Avi Vantage Configuration Guide v16.1

Table of Contents

Avi Vantage DataScript Guide

About

Most functionality desired from Avi Vantage can be configured directly through the GUI, CLI, or API. However, there may be occasions where the native feature set is not robust enough to cater to a specific use case. For this purpose Avi provides DataScript, which is a Lua based scripting environment capable of adding significant extensibility to the Avi Vantage functionality. DataScripts are more advanced versions of the Policies, which similarly allow inspection and manipulation of client and server traffic. DataScripts can be used to inspect client HTTP requests or server HTTP responses and perform content switching, redirection, header manipulation, logging, and more. The DataScript scripting language is built upon an embedded Lua interpreter, with additional Avi specific libraries added to extend the power. The basic DataScript syntax is based on Lua, with additional commands, called functions, available for Avi specific tasks.

For in-depth help on Lua scripting syntax and usage, see:www.lua.org ›

DataScripts are executed within the data plane on the Service Engines hosting the virtual service to which the DataScript has been attached. DataScript is different than ControlScript, which is a Python based scripting tool for automation of the control plane and executed from the Avi Controller. A typical DataScript will typically be in some form of if / then logic, similar to a Policy's match / action logic.

```
path = avi.http.get_path()
if string.beginswith(path, "/docs/") then
  avi.redirect("http://docs.avinetworks.com/index.html")
end
```

Assigning to a Virtual Service

DataScripts are reusable objects that are maintained within: *Templates* > *Scripts* > *DataScripts*. They are attached to a virtual service by editing a VS and navigating to: *Virtual Service Edit* > *Rules* > *DataScript* and select *New DataScript* to choose an existing DataScript or create a new one. A virtual service may have multiple DataScripts. Also, a DataScript may be used by multiple virtual services.

The exception is when a DataScript references a specific pool, such as the example shown, in which case the DataScript may only be used by one virtual service at a time.

Recommended Reading

- Events: Each DataScript will executed when an event is triggered, such as when an HTTP request or HTTP response is received by the virtual service. A DataScript must be executed within one or more events.
- Operators: DataScripts may make use of operators, which may be relational, logical, or arithmetic.
- Functions: Functions are Avi specific commands, such as HTTP redirects or closing a client connection.
- Execution Priority: DataScripts have a complex relationship with other features within Avi Vantage, and it is important to understand the order of execution priority when multiple DataScripts are attached to a virtual service, or Policies and other features conflict with the DataScripts.
- Troubleshooting: DataScript's may fail to save / load at time of configuration, or they may be created without issue but fail to execute when applied to traffic.

DataScript Object Model

DataScripts are attached to Virtual Services. Each DataScript will executed when an event is triggered, such as an HTTP request or HTTP response is received by the Virtual Service. Each DataScript objects will include at least one, and potentially more events. A Virtual Service may have multiple DataScripts attached. If multiple DataScripts use the same events, such as HTTP request, then the order of the DataScripts assigned to the Virtual Service will be taken into account for the order of processing of the DataScripts. For instance, if the first DataScript is set to discard all client requests to the /secure directory, and the second DataScript is set to redirect all authenticated clients making requests for the /secure directory to a different path, the second DataScript will never be executed. The supported events are:

- **HTTP_REQ** - This event triggers when the request line and all the headers of the HTTP request have been parsed successfully, but before any potential POST body has been received.
- **HTTP_RESP** - This event triggers when the response status line and all headers of the HTTP response have been parsed successfully, but before the response body has been received.

This object model allows the flexibility to create a library of DataScripts. A Virtual Service can be configured to refer to any DataScript object from this collection, provided it is of the same protocol

as the DataScript's events. To use a DataScript with an HTTP request event, the Virtual Service must be configured for application type HTTP. Each Virtual Service object can refer to any number of DataScripts. All DataScripts are parsed during configuration time. Hence, any incorrect DataScript would result in a failure when attempting to save the new or modified DataScript. The configuration time error is reported as an aid in debugging. When a DataScript execution encounters a failure, the script execution aborts for that HTTP request or response. An HTTP Internal Server Error is sent to the client and a client log is generated with the stack trace of the aborted script to aid in debugging.

DataScript Constructs

Operators

Operators compare or contrast sets of data, and will return as true or false. DataScripts rely on the Lua scripting language for the syntax and supported usage of arithmetic, relational and logical operators. When evaluating strings, DataScripts is case sensitive, so "a" does not equal "A".

Arithmetic Operators

```
+ — Addition
- — Subtraction
* — Multiplication
/ — Division
```

Relational Operators

```
> — Greater than
< — Less than
>= — Greater than or equal
<= — Less than or equal
== — Equal
~= — Not equal
```

Logical Operators

```
and
or
not
string.find — Used to search for a string within another string.
```

DataScript Precedence

Many DataScript capabilities can be performed via Policies and potentially standard features. It is important to understand the order of precedence, as this will dictate the order that various functions will occur. For instance, if a network security policy is set to discard a connection, a DataScript set to redirect certain HTTP requests may never see the client.

Responses generated by Avi DataScripts or Avi Policies are not evaluated by DataScripts. For instance, an HTTP response generated by the HTTP Request Policy cannot be inspected or modified by a DataScript.

More than one DataScript may be applied to a single Virtual Service. The order of the DataScripts is important, as DataScripts with the same event will be executed in the order set. When adding DataScripts via the UI, use the up and down arrows next to the DataScript to reorder. The DataScript at the top of the list will execute first. Avi Vantage processes traffic in the following order of precedence:

Client Request:

- TCP Setup
- Network Security Policy
- SSL Handshake
- HTTP Security Policy
- HTTP Request Policy
- DataScript Request Event
- Normal VS/Pool/Profile Processing

Server Response:

- DataScript Response Event
- HTTP Response Policy

In the order of precedence, individual features may be inserted at different points based on the functionality of the feature. For instance, connection throttling will occur during step 1 of the client request, not step 7.

DataScript Samples

Example

- Generate a different redirect URL based on different host names.
- Check if the host name begins with “www.avi.”, then get the ccTLD from the host.
- Form the beginning part of the new query using the ccTLD map table to get the value of the query argument “mandate”.
- Append the old query, if it exists, to the new query.
- Generate the HTTP redirect to the new URL using the domain “www.avi.com” and the new query.

```
local ccTLD_map = {es="01345F", fi="5146FF", cn="45DFX", io="123456", ly="ABC123"}
host = avi.http.host()

if not host then
  avi.http.close_conn()
elseif string.sub(host, 1, #"www.avi.") == "www.avi." then
  i,j = string.find(host, "avi")
  ccTLD = string.sub(host, j+2, -1)
  new_query = "?mandate=" .. ccTLD_map[ccTLD]
  old_query = avi.http.get_query()

  if #old_query > 0 then
    new_query = new_query .. "&" .. old_query
  end
    avi.http.redirect("www.avi.com" ..
      avi.http.get_path() ..
      new_query, 302)
end
```

DataScript Functions

DataScripts are comprised of any number of function or method calls which can be used to inspect and act on traffic flowing through a virtual service. DataScript’s functions are exposed via Lua libraries and grouped into modules: *string, vs, http, pool, ssl and crypto*. Other Lua libraries may also be used, following the documentation from www.lua.org. The following functions are

available:

String	
string.beginswith(source, target)	Search for string in beginning of a string
string.endswith(source, target)	Search for string at the end of a string
string.contains(source, target)	Search contains a string in another string
string.upper(source)	Change a string to upper case
string.lower(source)	Change a string to lower case
string.len(source)	Returns number of characters in string
string.sub(source, begin, [end])	Extract a sub-string from a string
VS	
avi.vs.ip()	Returns IP address of the VS
avi.vs.name()	Returns the name of the VS
avi.vs.client_ip()	Returns the client IP address
avi.vs.client_port()	Returns the client source port
avi.vs.log()	Write a custom log to VS > client logs
avi.vs.table_insert([table_name,] key, value [, lifetime])	Store custom data in a time based table
avi.vs.table_lookup([table_name,] key [, lifetime_exten])	Lookup data in a table
avi.vs.table_remove([table_name,] key)	Remove data from a table
avi.vs.table_refresh([table_name,] key [, lifetime_exten])	Update the expire time for a table entry
avi.vs.reqvar.*	Set a global variable usable across

	events
HTTP	
avi.http.get_uri([false])	Returns the URI (path plus query)
avi.http.set_uri(new_uri)	Change the URI
avi.http.get_path([false])	Returns the URI's path /path.index.htm
avi.http.set_path(new_uri)	Modify the path of a request
avi.http.get_query([arg_name | avi.QUERY_TABLE] [, decode])	Returns the URI's query ?a=1&b=2
avi.http.set_query(integer | string | table)	Modify the query of a request
avi.http.redirect(uri [,status])	Redirect a request
avi.http.response(status, [headers, [body]])	Send a defined HTTP response page
avi.http.close_conn([reset])	Close or reset a TCP connection
avi.http.get_path_tokens([start [, end]])	Return a subsection of the path
avi.http.get_host_tokens([start [, end]])	Return a subsection of the host
avi.http.get_header([[name] [context]])	Return header names or their values
avi.http.add_header(name, value)	Insert a new header and value
avi.http.replace_header(name, value)	Replace an existing headers value
avi.http.remove_header(name)	Remove all instances of a header
avi.http.get_cookie(name [, context])	Return the names or values of cookies
avi.http.method()	Return the client's request method
avi.http.hostname()	Return the hostname requested by client
avi.http.protocol()	Returns the session protocol, http or https

avi.http.scheme()	Returns http:// or https://
avi.http.secure()	Returns on for https, nil for http
avi.http.get_userid()	Returns the user ID for the session
avi.http.set_userid()	Sets the user ID for the session
avi.http.status()	Returns status code to be sent to client
Pool	
avi.pool.get_servers(pool)	Returns up and total server count
avi.pool.select(pool [, server [, port]])	Pick a specific pool
avi.pool.server_ip()	Pick a specific pool server
SSL	
avi.ssl.protocol()	Return the SSL version
avi.ssl.server_name()	Return SNI name field
avi.ssl.client_cert([[avi.CLIENT_CERT] [, avi.CLIENT_CERT_FINGERPRINT] [, avi.CLIENT_CERT_SUBJECT] [, avi.CLIENT_CERT_ISSUER] [, avi.CLIENT_CERT_SERIAL]])	Returns the client's certificate, or part of it
Crypto	
avi.crypto.encrypt(key, plaintext [, iv [, algo]])	Encrypt content
avi.crypto.decrypt(key, ciphertext [, iv [, algo]])	Decrypt content