HCP Stellar Quick Reference 1

Stellar Language Quick Reference

Date of Publish: 2019-04-09

https://docs.hortonworks.com

Contents

Introduction to Stellar Language

Stellar is a domain-specific language (DSL) provided with HCP as a scripting environment. You can use Stellar to filter, transform, and enrich data streaming from your topologies, along with create threat triage rules. You can also use Stellar to interact with various subsystems in HCP, including HBase, Profiler, and Model as a Service (MaaS).

One of the most comon uses of Stellar is to enhance and manage data streaming from your topologies. You can create scripts to enrich and transform data. You can also use Stellar to create threat triage rules to provide a sortable score to alert messages. Because Stellar statements are stored in ZooKeeper, you do not need to restart a topology to add the new enrichment or field transformation. To implement a Stellar enrichment or transformation, you only need to update ZooKeeper.

You can also use Stellar to interact with various subsystems in HCP. For example, you can use Stellar to interact with data stores in HBase as an enrichment. You can also interact with Profiler to enable querying historical context. And you can interact with machine learning models deployed with MaaS to integrate the output of machine learning as an enrichment.

The power and flexibility of Stellar is a result of the capabilities available in the scripting environment:

- Provides commonly used simple functions (such as TO_UPPER, TRIM, IN_SUBNET)
- Provides functions to interace with the system at large (for example, data in HBase, models deployed via MaaS)
- Provides ability to create simple conditional statements
- Provides map, list, numeric and string primitives
- Can refer to variables
- Provides the ability to compose simple solutions into more complex solutions
- Provides the ability to define new functions
- Provides a REPL (read, evaluate, print loop) to allow you to test Stellar functions

Stellar Examples

Stellar examples help to illustrate how you can use to Stellar statements to transform and enrich steaming data to identify suspicious behavior.

Consider the basic example of taking a message and applying a couple of enrichments, such as converting the hostname field to lowercase. For this conversion, you must specify the transformation inside of the config file for the stellar fieldMap option. Two syntaxes are supported, specifying the transformations as a map with the key as the field and the value as the Stellar expression:

```
"fieldMap": {
       ...
      "stellar" : {
        "config" : {
          "hostname" : "To_LOWER(hostname)"
        }
      }
    }
```

Let's consider a situation where you have a message containing field ip_src_addr and you want to determine if the src address is one of a few subnet ranges. You also want to store the information in a variable called is_local:

```
"fieldMap": {
               ...
               "stellar" : {
               "config" : {
```

```
                "is_local := IN_SUBNET( ip_src_addr, '192.168.0.0/16',
 '10.10.0.0/8')"
            }
        }
    }
```

Now, let's consider a situation where you want to determine if the top level domain of a domain name, stored in a field called domain, is from a specific set of whitelisted TLDs:

```
is_government := DOMAIN_TO_TLD(domain) in [ 'mil', 'gov' ]
```

Let's assume further that the data coming in is known to be spotty with possible spaces and a dot at the end periodically due to a known upstream data ingest mistake. You can do that with three Stellar statements, the first two sanitizing the domain field and the final statement performing the whitelist check:

```
sanitized_domain := TRIM(domain)

sanitized_domain := if ENDS_WITH(sanitized_domain, '.') then
 CHOP(sanitized_domain) else sanitized_domain

is_government := DOMAIN_TO_TLD( sanitized_domain ) in [ 'mil', 'gov' ]
```

Now, let's consider a situation where you have a blacklist of known malicious domains. You can use the HCP data importer to store this data in HBase under the enrichment type malicious_domains. As data streams by, you will want to indicate whether a domain is malicious or not. Further, as before, you still have some ingestion cruft to adjust:

```
sanitized_domain := TRIM(domain)

sanitized_domain := if ENDS_WITH(sanitized_domain, '.') then
 CHOP(sanitized_domain) else sanitized_domain

in_blacklist := ENRICHMENT_EXISTS('malicious_domains', sanitized_domains,
 'enrichments', 't')
```

Stellar Functions

The Stellar language supports a number of functions.

- Referencing fields in the enriched JSON
- String literals are quoted with either ' or "
- String literals support escaping for ', ", \t, \r, \n, and backslash
  - The literal '\'foo\'' would represent 'foo'
  - The literal "\"foo\"" would represent "foo"
  - The literal 'foo \\ bar' would represent foo \ bar
- Simple boolean operations: and, not, or
- Simple arithmetic operations: *, /, +, - on real numbers or integers
- Simple comparison operations <, >, <=, >=
- Simple equality comparison operations ==, !=
- if/then/else comparisons (for example, if var1 < 10 then 'less than 10' else '10 or more')
- Simple match evaluations (for example, match{ var1 < 10 => 'warn', var1 >= 10 => 'critical', default => 'info'}
- Determining whether a field exists (via exists)
- An in operator that works like the in in Python
- The ability to have parenthesis to make order of operations explicit
- User defined functions, including Lambda expressions

Stellar Boolean Expressions

In Stellar, you can use variables in boolean expressions and variables that are not explicitly interpreted as Booleans subject to specifies rules.

Stellar optimizes Boolean searches with a short-circuit rule evaluation. With short-circuit evaluation, the second argument is executed or evaluated only if the first argument does not suffice to determine the value of the expression. In other words, for a Boolean search of A AND B, if A is false, then B is not evaluated. This is very useful for enrichments that are expensive to run due to external lookups.

In Stellar, if you use variables in Boolean expressions or variables that are not explicitly Boolean, these variables will be subject to the following rules:

- Similar to python and javascript, empty collections (for example, []) will be interpreted as false
- Similar to python and javascript, missing variables will be interpreted as false
- Variables set to null will be interpreted as false

Stellar Language Keywords

You can use Stellar language keywords to identify a syntactic form such as "not", "else", and "or."

The following keywords need to be single quote escaped in order to be used in Stellar expressions:

Table 1: Stellar Language Keywords

not	else	exists	if	then
and	or	in	NaN	match
default	==	!=	<=	>
<=	+	-	<	?
*	/	,	{	}
=>				

Using parens such as: "foo" : "<ok>" requires escaping; "foo": "'<ok>'"

Stellar Language Inclusion Checks

You can use Stellar language inclusion checks such as "in", "and", and "not" to define the content of the Stellar syntax.

- in supports string contains. For example, 'foo' in 'foobar' == true
- in supports collection contains. For example, 'foo' in ['foo', 'bar'] == true
- in supports map key contains. For example, 'foo' in { 'foo' : 5} == true
- not in is the negation of the in expression. For example, 'grok' not in 'foobar' == true`

Stellar Language Comparisons

You can use Stellar language comparisons to define the Stellar syntax.

- If either side of the comparison is null then return false.
- If both values being compared implement number then the following:
  - If either side is a double then get double value from both sides and compare using given operator.
  - Else if either side is a float then get float value from both sides and compare using given operator.

    - Else if either side is a long then get long value from both sides and compare using given operator.
    - Otherwise get the int value from both sides and compare using given operator.
- If both sides are of the same type and are comparable then use the compareTo method to compare values.
- If none of the above are met then an exception is thrown.

Stellar Language Equality Check

You can use the Stellar language equality check to define both sides of a Stellar syntax.

Below is how the == operator is expected to work:

- If either side of the expression is null then check equality using Java's `==` expression.
- Else if both sides of the expression are of Java's type Number then:
    - If either side of the expression is a double then use the double value of both sides to test equality.
    - Else if either side of the expression is a float then use the float value of both sides to test equality.
    - Else if either side of the expression is a long then use long value of both sides to test equality.
    - Otherwise use int value of both sides to test equality
- Otherwise use equals method compare the left side with the right side.

The `!=` operator is the negation of the above.

Stellar Language Lambda Expressions

Stellar provides the capability to pass lambda expressions to functions which wish to support that layer of indirection.

- (named_variables) -> stellar_expression : Lambda expression with named variables

  For instance, the lambda expression which calls TO_UPPER on a named argument x could be expressed as (x) -> TO_UPPER(x).
- var -> stellar_expression : Lambda expression with a single named variable, var
    - For instance, the lambda expression which calls TO_UPPER on a named argument x could be expressed as x -> TO_UPPER(x). Note, this is more succinct but equivalent to the example directly above.
- () -> stellar_expression : Lambda expression with no named variables.
    - If no named variables are needed, you may omit the named variable section. For instance, the lambda expression which returns a constant false would be () -> false

where

- named_variables is a comma separated list of variables to use in the Stellar expression
- stellar_expression is an arbitrary stellar expression

In the core language functions, we support basic functional programming primitives such as

- MAP - Applies a lambda expression over a list of input. For instance MAP(['foo', 'bar'], (x) -> TO_UPPER(x)) returns ['FOO', 'BAR']
- FILTER - Filters a list by a predicate in the form of a lambda expression. For instance FILTER(['foo', 'bar'], (x) -> x == 'foo') returns ['foo']
- REDUCE - Applies a function over a list of input. For instance REDUCE([1, 2, 3], (sum, x) -> sum + x, 0) returns 6

Important:

Any property that is managed by Ambari should only be modified via Ambari. Otherwise, when you restart a service, Ambari might overwrite your updates. For more information, see Update Properties.

Stellar Language Match Expression

Stellar provides the capability to write match expressions, which are similar to switch statements commonly found in C-like languages.

```
match{ logical_expression1 => evaluation expression1, logical_expression2 =>
 evaluation_expression2, default => default_expression}
```

Where:

- logical_expression is a Stellar expression that evaluates to true or false. For instance var > 0 or var > 0 AND var2 == 'foo' or IF ... THEN ... ELSE
- evaluation_expression is a Stellar Expression
- default is a required default return value, should no logical expression match
  - default is required
  - Lambda expressions are supported, but they must be no argument lambdas such as () -> STATEMENT
- Only the first clause that evaluates to true will be executed.

Stellar Language Functions

HCP supports an extensive list of core Stellar language functions.

Table 2: Stellar Core Functions

Function	Description	Input	Returns
ABS	Returns the absolute value of a number	number - The number to take the absolute value of	The absolute value of the number passed in.
APPEND_IF_MISSING	Appends the suffix to the end of the string if the string does not already end with any of the suffixes.	• string - The string to be appended. • suffix - The string suffix to append to the end of the string. • additionalsuffix - Optional - Additional string suffix that is a valid terminator.	A new string if prefix was prepended, the same string otherwise.
ASN_GET	Look up an IPV4 address and returns Autonomous System Number information about it.	• ip - The IPV4 address to lookup. • fields - Optional list of ASN fields to grab. Options are network, autonomous_system_number, autonomous_system_organization.	If a single field is requested, a string of the field. If mutiple fields, a map of string of the the fields, and null otherwise.
BIN	Computes the bin that the value is in given a set of bounds	• value - the value to bin • bounds -A list of value bounds (excluding min and max) in sorted order	Which bin N the value falls in such that bound(N-1) <value <= bound(N). No min and max bounds are provided, so values small than the 0'th bound go in the 0'th bin, and values great than the last bound go in the M'th bin.
BLOOM_ADD	Adds an element to the bloom filter passed in	• bloom - The bloom filter • value* - The values to add	Bloom Filter
BLOOM_EXISTS	If the bloom filter contains the value	• bloom - The bloom filter • value - The value to check	True if the filter might contain the value and false otherwise

Function	Description	Input	Returns
BLOOM_INIT	Returns an empty bloom filter	• expectedInsertions - The expected insertions • falsePositiveRate - The false positive rate you are willing to tolerate	Bloom Filter
BLOOM_MERGE	Returns a merged bloom filter	• bloomfilters - A list of bloom filters to merge	Bloom Filter or null if the list is empty
CEILING	Returns the ceiling of a number.	• number - The number to take the ceiling of	The ceiling of the number passed in.
CHOP	Remove the last character from a string.	• string- The string to chop last character from, may be null.	String without last character, null if null string input.
CHOMP	Removes one newline from end of a string if its there, otherwise leaves it alone. A newline is "/n", "/r", "/r/n".	• The string to chomp a newline from, may be null.	String without newline, null if null string input.
COS	Returns the cosine of a number.	• number - The number to take the cosine of.	The cosine of the number passed in.
COUNT_MATCHES	Counts how many times the substring appears in the larger string.	• string - The CharSequence to check, may be null. • substring/character - The number of non-overlapping occurrences, 0 if either CharSequence is null.	
DATE_FORMAT	Takes an epoch timestamp and converts it to a date format.	• format - DateTime format as a String • timestampField - Optional epoch time in Long format. Defaults to now. • timezone - Optional timezone in String format	Returns: Formatted date.
DAY_OF_MONTH	The numbered day within the month. The first day within the month has a value if 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered day within the month
DAY_OF_WEEK	The numbered day within the week. The first day of the week, Sunday, has a value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered day within the week.
DAY_OF_THE_YEAR	The day number within the year. The first day of the year has value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The day number within the year
DECODE	Decodes the passed string with the provided encoding, which must be one of the encodings returned from GET_SUPPORTED_ENCODINGS	• string - The string to decode • encoding - the encoding to use, must be one of the encodings returned from GET_SUPPORTED_ENCODINGS • verify - (optional), true or false to determine if string should be verified as being encoded with the passed encoding	• The decoded string on success • The original string the string cannot be decoded null on usage error

Function	Description	Input	Returns
DOMAIN_REMOVE_SUBDOMAINS	Remove subdomains from a domain	• domain - Fully qualified domain name	The domain without the subdomains. (For example, DOMAIN_REMOVE_SUBDOMAINS ('mail.yahoo.com') yields 'yahoo.com')
DOMAIN_REMOVE_TLD	Removes the top level domain (TLD) suffix from a domain	• domain - Fully qualified domain name	The domain without the TLD. (For example, DOMAIN_REMOVE_TLD('mail.yahoo.co.uk') yields 'mail.yahoo')
DOMAIN_TO_TLD	Extracts the top level domain from a domain	• domain - Fully qualified domain name	The domain of the TLD. (For example, DOMAIN_TO_TLD('mail.yahoo.com.uk') 'yields 'co.uk')
ENCODE	Encodes the passed string with the provided encoding, which must be one of the encodings returned from GET_SUPPORTED_ENCODINGS	• string - the string to encode • encoding - the encoding to use, must be one of the encodings returned from GET_SUPPORTED_ENCODINGS	• The encoded string on success • null on error
ENDS_WITH	Determines whether a string ends with a suffix	• string - The string to test • suffix - The proposed suffix	True if the string ends with the specified suffix and false if otherwise
ENRICHMENT_EXISTS	Interrogates the HBase table holding the simple HBase enrichment data and returns whether the enrichment type and indicator are in the table	• enrichment_type - The enrichment type • indicator - The string indicator to look up • nosql_table - The NoSQL table to use • column_family - The column family to use	True if the enrichment indicator exists and false otherwise
ENRICHMENT_GET	Interrogates the HBase table holding the simple HBase enrichment data and retrieves the tabular value associated with the enrichment type and indicator	• enrichment_type - The enrichment type • indicator - The string indicator to look up • nosql_table - The NoSQL table to use • column_family - The column family to use	A map associated with the indicator and enrichment type. Empty otherwise.
EXP	Returns Euler's number raised to the power of the argument.	• number - the power to which e is raised	Euler's number raised to the power of the argument.
FILL_LEFT	Fills or pads a given string with a given character, to a given length on the left.	• input - string • fill - the fill character • len - the required length	The filled string
FILL_RIGHT	Fills or pads a given string with a given character, to a given length on the right.	• input - string • fill - the fill character • len - the required length	Last element of the list
FILTER	Applies a filter in the form of a lambda expression to a list. For example, `FILTER( [ 'foo', 'bar' ] , (x) -> x == 'foo')` would yield `['foo'.	• list - List of arguments. • predicate - The lambda expression to apply. This expression is assumed to take one argument and return a boolean.	The input list filtered by the predicate.
FLOOR	Returns the floor of a number.	• number - The number to take the floor of	• The floor of the number passed in.

Function	Description	Input	Returns
FORMAT	Returns a formatted string using the specified format string and arguments. Uses Java's string formatting conventions	• format - string • arguments - object(s)	A formatted string
FUZZY_LANGS	Returns a list of IETF BCP 47 available to the system, such as en, fr, de.		A list of IEF BGP 47 language tag strings
FUZZY_SCORE	Returns the Fuzzy Score which indicates the similarity score between two strings. One point is given for every matched character. Subsequent matches yield two bonus points. A higher score indicates a higher similarity.	• string - The full term that should be matched against. • string - The query that will be matched against a term. • string - The IETF BCP 47 language code to use.	An Integer representing the score.
GEO_GET	Look up an IPV4 address and returns geographic information about it.	• ip - The IPV4 address to look up • fields – Optional list of GeoIP fields to grab. Options are locID, country, city postalCode, dmaCode, latitude, longitude, location_point • len - the required length	If a Single field is requested, a string of the field. If multiple fields are requested, a map of string of fields. Otherwise null.
GEOHASH_CENTROID	Compute the centroid (geographic midpoint or center of gravity) of a set of geohashes	• hashes - A collection of geohashes or a map associating geohashes to numeric weights • character_precision? - The number of characters to use in the hash. Default is 12.	The geohash of the centroid.
GEO_DIST	Compute the distance between geohashes.	• hash1 - The first point as a geohash • hash2 - The second point as a geohash • strategy? - The great circle distance strategy to use. One of HAVERSINE, LAW_OF_COSINES, or VICENTY. Haversine is default.	The distance in kilometers between the hashes.
GEOHASH_FROM_LATLONG	Compute geohash given a lat/long.	• latitude - The latitude • longitude - The longitude • character_precision? - The number of characters to use in the hash. Default is 12.	A geohash of the lat/long.
GEOHASH_FROM_LOC	Compute geohash given a geo enrichment location.	• map - the latitude and logitude in a map (the output of GEO_GET) • longitude - The longitude • character_precision? - The number of characters to use in the hash. Default is 12	A geohash of the location.

Function	Description	Input	Returns
GEOHASH_MAX_DIST	Compute the maximum distance among a list of geohashes	• hashes - A set of geohashes • strategy? - The great circle distance strategy to use. One of HAVERSINE, LAW_OF_COSINES, or VICENTY. Haversine is default.	The maximum distance in kilometers between any two locations
GEOHASH_TO_LATLONG	Compute geohash given a lat/long.	• • hash - The geohash	A map containing the latitude and longitude of the hash (keys "latitude" and "longitude")
GET	Returns the i'th element of the list	• input - List • i - The index (0-based)	First element of the list
GET_FIRST	Returns the first element of the list	• input - List	First element of this list
GET_HASHES_AVAILABLE	Will return all available hashing algorithms to 'HASH'.		A list containing all supported hashing algorithms.
GET_LAST	Returns the last element of the list	• input - List	Last element of the list
GET_SUPPORTED_ENCODINGS	Returns a list of the encodings that are currently supported.		A List of String
HASH	Hashes a given value using the given hashing algorithm and returns a hex encoded string.	• toHash - value to hash. • hashType - A valid string representation of a hashing algorithm. See 'GET_HASHES_AVAILABLE'.	A hex encoded string of a hashed value using the given algorithm. If 'hashType' is null then '00', padded to the necessary length, will be returned. If 'toHash' is not able to be hashed or 'hashType' is null then null is returned.
HLLP_ADD	Add value to the HyperLogLogPlus estimator set.	• hyperLogLogPlus - the hllp estimator to add a value to • value+ - value to add to the set. Takes a single item or a list.	The HyperLogLogPlus set with a new value added
HLLP_CARDINALITY	HyperLogLogPlus-estimated cardinality for this set.	• input - hyperLogLogPlus - the hllp set	Long value representing the cardinality for this set
HLLP_INIT	Initializes the set	• p (required) - The precision value for the sparse set. • sp - The precision value for the sparse set. If sp Is not specified the sparse set will be disabled.	A new HyperLogLogPlus set
HLLP_MERGE	Merge hllp sets together	• hllp1 - First hllp set • hllp2 - Second hllp set • hllpn - Additional sets to merge	A new merged HyperLogLogPlus estimator set
IN_SUBNET	Returns true if an IP is within a subnet range	• ip - The IP address in string form • cidr+ - One or more IP ranges specified in CIDR notation (for example, 192.168.0.0/24)	True if the IP address is within at least one of the network ranges and false if otherwise
IS_DATE	Determines if the date contained in the string conforms to the specified format	• date - The date in string form • format - The format of the date	True if the date is in the specified format and false if otherwise

Function	Description	Input	Returns
IS_DOMAIN	Tests if a string is a valid domain. Domain names are evaluated according to the standards RFC1034 Section 3, and RFC1123 section 2.1.	• address - The string to test	True if the string is a valid domain and false if otherwise
IS_EMAIL	Tests if a string is a valid email address	• address -The string to test	True if the string is a valid email address and false if otherwise
IS_EMPTY	Returns true if string or collection is empty or null and false if otherwise	• input - Object of string or collection type (for example, list)	True if the string or collection is empty or null and false if otherwise
IS_ENCODING	Returns true if the passed string is encoded in one of the supported encodings and false if otherwise.	• string - The string to test • encoding - The name of the encoding as string. See GET_SUPPORTED_ENCODINGS	True if the passed string is encoded in one of the supported encodings and false if otherwise.
IS_INTEGER	Determines whether or not an object is an integer	• x - The object to test	True if the object can be converted to an integer and false if otherwise
IS_IP	Determine if a string is an IP or not	• ip - An object which we wish to test is an IP • type (optional) - Object of string or collection type (for example, list) one of IPv4 or IPv6. The default is IPv4.	True if the string is an IP and false if otherwise
IS_NAN	Evaluates if the passed number is NaN. The number is evaluated as a double.	• number - number to evaluate"	True if the number is NaN, false if it is
IS_URL	Tests if a string is a valid URL	• url - The string to test	True if the string is a valid URL and false otherwise
JOIN	Joins the non-null items in the iterable as strings with the specified delimiter. Null items are dropped.	• iterable - Java iterable (for example, List, LinkedHashSet, etc.) of items treated as strings • delim - String delimiter	String
KAFKA_GET	Retrieves messages from a Kafka topic. Subsequent calls will continue retrieving messages sequentially from the original offset.	• topic - the name of the Kafka topic. • count - The number of Kafka messages to retrieve. • config - Optional map of key/values that override any global properties.	List of String
KAFKA_PROPS	Retrieves the Kafka properties that are used by other KAFKA_* functions like KAFKA_GET and KAFKA_PUT. The Kafka properties are compiled from a set of default properties, the global properties, and any overrides.	• config - An optional map of key/values that override any global properties	Map of key/value pairs
KAFKA_PUT	Sends messages to a Kafka topic.	• topic - The name of the Kafka topic. • messages -A list of messages to write. • config - Optional map of key/values that override any global properties.	N/A

Function	Description	Input	Returns
KAFKA_TAIL	Retrieves messages from a Kafka topic always starting with the most recent message first.	• topic - The name of the Kafka topic. • count - The number of Kafka messages to retrieve. • config - Optional map of key/values that override any global properties.	List of String
LENGTH	Returns the length of a string or size of a collection. Returns 0 for empty or null strings.	• input - Object of string or collection type (for example, list). • element - Element to add to list.	Integer
LIST_ADD	Adds an element to a list.	• list - List to add element to. •	Resulting list with the item added at the end.
LN	Returns the natural log of a number.	• number - The number to take the natural log of •	The natural log of the number passed in.
LOG2	Returns the log (base 2) of a number.	• number - The number to take the log (base 10) of •	The log (base 2) of the number passed in.
LOG10	Returns the log (base 10) of a number.	• number - The number to take the log (base 2) of •	The log (base 10) of the number passed in.
MAAS_GET_ENDPOINT	Inspects ZooKeeper and returns a map containing the name, version, and url for the model referred to by the input parameters	• model_name - The name of the model • model_version - The optional version of the model. If the model version is not specified, the most current version is used.	A map containing the name, version, url for the REST endpoint (fields named name, version, and url). Note that the output of this function is suitable for input into the first argument of MAAS_MODEL_APPLY.
MAAS_MODEL_APPLY	Returns the output of a model deployed via Model as a Service. Note: Results are cached locally 10 minutes.	• endpoint - A map containing name, version, and url for the REST endpoint • function - The optional endpoint path; default is 'apply' • model_args - A dictionary of arguments for the model (these become request params)	The output of the model deployed as a REST endpoint in map form. Assumes REST endpoint returns a JSON map.
MAP	Applies lambda expression to a list of arguments. e.g. `MAP( [ 'foo', 'bar' ] , (x) -> TO_UPPER(x) )` would yield `[ 'FOO', 'BAR' ]`.	• string -List of arguments. • prefix - The string prefix to prepend to the start of the string. • additionalprefix - Optional - Additional string prefix that is valid.	A new String if prefix was prepended, the same string otherwise.
MAP_EXISTS	Checks for existence of a key in a map	• key - The key to check for existence • map - The may to check for existence of the key	True if the key is found in the map and false if otherwise
MAP_MERGE	Merges a list of maps	• maps - A collection of maps to merge. Last entry wins the ovelapping keys.	A map. Null if the list of maps is empty.

Function	Description	Input	Returns
MAP_PUT	Adds a key/value pair to a map	• key - The key • value - The value • map - The map on which to perform the put	The original map modified with the key/value. If the map argument is null, a new map will be created and returned that contains the provided key and value. Note: If the 'map' argument is null, only the returned map will be non-null and contain the key/value.
MAP_GET	Gets the value associated with a key from a map	• key - The key • map - The map • default - Optionally the default value to return if the key is not in the map.	The object associated with the key in the map. If no value is associated with the key and default is specified, then default is returned. If no value is associated with the key or default, then null is returned.
MAX	Returns the maximum value of a list of input values.	• "list - List of arguments. The list may only contain objects that are mutually comparable / ordinal (implement java.lang.Comparable interface). Multi type numeric comparisons are supported: MAX([10,15L,15.3]) would return 15.3, but MAX(['23',25]) will fail and return null as strings and numbers can't be compared.	The maximum value of the list, or null if the list is empty or the input values were not comparable.
MIN	Returns the minimum value of a list of input values.	• "list - List of arguments. The list may only contain objects that are mutually comparable / ordinal (implement java.lang.Comparable interface). Multi type numeric comparisons are supported: MIN([10,15L,15.3]) would return 10, but MIN(['23',25]) will fail and return null as strings and numbers can't be compared.	The minimum value of the list, or null if the list is empty or the input values were not comparable.
MONTH	The number representing the month. The first month, January, has a value of 0.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The current month (0-based).
MULTISET_ADD	Adds to a multiset, which is a map associating objects to their instance counts.	• set - The multiset to add to • o - object to add to multiset	A multiset
MULTISET_INIT	Creates an empty multiset, which is a map associating objects to their instance counts.	• input? - An initialization of the multiset	A multiset
MULTISET_MERGE	Merges a list of multisets, which is a map associating objects to their instance counts.	• sets - A collection of multisets to merge	A multiset
MULTISET_REMOVE	Removes from a multiset, which is a map associating objects to their instance counts.	• set - The multiset to add to • o - object to add to multiset	A multiset

Function	Description	Input	Returns
MULTISET_TO_SET	Create a set out of a multiset, which is a map associating objects to their instance counts.	• multiset - The multiset to convert	The set of objects in the multiset ignoring multiplicity
OBJECT_GET	Retrieve and deserialize a serialized object from HDFS. The cache can be specified via two properties in the global config: "object.cache.size" (default 1000), "object.cache.expiration.minutes" (default 1440). Note, if these are changed in global config, topology restart is required.	• path - The path in HDFS to the serialized object	The deserialized object.
PREPEND_IF_MISSING	Prepends the prefix to the start of the string if the string does not already start with any of the prefixes.	• string - The string to be prepended. • prefix - The string prefix to prepend to the start of the string. • additionalprefix - Optional - Additional string prefix that is valid.	A new string if prefix was prepended, the same string otherwise.
PREPEND_IF_MISSING	Prepends the prefix to the start of the string if the string does not already start with any of the prefixes.	• string - The string to be prepended. • prefix - The string prefix to prepend to the start of the string. • additionalprefix - Optional - Additional string prefix that is valid.	A new String if prefix was prepended, the same string otherwise.
PROFILE_FIXED	The profile periods associated with a fixed lookback starting from now	• durationAgo - How long ago should values be retrieved from? • units - The units of 'durationAgo' • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty Map, meaning no overrides.	The selected profile measurement timestamps. These are ProfilePeriod objects.

Function	Description	Input	Returns
PROFILE_GET	Retrieves a series of values from a stored profile	• profile - The name of the profile • entity - The name of the entity • periods - The list of profile periods to grab. These are ProfilePeriod objects. • groups_list -Optional - Must correspond to the 'groupBy' list used in profile creation - List (in square brackets) of groupBy values used to filter the profile. Default is the empty list, meaning groupBy was not used when creating the profile. • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty Map, meaning no overrides.	The profile measurements
PROFILE_VERBOSE	Retrieves a series of measurements from a stored profile. Returns a map containing the profile name, entity, period id, period start, period end for each profile measurement. Provides a more verbose view of each measurement than PROFILE_GET.	• profile - The name of the profile • entity - The name of the entity • periods - The list of profile periods to fetch. Use PROFILE_WINDOW or PROFILE_FIXED. • groups - Optional - The groups to retrieve. Must correspond to the 'groupBy' used during profile creation. Defaults to an empty list, meaning no groups.	The selected profile measurements.
PROFILE_WINDOW	The profiler periods associated with a window selector statement from an optional reference timestamp.	• WindowSelector - The statement specifying the window to select. • now - Optional - The timestamp to use for now. • config_overrides - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty Map, meaning no overrides.	Returns: The selected profile measurement periods. These are ProfilePeriod objects.
PROTOCOL_TO_NAME	Converts the IANA protocol number to the protocol name	• IANA number	The protocol name associated with the IANA number
REDUCE	Reduces a list by a binary lambda expression. That is, the expression takes two arguments. Usage example: `REDUCE( [ 1, 2, 3 ] , (x, y) -> x + y, 0)` would sum the input list, yielding `6`.	• list - List of arguments. • binary operation - The lambda expression function to apply to reduce the list. It is assumed that this takes two arguments, the first being the running total and the second being an item from the list.initial. • initial_value - The initial value to use.	The reduction of the list.

Function	Description	Input	Returns
REGEXP_MATCH	Determines whether a regex matches a stringASN	• input -String to split • delim - String delimiter	List of strings
REGEXP_REPLACE	Replace all occurences of the regex pattern within the string by value	• string - The input string • pattern - The proposed regex pattern • value - The value to replace the regex pattern	The modified input string with replaced values.
REGEX_GROUP_VAL	Returns the value of a group in a regex against a string	• string - The string to test • pattern -The proposed regex pattern • group - The integer that selects what group to select, starting at 1	The value of the group, or null if not matched or no group at index.
REST_GET	Performs a REST GET request and parses the JSON results into a map.	• url - URL to the REST service • rest_config - Optional - Map (in curly braces) of name:value pairs, each overriding the global config parameter of the same name. Default is the empty map, meaning no overrides.	JSON results as a map.
ROUND	Rounds a number to the nearest integer. This is half-up rounding.	• number - The number to round	The nearest integer (based on half-up rounding).
SET_ADD	Adds to a set	• set - The set to add to • o - object to add to set	A Set
SET_INIT	Creates an new set	• Input? - An initialization of the set	A Set
SET_MERGE	Merges a list of sets	• sets - A collection of sets to merge	A Set
SET_REMOVE	Removes from a set	• set - The set to add to • o - object to add to set	A Set
SHELL_EDIT	Open an editor (optional initialized with text) and return whatever is saved from the editor. The editor to use is pulled from EDITOR or VISUAL environment variable.	• string - (Optional) A string whose content is used to initialize the editor.	The content that the editor saved after editor exit.
SHELL_GET_EXPRESSION	Get a stellar expression from a variable.	• variable - Variable name	The stellar expression associated with the variable.
SHELL_LIST_VARS	Return the variables in a tabular form.	• wrap - Length of string to wrap the columns.	A tabular representation of the variables.
SHELL_MAP2TABLE	Take a map and return a table.	• map - Map	The map in table form.
SHELL_VARS2MAP	Take a set of variables and return a map.	• variables* - Variable name to use to create map.	A map associating the variable name with the stellar expression.
SIN	Returns the sine of a number.	• number - The number to take the sine of	The sine of the number passed in.
SPLIT	Splits the string by the delimiter	• inputs - String to split • delim - String delimiter	List of strings

Function	Description	Input	Returns
SQRT	Returns the square root of a number.	• number - The number to take the square root of	The square root of the number passed in.
STARTS_WITH	Determines whether a string starts with a prefix	• string -the string to test • prefix - The proposed prefix	True if the string starts with the specified prefix and false if otherwise
STATS_ADD	Add one or more input values to those that are used to calculate the summary statistics	• stats - The Stellar statistics object. If null, then a new one is initialized • value+ - One or more numbers to add	A Stellar statistics object
STATS_BIN	Computes the bin that the value is in based on the statistical distribution.	• stats - The Stellar statistics object • value - The value to bin • bound? - A list of percentile bin bounds (excluding min and max) or a string representing a known and common set of bins. For convenience, we have provided QUARTILE, QUINTILE, and DECILE which you can pass in as a string arg. If this argument is omitted, then we assume a Quartile bin split.	Which bin N the value falls in such that bound(N-1) < value <= bound(N). No min and max bounds are provided, so values smaller than the 0'th bound go in the 0'th bin, and values greater than the last bound go in the M'th bin.
STATS_COUNT	Calculates the count of the values accumulated (or in the window if a window is used)	• stats - The Stellar statistics object	The count of the values in the window or NaN if the statistics object is null
STATS_GEOMETRIC_MEAN	Calculates the geometric mean of the accumulated values (or in the window if a window is used). See http://commons.apache.org.proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The geometric mean of the values in the window or NaN if the statistics object is null
STATS_INIT	Initializes a statistics object	• window_size - The number of input data values to maintain in a rolling window in memory. If window_size is equal to 0, then no rolling window is maintained. Using no rolling window is less memory intensive, but cannot calculate certain statistics like percentiles and kurtosis.	A Stellar statistics object
STATS_KURTOSIS	Calculates the kurtosis of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The kurtosis of the values in the window or NaN if the statistics object is null
STATS_MAX	Calculates the maximum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The maximum of the accumulated values in the window or NaN if the statistics object is null
STATS_MEAN	Calculates the mean of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The mean of the values in the window or NaN if the statistics objects is null

Function	Description	Input	Returns
STATS_MERGE	Merges statistics objects	• statistics - A list of statistics providers	A Stellar statistics object
STATS_MIN	Calculates the minimum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The minimum of the accumulated values in the window of NaN if the statistics object is null
STATS_PERCENTILE	Computes the p'th percentile of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object • p - A double where 0<=1 representing the percentile	The p'th percentile of the data or NaN if the statistics object is null
STATS_POPULATION_VARIANCE	Calculates the population variance of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The population variance of the values in the window of NaN if the statistics object is null
STATS_QUADRATIC_MEAN	Calculates the quadratic mean of the accumulated values (or in the window if the window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The quadratic mean of the values in the window or NaN if the statistics object is null
STATS_SD	Calculates the standard deviation of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The standard deviation of the values in the window or NaN if the statistics object is null
STATS_SKEWNESS	Calculates the skewness of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The skewness of the values in the window of NaN if the statistics object is null
STATS_SUM	Calculates the sum of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The sum of the values in the window or NaN if the statistics object is null
STATS_SUM_LOGS	Calculates the sum of the (natural) log of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The sum of the (natural) log of the values in the in window or NaN if the statistics object is null
STATS_SUM_SQUARES	Calculates the sum of the squares of the accumulated values (or in the window if a window is used)	• stats - The Stellar statistics object	The sum of the squares of the values in the window or NaN if the statistics object is null
STATS_VARIANCE	Calculates the variance of the accumulated values (or in the window if a window is used). See http://commons.apache.org/proper/commons-math/userguide/stat.html#a1.2_Descriptive_statistics	• stats - The Stellar statistics object	The variance of the values in the window or NaN if the statistics object is null
STRING_ENTROPY	Computes the base-2 shannon entropy of a string.	input - string	The base-2 shannon entropy of the string (https://en.wikipedia.org/wiki/Entropy_(information_theory)#Definition). The unit of this is bits.

Function	Description	Input	Returns
SUBSTRING	Returns the substring of a string	• input - The string to take the substring of • start - The starting position (0 -based and inclusive) • end? - The ending position (0 -based and exclusive)	The substring of the input
SYSTEM_ENV_GET	Returns the value associated with an environment variable	• env_var - Environment variable name to get the value for	String
SYSTEM_PROPERTY_GET	Returns the value associated with a Java system property	• key - Property to get the value for	String
TAN	Returns the tangent of a number.	• number - The number to take the tangent of	The tangent of the number passed in.
TLSH_DIST	Will return the hamming distance between two TLSH hashes (note: must be computed with the same params). For more information, see https://github.com/trendmicro/tlsh and Jonathan Oliver, Chun Cheng, and Yanggui Chen, TLSH - A Locality Sensitive Hash. 4th Cybercrime and Trustworthy Computing Workshop, Sydney, November 2013. For a discussion of tradeoffs, see Table II on page 5 of https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf	• hash1 - The first TLSH hash • hash2 - The first TLSH hash • includeLength? - Include the length in the distance calculation or not? Returns: An integer representing the distance between hash1 and hash2. The distance is roughly hamming distance, so 0 is very similar.	
TO_DOUBLE	Transforms the first argument to a double precision number	• Input - Object of string or numeric type	Double version of the first argument
TO_EPOCH_TIMESTAMP	Returns the epoch timestamp of the dateTime in the specified format. If the format does not have a timestamp and you wish to assume a given timestamp, you may specify the timezone optionally.	• dateTime - DateTime in string format • format - DateTime format as string • timezone - Optional timezone in a string format	Epoch timestamp
TO_FLOAT	Transforms the first argument to an integer	• Input - Object of string or numeric type	Float version of the first argument
TO_INTEGER	Transforms the first argument to an integer	• Input - Object of string or numeric type	Integer version of the first argument
TO_JSON_LIST	Accepts JSON string as an input and returns a List object parsed by Jackson. You need to be aware of content of JSON string that is to be parsed. For e.g. GET_FIRST(TO_JSON_LIST('["foo", 2]') would yield foo	• string - The JSON string to be parsed	A parsed List object
TO_JSON_MAP	Accepts JSON string as an input and returns a Map object parsed by Jackson. You need to be aware of content of JSON string that is to be parsed. For e.g. MAP_GET('bar', TO_JSON_MAP('{ "foo" : 1, "bar" : 2}')would yield 2	• string - The JSON string to be parsed	A parsed Map object

Function	Description	Input	Returns
TO_JSON_OBJECT	Accepts JSON string as an input and returns a JSON Object parsed by Jackson. You need to be aware of content of JSON string that is to be parsed. For e.g. MAP_GET('bar', TO_JSON_OBJECT('{ "foo" : 1, "bar" : 2}')would yield 2	• string - The JSON string to be parsed	A parsed JSON object
TO_LONG	Transforms the first argument to a long integer	• input - Object of string or numeric type	Long version of the first argument
TO_LOWER	Transforms the first argument to a lowercase string	• Input -String	String
TO_STRING	Transforms the first argument to a string	• Input - Object	String
TO_UPPER	Transforms the first argument to an uppercase string	• Input -String	Uppercase string
TRIM	Trims white space from both sides of a string	• Input -String	String
URL_TO_HOST	Extract the hostname from a URL	• url - URL in string form	The hostname from the URL as a string (for example URL_TO_HOST('http://www.yahoo.com/foo') would yield 'www.yahoo.com'
URL_TO_PATH	Extract the path from a URL	• url - URL in string form	The path from the URL as a string (for example URL_TO_PATH('http://www.yahoo.com/foo') would yield 'foo'
URL_TO_PORT	Extract the port from a URL. If the port is not explicitly stated in the URL, then an implicit port is inferred based on the protocol.	• url - URL in string form	The port used in the URL as an integer (for example URL_TO_PORT('http://www.yahoo.com/foo') would yield 80)
URL_TO_PROTOCOL	Extract the protocol from a URL	• url - URL in string form	The protocol from the URL as a string (for example URL_TO_PROTOCOL('http://www.yahoo.com/foo') would yield 'http'
WEEK_OF_MONTH	The numbered week within the month. The first week within the month has a value of 1.	• dateTime -The datetime as a long representing the milliseconds since UNIX epoch	The numbered week within the month
WEEK_OF_YEAR	The numbered week within the year. The first week in the year has a value of 1.	• dateTime - The datetime as a long representing the milliseconds since UNIX epoch	The numbered week within the year
YEAR	The number representing the year	• dateTime -The datetime as a long representing the milliseconds since UNIX epoch	The current year
ZIP	Zips lists into a single list where the ith element is an list containing the ith items from the constituent lists. See python and wikipedia for more context.	• lists* - Lists to zip.	• Returns: The zip of the lists. The returned list is the min size of all the lists. e.g., ZIP([1, 2], [3, 4, 5]) == [[1, 3], [2, 4]]

Function	Description	Input	Returns
ZIP_LONGEST	Zips lists into a single list where the ith element is an list containing the ith items from the constituent lists. See python and wikipedia for more context.	• lists* - Lists to zip.	• Returns: The zip of the lists. The returned list is the max size of all the lists. Empty elements are null e.g., ZIP_LONGEST([1, 2], [3, 4, 5]) == [[1, 3], [2, 4], [null, 5]]

The following is an example query (in other words, a function which returns a boolean) which would be seen possibly in threat triage:

```
IN_SUBNET( ip, '192.168.0.0/24') or ip in [ '10.0.0.1', '10.0.0.2' ] or
 exists(is_local)
```

This evaluates to true precisely when one of the following is true:

- The value of the ip field is in the 192.168.0.0/24 subnet
- The value of the ip field is 10.0.0.1 or 10.0.0.2
- The field is_local exists

The following is an example transformation which might be seen in a field transformation:

```
TO_EPOCH_TIMESTAMP(timestamp, 'yyyy-MM-dd HH:mm:ss', MAP_GET(dc, dc2tz,
 'UTC'))
```

For a message with a timestamp and dc field, we want to set the transform the timestamp to an epoch timestamp given a timezone which we will lookup in a separate map, called dc2tz.

This will convert the timestamp field to an epoch timestamp based on the

- Format yyyy-MM-dd HH:mm:ss
- The value in dc2tz associated with the value associated with field dc, defaulting to UTC

Stellar Benchmarks

A microbenchmarking utility is included to assist in executing microbenchmarks for Stellar functions.

The utility can be executed via maven using the `exec` plugin, like so, from the `metron-common` directory:

```
mvn -DskipTests clean package && \
mvn exec:java -
Dexec.mainClass="org.apache.metron.common.stellar.benchmark.StellarMicrobenchmark"
 -Dexec.args="..."
```

where exec.args can be one of the following:

```
    -e,--expressions <FILE>   Stellar expressions
    -h,--help                 Generate Help screen
    -n,--num_times <NUM>      Number of times to run per expression (after
                              warmup). Default: 1000
    -o,--output <FILE>        File to write output.
    -p,--percentiles <NUM>    Percentiles to calculate per run. Default:
                              50.0,75.0,95.0,99.0
    -v,--variables <FILE>     File containing a JSON Map of variables to use
    -w,--warmup <NUM>         Number of times for warmup per expression.
                              Default: 100
```

For instance, to run with a set of Stellar expression in file /tmp/expressions.txt:

```
```
 # simple functions
 TO_UPPER('john')
 TO_LOWER(name)
 # math functions
 1 + 2*(3 + int_num) / 10.0
 1.5 + 2*(3 + double_num) / 10.0
 # conditionals
 if ('foo' in ['foo']) OR one == very_nearly_one then 'one' else 'two'
 1 + 2*(3 + int_num) / 10.0
 #Network funcs
 DOMAIN_TO_TLD(domain)
 DOMAIN_REMOVE_SUBDOMAINS(domain)
```

And variables in file /tmp/variables.json:

```
{
  "name" : "john",
  "int_num" : 1,
  "double_num" : 17.5,
  "one" : 1,
  "very_nearly_one" : 1.000001,
  "domain" : "www.google.com"
}
```

Written to file /tmp/output.txt would be the following command:

```
mvn -DskipTests clean package && \
mvn exec:java -
Dexec.mainClass="org.apache.metron.common.stellar.benchmark.StellarMicrobenchmark"
 \
-Dexec.args="-e /tmp/expressions.txt -v /tmp/variables.json -o ./
output.json"
```
```