation of date and time and converts it to UNIX epoch time Do time-based calculations easily in AQL when you convert time in text format to epoch time. Include time-based calculations in your AQL queries and use the time-based criteria that you specify to return events that helps to enhance the security of your organization by making it easier to monitor user activity. For example, you might want to find out that the difference between user logout and re-login times is less than 30 minutes. If this timing seems suspicious, you can investigate further. Learn more about the PARSETIMESTAMP function... Learn more about the PARSETIMESTAMP function... For more information, see the AQL data calculation and formatting functions topic in the IBM QRadar Ariel Query Language Guide. Retrieve information about the location and distance of IP addresses Use geographical data that is provided by MaxMind to find information about the location and distance between IP addresses in QRadar. The GEO::LOOKUP AQL function returns location data for a selected IP address. The GEO::DISTANCE AQL function returns the distance, in kilometers, of two IP addresses. Easily recognize the geographical origin of your data by organizing your data by location such as city or country instead of by IP address, and use the distance between IP addresses to evaluate the relative distance between your QRadar locations. Learn more about using the GEO::LOOKUP and the GEO::DISTANCE functions... Learn more about using the GEO::LOOKUP and the GEO::DISTANCE functions... For more information, see the AQL data retrieval functions topic in the IBM QRadar Ariel Query Language Guide. Enhanced support for the AQL subquery In QRadar V.7.2.8 and V.7.3.0, the subquery was accessible only by using API. The subquery is now available for use in searches from the Log Activity or Network Activity tabs. Learn more about the AQL subquery... Learn more about the AQL subquery... For more information, see the AQL subquery topic in the IBM QRadar Ariel Query Language Guide. Enhanced support for the SESSION BY clause In QRadar V.7.3.0 the SESSION BY clause was accessible only by using API. The SESSION BY clause is now available for use in searches in QRadar. Learn more about the AQL SESSION BY clause... Learn more about the AQL SESSION BY clause...4 IBM QRadar : Ariel Query Language Guide For more information, see the Grouping related events into sessions topic in the IBM QRadar Ariel QueryLanguage Guide. What's new for users in AQL 5 6 IBM QRadar : Ariel Query Language Guide Chapter 2. 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.Ariel Query Language deprecated versions Ariel Query Language (AQL) V1 and V2 are deprecated. The command-line script, /opt/qradar/bin/arielClient is deprecated. The following warning message is displayed both before and after the results are returned: WARNING: AQL V1 and V2 will be deprecated in the future. For information about using AQL V3, see the product documentation. During your migration to AQL V3, you can suppress the warning message by typing: /opt/qradar/bin/ arielClient | grep -v WARNING The Python client and the Advanced search option use AQL V3.AQL fields changed in AQL V3 Ariel Query Language (AQL) V2 is deprecated in QRadar V7.2.4 and later. Some Ariel database fields were changed or removed in AQL V3. If you have queries that use these fields, you must replace them. Table 1 shows the new Ariel database fields. Table 1. Fields that were replaced in AQL V3 Field name (AQL V2) Replacement function name (AQL V3) This following Ariel database fields were removed. partialorMatchList qidNumber token destinationHost destinationIPSearch destinationPortNA sourceHost sourceIPSearch Copyright IBM Corp. 2013, 2019 7 sourcePortNA destinationDscpOnly anyDestinationFlag smallDestinationPayload smallDestinationPayloadHex destinationPrecedanceOnly lastPacketDate localHost remoteHost sourceDscpOnly anySourceFlag sourcePayloadHex smallSourcePayload smallSourcePayloadHex sourcePrecedanceOnly sourceHostString destinationHostString destinationNetwork application sourceNetwork smallPayload smallPayloadHex quickSearchMatches bitsPerSecond srcBitsPerSecond dstBitsPerSecond bytesPerSecond bytesPerPacket srcBytesPerPacket dstBytesPerPacket destinationByteRatio destinationPacketRatio packetsPerSecond sourceByteRatio sourcePacketRatio totalBytes totalPackets retentionBucket properLastPacketTime properLastPacketDate8 IBM QRadar : Ariel Query Language Guide Introduction to AQL with sample queries You use Ariel Query Language (AQL) to get information about events and flows from IBM QRadar that you can't get from tables or graphs on the QRadar tabs. Use AQL queries in the Advanced Search box on the Log Activity or Network Activity tabs, or in some apps in QRadar to graph dashboard items. Use the following syntax order when you use AQL: SQL Structure [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 2. Simple AQL queries Basic AQL Commands Comments Returns all the fields from the events table that SELECT * FROM events LAST 10 MINUTES were sent in the last 10 minutes. Returns the sourceip and destinationip from SELECT sourceip,destinationip FROM events LAST 24 HOURS the events table that were sent in the last 24 hours. Returns all the fields from the events table during SELECT * FROM events START '2017 01 01 9:00:00' that time interval. STOP '2017 01 01 10:20:00' Ariel Query Language 9 Table 2. Simple AQL queries (continued) Basic AQL Commands Comments Returns all the fields in the events table during the SELECT * FROM events limit 5 LAST 24 HOURS last 24 hours, with output limited to five results. Returns all the fields in the events table sent in the SELECT * FROM events ORDER BY magnitude DESC LAST 24 HOURS last 24 hours, sorting the output from highest to lowest magnitude. Returns all the fields in the events table that have a SELECT * FROM events WHERE magnitude >= 3 LAST 24 HOURS magnitude that is less than three from the last 24 hours. Returns all the fields in the events table that have SELECT * FROM events WHERE sourceip = '192.0.2.0' the specified source IP and destination IP within AND destinationip = '198.51.100.0' START the specified time period. '2017 01 01 9:00:00' STOP '2017 01 01 10:20:00' Returns all the fields in the events table where the SELECT * FROM events WHERE INCIDR('192.0.2.0/24', source IP address is within the specified CIDR IP sourceip) range. Returns all the fields in the events table where the SELECT * FROM events WHERE username LIKE '%roul%' user name contains the example string. The percentage symbols (%) indicate that the user name can match a string of zero or more characters. Returns all the fields in the events table where the SELECT * FROM events WHERE username ILIKE '%ROUL%' 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. Returns the sourceip, category, and SELECT sourceip,category,credibility FROM events credibility fields from the events table with WHERE (severity > 3 AND category = 5018)OR specific severity levels, a specific category, and a (severity < 3 AND credibility > 8) specific credibility level. The AND clause allows for multiple strings of types of results that you want to have. Returns all the fields from the events table that SELECT * FROM events WHERE TEXT SEARCH 'firewall' have the specified text in the output. Returns all the fields in the events table where the SELECT * FROM events WHERE username ISNOT NULL username value is not null.Ariel Query Language in the QRadar 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 QRadar in the Log Activity or Network Activity tabs. To use AQL in the search fields, consider the following functions:10 IBM QRadar : Ariel Query Language Guide In the search fields on the Log Activity or Network Activity tabs, type Ctrl + Space to see the full list of AQL functions, fields, 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 3. 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, ACS (ascending), and more. Field Indicates basic information that you can query from the database. Examples include Access intent, VPC ID, and domainid. Function Various functions from string functions to call in more information. Functions work on all fields and databases. Examples of functions include DATEFORMAT, HOSTNAME, and LOWER. Figure 1. AQL in the advanced search fieldOverview of Ariel Query Language Use AQL to extract, filter, and perform actions on event and flow data that you extract from the Ariel database in IBM QRadar. 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. Ariel Query Language 11 SELECT column_name, column_name, column_name column_name, column_name FROM events column name, * options or * FROM flows column_name AS alias, column_name FUNCTION(property) AS alias, column_name WHERE condition GROUP BY column data from results Optional HAVING condition ORDER BY column_name Query results Figure 2. AQL query flow12 IBM QRadar : Ariel Query Language Guide 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 4. AQL query results sourceip destinationip Username Protocolid eventcount 192.0.2.21 198.51.100.21 Joe Ariel 233 1 192.0.2.22 198.51.100.24 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 DAYSBest practices for using quotation marks in AQL queries In an AQL query, query terms and queried columns sometimes require single or double quotation marks so that QRadar can parse the query. The following table defines when to use single or double quotation marks. Ariel Query Language 13 Table 5. Type of quotation marks to use in a query Type of quotation When to use marks 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. Examples: 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'14 IBM QRadar : Ariel Query Language Guide Table 5. Type of quotation marks to use in a query (continued)Type of quotation When to usemarksDouble 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. Examples: 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. Example: 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 Examples: SELECT sourceIP, destinationIP, sourcePort, EventCount AS 'Event Count', category, hasidentity, username, payload, UtF8(payLoad), QiD, QiDnAmE(qid) FROM events WHERE (NOT (sourcePort 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.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:16 IBM QRadar : Ariel Query Language Guide Equal sign (=) Not equal to symbol () Less than symbol () Less that 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 matchExamples of WHERE clausesThe following query example shows events that have a severity level of greater than nine and are from aspecific category. SELECT sourceIP, category, credibility FROM events WHERE severity > 9 AND category = 5013Change the order of evaluation by using parentheses. The search conditions that are enclosed inparentheses 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 sourcename. 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 wherethe QID is equal to 44250002, which is the identifier for Cron Status. SELECT * FROM events WHERE deviceType= '11' AND qid= '44250002' Ariel Query Language 17 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 aggregate functions 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 | | 10.105.2.10 | 4902509.0 | | 10.103.70.243 | 2802715.0 | | 10.103.77.143 | 3313370.0 | | 10.105.32.29 | 2467183.0 | | 10.105.96.148 | 8325356.0 | | 10.103.73.206 | 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 | | 10.105.2.10 | 2412426 | | 10.103.70.243 | 1793095 | | 10.103.77.143 | 1449148 | | 10.105.32.29 | 1097523 | | 10.105.96.148 | 4096834 | | 192.0.2.0 | 2833961 | | 10.105.2.10 | 2490083 | | 10.103.73.206 | 1629768 | | 10.103.70.243 | 1009620 | | 10.105.32.29 | 1369660 | | 10.103.77.143 | 1864222 | | 10.105.96.148 | 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 | | 10.105.2.10 | UDP.udp.ip | | 10.103.70.243 | UDP.udp.ip | | 10.103.77.143 | UDP.udp.ip | | 10.105.32.29 | TCP.tcp.ip | | 10.105.96.148 | TCP.tcp.ip | | 192.0.2.0 | TCP.tcp.ip |18 IBM QRadar : Ariel Query Language Guide | 10.105.2.10 | 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 () Less that 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 Ariel Query Language 19 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.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 6. 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%'20 IBM QRadar : Ariel Query Language Guide 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 addres

are likely to take longer considering the complexity and duration of the AQL Query. In practice, the recommended approach is to continuously poll the REST API for the status of the QRadar Ariel Search at defined intervals. You can define the interval as 30 seconds, 1 minute, 5 minutes, 10 minutes, or longer based on previous knowledge and experience. Note: Run the AQL Query or Saved Search manually at least once on the QRadar Console to approximately determine its execution time. 4. Use Search ID to retrieve result once QRadar Ariel Search is Completed Once it is determined that the QRadar Ariel Search is successfully completed, we can create a new REST API request with the Search ID to retrieve the result. The below diagram summarizes the workflow and its steps: QRadar Ariel Search REST API Endpoints Let us understand the various QRadar Ariel Search REST API endpoints and their specifications, which will allow us to complete all the steps in the above workflow. They are: 1. Find QRadar Ariel Saved Searches It was mentioned above that we can create a new QRadar Ariel Search using a Saved Search ID or an AQL Query. If you want to proceed with Saved Search ID, you will need to first query QRadar and capture the correct Saved Search ID for the desired search/AQL Query. The /ariel/saved_searches REST API endpoint can be used to retrieve a list of existing Saved Searches on QRadar. As seen in the screenshot below, a GET request to /ariel/saved_searches returns many useful fields including the name of the Saved Search, its ID, and its corresponding AQL Query. Below is a sample JSON snippet displaying the name, id, and aql fields for a Saved Search titled Top Log Sources. { "name": "Top Log Sources", "id": 2721, "aql": "SELECT logsourcename(logSourceId) AS 'Log Source', UniqueCount(\"sourceIP\") AS 'Source IP (Unique Count)', UniqueCount(\"destinationIP\") AS 'Destination IP (Unique Count)', UniqueCount(\"destinationPort\") AS 'Destination Port (Unique Count)', UniqueCount(qid) AS 'Event Name (Unique Count)', UniqueCount(category) AS 'Low Level Category (Unique Count)', UniqueCount(\"protocolId\") AS 'Protocol (Unique Count)', UniqueCount(\"userName\") AS 'Username (Unique Count)', MAX(\"magnitude\") AS 'Magnitude (Maximum)', SUM(\"eventCount\") AS 'Event Count (Sum)', COUNT(*) AS 'Count' from events GROUP BY logSourceId order by \"Event Count (Sum)\" desc last 6 hours"} It is to be noted that making a GET request to /ariel/saved_searches will return an Array of JSON objects. To make it easier, we can consider using a filter within the GET request. As seen in the screenshot below, the REST API endpoint has an optional Query parameter called filter, which can be used to limit the response to a specific Saved Search or a subset of Saved Searches. Similarly, the fields optional Query parameter can be used to specify which fields should be returned in the query response. 2. Create QRadar Ariel Search To create a new QRadar Ariel Search, make a POST request to the /ariel/searches REST API endpoint. As seen in the screenshot below, there are 2 optional Query parameters - query_expression and saved_search_id, corresponding to the AQL Query and Saved Search ID respectively. Depending on the selected approach, provide an appropriate value. The request will return a JSON response containing a unique Search ID. Below is a sample JSON snippet displaying the search_id field. { "search_id": "fdd8c0be-c88b-43fe-a3fd-6f88abfb9046"} 3. Check Status of QRadar Ariel Search Once a new QRadar Ariel Search is created, its unique Search ID can be used to check the completion status. To retrieve the status of a created search, make a GET request to /ariel/searches/{search_id} by replacing {search_id} with the actual Search ID associated with the search. As seen in the screenshot below, search_id is a required Path parameter to be sent along with the request. If we replace search_id with the Search ID from the previous snippet, the request URL would look like: /ariel/searches/fdd8c0be-c88b-43fe-a3fd-6f88abfb9046 The request will return a JSON response containing many fields pertaining to the status of the search. Below is a sample JSON snippet of the response displaying the progress, query_execution_time, and status fields. { "progress": 46, "query_execution_time": 1480, "status": "COMPLETED"} 4. Get Result of QRadar Ariel Search Once it is ascertained that the QRadar Ariel Search is completed, make a GET request to /ariel/searches/{search_id}/results to retrieve the result of the search by replacing {search_id} with the actual Search ID associated with the search. As seen in the screenshot below, search_id is a required Path parameter to be sent along with the request. It is also worth noting that the result can be retrieved in various formats. The Accepts request header indicates the format of the result. The formats are RFC compliant and can be JSON, CSV, XML, or tabular text. Below is a sample JSON snippet of the response displaying the fields specified in the AQL Query associated with the QRadar Ariel Search. "events": [{ "Log Source": "Health Metrics-2 :: localhost", "Source IP (Unique Count)": 1.0, "Destination IP (Unique Count)": 1.0, "Destination Port (Unique Count)": 1.0, "Event Name (Unique Count)": 1.0, "Low Level Category (Unique Count)": 1.0, "Protocol (Unique Count)": 1.0, "Username (Unique Count)": 0.0, "Magnitude (Maximum)": 4.0, "Event Count (Sum)": 30040.0, "Count": 30040.0 }, . . .] It is to be noted that the request will mostly return an Array of JSON objects. In the snippet above, events is an Array containing raw JSON objects, each pertaining to a specific Log Source. The fields returned in the response are solely dependent on the AQL Query associated with the QRadar Ariel Search. We can see that all the fields returned in the JSON response above are specified in the SELECT statement of the AQL Query below. SELECT logsourcename(logSourceId) AS 'Log Source', UniqueCount("sourceIP") AS 'Source IP (Unique Count)', UniqueCount("destinationIP") AS 'Destination IP (Unique Count)', UniqueCount("destinationPort") AS 'Destination Port (Unique Count)', UniqueCount(qid) AS 'Event Name (Unique Count)', UniqueCount(category) AS 'Low Level Category (Unique Count)', UniqueCount("protocolId") AS 'Protocol (Unique Count)', UniqueCount("userName") AS 'Username (Unique Count)', MAX("magnitude") AS 'Magnitude (Maximum)', SUM("eventCount") AS 'Event Count (Sum)', COUNT(*) AS 'Count'FROM eventsGROUP BY logSourceIdORDER BY "Event Count (Sum)" DESC LAST 6 HOURS Python Code We will use the programming concept of recursion to implement the QRadar Ariel Search workflow on Python. According to GeeksforGeeks: The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. We will start by importing the necessary Python packages as seen below. import requestsimport pandasimport time The next step is to define a variable called SEC_TOKEN to hold the QRadar API Token as seen below. Please refer here on how to generate a QRadar API Token. SEC_TOKEN = '4150d602-11ba-4d55-b3de-b6ebfe8b93ac' The next step is to define a variable called header to hold the Header content for the API request as seen below. We will utilize the SEC_TOKEN variable that was defined above as a value to the key SEC. header = { 'SEC':SEC_TOKEN, 'Content-Type':'application/json', 'accept':'application/json'} After the variables have been defined, we will define 2 functions as follows: 1. do_request function This function is responsible for making the actual REST API request using the requests Python module as seen below. It takes the HTTP method, request URL, and request parameters as function arguments and returns the JSON response. It is generic by design to promote re-usability and reduce the lines of code. Note: params in this function is an example of a default parameter which allows us to specify a default value for the parameter in case we do not pass an argument. By default, params will take the value of {} which is an empty dictionary unless a value is explicitly passed as an argument. def do_request(method, url, params={}): r = requests.request(method=method, url=url, params=params, headers=header, verify=False) return r.json() 2. check_status function This function is the recursive function responsible for checking the status of the QRadar Ariel Search at a defined interval of 3 seconds as seen below. The function will return the JSON response once the search is completed. The base case in the function is when the variable search_status is set to COMPLETED. In the base case, the do_request function is called to retrieve the result of the QRadar Ariel Search. When search_status is set a value other than COMPLETED, the recursive case is triggered and the same function (check_status) calls itself. First, we use time.sleep(3) to suspend the execution for 3 seconds. Then, the do_request function is called to fetch the status of the QRadar Ariel Search. The status of the search, accessed via resp_json['status'], is used as an argument in the recursive function call. The recursive function calls are repeated until the base case is satisified i.e., when search_status="COMPLETED", which then stops the recursion and retrieves the result of the search. Our goal is to ensure that the base case is triggered successfully, else the function will call itself over and over endlessly resulting in infinite recursion. def check_status(search_status, search_id): if search_status=="COMPLETED": print("Search Completed") method = "GET" url = ' s/results' % search_id return do_request(method, url) else: print("Waiting for 3 seconds...") time.sleep(3) method = "GET" url = ' s' % search_id resp_json = do_request(method, url) return check_status(resp_json['status'], search_id) According to IBM QRadar documentation: The search status value be one of: WAIT, EXECUTE, SORTING, COMPLETED, CANCELED, or ERROR. It is to be noted that we are only considering COMPLETED as the base case in our code for the sake of simplicity. A more concrete implementation of this function will likely have more base cases in the recursive function to consider CANCELED and ERROR search statuses. According to MIT: A recursive implementation may have more than one base case, or more than one recursive step. For example, the Fibonacci function has two base cases, n=0 and n=1. The next step is to utilize the above 2 defined functions to perform a new QRadar Ariel Search and display its result. Let us attempt to perform the Saved Search titled Top Log Sources. To capture the correct Saved Search ID associated with the Top Log Sources Saved Search, we will define the request URL and request parameters as seen below. url = ' params = {'filter':'name="Top Log Sources"'}type(params)# dict params is a dictionary with a single key called filter. The associated value is name="Top Log Sources". It is important to note the double quotes encapsulating the Saved Search name. The next step is to make a GET request using our previously defined function do_request as seen below. The result is stored in a variable called res_json. method = "GET"res_json = do_request(method, url, params)res_json'''[{'owner': 'admin', 'is_dashboard': True, 'description': '', 'creation_date': 1245191315681, 'uid': 'SYSTEM-13', 'database': 'EVENTS', 'is_default': False, 'is_quick_search': True, 'name': 'Top Log Sources', 'modified_date': 1622547778276, 'id': 2721, 'is_aggregate': True, 'aql': 'SELECT logsourcename(logSourceId) AS \'Log Source\', UniqueCount("sourceIP") AS \'Source IP (Unique Count)\', UniqueCount("destinationIP") AS \'Destination IP (Unique Count)\', UniqueCount("destinationPort") AS \'Destination Port (Unique Count)\', UniqueCount(qid) AS \'Event Name (Unique Count)\', UniqueCount(category) AS \'Low Level Category (Unique Count)\', UniqueCount("protocolId") AS \'Protocol (Unique Count)\', UniqueCount("userName") AS \'Username (Unique Count)\', MAX("magnitude") AS \'Magnitude (Maximum)\', SUM("eventCount") AS \'Event Count (Sum)\', COUNT(*) AS \'Count\' from events GROUP BY logSourceId order by "Event Count (Sum)" desc last 6 hours', 'is_shared': True}]'''type(res_json)# listlen(res_json)# 1 It is to be noted that res_json is of type list with a length of 1. We must remember this while attempting to parse the values. Our goal is to capture the Saved Search ID using its key - id. We will define a variable called SAVED_SEARCH_ID to hold the Saved Search ID as seen below. SAVED_SEARCH_ID = res_json[0]['id']SAVED_SEARCH_ID# 2721 Now that we have the Saved Search ID (2721), we can create the QRadar Ariel Search by defining the request URL and request parameters as seen below. method = "POST"url = ' params = {'saved_search_id':SAVED_SEARCH_ID}params# {'saved_search_id': 2721} The next step is to make a POST request using our previously defined function do_request as seen below. The result is stored in a variable called res_json. res_json = do_request(method, url, params)res_json'''{'cursor_id': '789355dd-2bb9-454a-9d05-26ba4d373d48', 'status': 'WAIT', 'compressed_data_file_count': 0, 'compressed_data_total_size': 0, 'data_file_count': 0, 'data_total_size': 0, 'index_file_count': 0, 'index_total_size': 0, 'processed_record_count': 0, 'desired_retention_time_msec': 86400000, 'progress': 0, 'progress_details': [], 'query_execution_time': 0, 'query_string': 'SELECT logsourcename(logSourceId) AS \'Log Source\', UniqueCount("sourceIP") AS \'Source IP (Unique Count)\', UniqueCount("destinationIP") AS \'Destination IP (Unique Count)\', UniqueCount("destinationPort") AS \'Destination Port (Unique Count)\', UniqueCount(qid) AS \'Event Name (Unique Count)\', UniqueCount(category) AS \'Low Level Category (Unique Count)\', UniqueCount("protocolId") AS \'Protocol (Unique Count)\', UniqueCount("userName") AS \'Username (Unique Count)\', MAX("magnitude") AS \'Magnitude (Maximum)\', SUM("eventCount") AS \'Event Count (Sum)\', COUNT(*) AS \'Count\' from events GROUP BY logSourceId order by "Event Count (Sum)" desc last 6 hours', 'record_count': 0, 'size_on_disk': 0, 'save_results': False, 'completed': False, 'subsearch_ids': [], 'snapshot': None, 'search_id': '789355dd-2bb9-454a-9d05-26ba4d373d48'}''' Our goal is to capture the Search ID using its key - search_id. We will define a variable called SEARCH_ID to hold the Search ID as seen below. SEARCH_ID = res_json['search_id']SEARCH_ID# '789355dd-2bb9-454a-9d05-26ba4d373d48' The next step is to invoke the check_status recursive function with the Search ID as seen below. The return value will be stored into a variable called resp. resp = check_status("WAIT", SEARCH_ID)'''Waiting for 3 seconds...Search Completed'''resp'''{'events': [{'Log Source': 'Health Metrics-2 :: localhost', 'Source IP (Unique Count)': 1.0, 'Destination IP (Unique Count)': 1.0, 'Destination Port (Unique Count)': 1.0, 'Event Name (Unique Count)': 1.0, 'Low Level Category (Unique Count)': 1.0, 'Protocol (Unique Count)': 1.0, 'Username (Unique Count)': 0.0, 'Magnitude (Maximum)': 5.0, 'Event Count (Sum)': 113760.0, 'Count': 113760.0}, {'Log Source': 'System Notification-2 :: qradar', 'Source IP (Unique Count)': 2.0, 'Destination IP (Unique Count)': 1.0, 'Destination Port (Unique Count)': 1.0, 'Event Name (Unique Count)': 4.0, 'Low Level Category (Unique Count)': 3.0, 'Protocol (Unique Count)': 1.0, 'Username (Unique Count)': 0.0, 'Magnitude (Maximum)': 7.0, 'Event Count (Sum)': 23292.0, 'Count': 23292.0}, {'Log Source': 'SIM Audit-2 :: qradar', 'Source IP (Unique Count)': 3.0, 'Destination IP (Unique Count)': 1.0, 'Destination Port (Unique Count)': 1.0, 'Event Name (Unique Count)': 8.0, 'Low Level Category (Unique Count)': 2.0, 'Protocol (Unique Count)': 1.0, 'Username (Unique Count)': 5.0, 'Magnitude (Maximum)': 8.0, 'Event Count (Sum)': 168.0, 'Count': 168.0}, {'Log Source': 'Anomaly Detection Engine-2 :: qradar', 'Source IP (Unique Count)': 1.0, 'Destination IP (Unique Count)': 1.0, 'Destination Port (Unique Count)': 1.0, 'Event Name (Unique Count)': 1.0, 'Low Level Category (Unique Count)': 1.0, 'Protocol (Unique Count)': 1.0, 'Username (Unique Count)': 0.0, 'Magnitude (Maximum)': 3.0, 'Event Count (Sum)': 16.0, 'Count': 16.0}]}'''type(resp)# dict The print statements defined in the check_status function help us understand if the search is still running or if it has completed. Note: You can customize the verbosity of the messages in the check_status function. While simple print statements are helpful, there are other logging mechanisms available at your disposal. We can see that resp contains the response - the result of our Top Log Sources QRadar Ariel Search in JSON format. However, the actual data we are interested in is stored under the key events. type(resp['events'])# listlen(resp['events'])# 4 At this point, it is useful to store the raw JSON data into a different data structure - namely, a Pandas DataFrame. The best way to convert our Array of JSON objects; i.e., resp['events'] which is of type list into a DataFrame is by using the pandas.json_normalize function as seen below. df = pandas.json_normalize(resp['events'])type(df)# pandas.core.frame.DataFramedf As per the above snippet, the variable df now holds our result DataFrame. The dimensions of the DataFrame can be retrieved using pandas.DataFrame.shape which returns a tuple of dimensions as seen below. df.shape# (4, 11) Now that we have our result DataFrame, we can aggregate, visualize, and export the data as desired. The below screenshot shows the final Jupyter Notebook. Conclusion In this tutorial, we learnt how to leverage the QRadar Ariel Search REST API endpoints to run Ariel searches and fetch their results programmatically using Python. To summarize: We started by understanding the relevance of searching in QRadar and how it is a basic but essential functionality. Then, we dissected the high-level steps involved in running a new QRadar Ariel Search programmatically. Here, we discussed when to use a raw AQL Query and when to use a Saved Search ID. A diagram was provided to visualize the steps in the workflow. Next, we delved into the various QRadar Ariel Search REST API endpoints available on QRadar to complete all the steps in the workflow. Here, we discussed about each endpoint including its response fields, parameters, and sample JSON response. Then, we wrote Python code using the concept of recursion to implement the steps in the workflow. We took an example Saved Search (Top Log Sources) and explained how we can capture its corresponding Saved Search ID, create a new QRadar Ariel Search, check its completion status, and retrieve the result in JSON format. We also converted the JSON response into a Pandas DataFrame to make querying and aggregation easier. Using the concepts discussed in this tutorial, you can easily write Python code to automate QRadar searching tasks (such as Threat Hunting and SOC Reporting) which previously required manual effort. You can view and download the Jupyter Notebook from this tutorial using the link below. Jupyter Notebook: QRadar Ariel Search API I hope you enjoyed reading this tutorial. Please reach out via email if you have any questions or comments.

Qradar advanced search syntax. Qradar advanced search. Qradar advanced search queries. Qradar advanced search reference set.