KPIs, Rules and Metrics

• KPIs, Rules and Metrics, on page 1

KPIs, Rules and Metrics

Cisco Elastic Services Controller VNF monitoring is done through the definition of Key Performance Indicators (KPIs) metrics. Core metrics are preloaded with ESC, a programmable interface gives to the end-user the ability to add and remove metrics, but also to define the actions to be triggered on specified conditions.These metrics and actions are defined at the time of deployment.

The ESC metrics and actions datamodel is divided into 2 sections:

1. KPI—Defines the type of monitoring, events, polling interval and other parameters. This includes the event_name, threshold and metric values. The event_name is user defined. The metric_values specify threshold conditions and other details. An event is triggered when the threshold condition is reached.
2. Rule—Defines the actions when the KPI monitoring events are triggered. The action element defines the actions to be performed when an event corresponding to the event_name is triggered.

Rules

The ESC object model defines for each vm_group a section where the end-user can specify the administrative rules to be applied based on the outcome of the KPIs selected metric collector.

```
<rules>
    <admin_rules>
        <rule>
            <event_name>VM_ALIVE</event_name>
            <action>TRUE esc_vm_alive_notification</action>
            <action>FALSE recover autohealing</action>
        </rule>
        : : : : : : : : : : : : : : : : :
    </admin_rules>
</rules>
```

As mentioned within the KPIs section, correlation between KPIs and Rules is done based on the value of the <event_name> tag.

In the Rules section above, if the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is TRUE, then the action identified by the key, TRUE esc_vm_alive_notification is selected for execution.

If the outcome of the KPIs defining event_name is VM_ALIVE, and the selected metric collector is FALSE, then the action identified by the key, FALSE recover autohealing is selected for execution.

For information on updating KPIs and Rules, see Updating the KPIs and Rules.

Metrics and Actions

ESC Metrics and Actions (Dynamic Mapping) framework is the foundation of the kpis and rules sections. As described in the KPIs section the metric type uniquely identifies a metric and its metadata.

The metrics and actions is as follows:

```
<metrics>
    <metric>
        <name>ICMPPING</name>
        <userLabel>ICMP Ping</userLabel>
        <type>MONITOR_SUCCESS_FAILURE</type>
        <metaData>
            <type>icmp_ping</type>
            <properties>
                <property>
                    <name>ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_events_after_success</name>
                    <value>true</value>
                </property>
                <property>
                    <name>vm_gateway_ip_address</name>
                    <value />
                </property>
                <property>
                    <name>enable_check_interface</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </metric>
    : : : : : : : : :
</metrics>
```

The above metric is identified by its unique name ICMPPING. The <type> tag identifies the metric type.

Currently ESC supports two types of metrics:

- MONITOR_SUCCESS_FAILURE
- MONITOR_THRESHOLD

The <metadata> section defines the attributes and properties that is processed by the monitoring engine.

The metric_collector type in the KPI show the following behavior:

At regular intervals of 3 seconds the behavior associated with the ICMPPING identifier is triggered. The ICMPPING metric is of type MONITOR_SUCCESS_FAILURE, that is the outcome of the monitoring action is either a success or a failure. In the sample above, an icmp_ping is performed using the <ip_address> field defined in the <metadata> section. In case of SUCCESS the rule action(s) with the TRUE prefix will be selected for execution. In case of FAILURE the rule action(s) with the FALSE prefix is selected for execution.

```
<actions>
    <action>
```

```
        <name>TRUE servicebooted.sh esc_vm_alive_notification</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>SERVICE_BOOTED</value>
                </property>
            </properties>
        </metaData>
    </action>
    : : : : : : : : :
</actions>
```

The action sample above describes the behavior associated with the SUCCESS value. The ESC rule action name TRUE servicebooted.sh esc_vm_alive_notification specifies the action to be selected. Once selected the action <type> ESC_POST_EVENT identifies the action that the monitoring engine selects.

Metrics and Actions APIs

In Cisco ESC Release 2.1 and earlier, mapping the actions and metrics defined in the datamodel to the valid actions and metrics available in the monitoring agent was enabled using the *dynamic_mappings.xml* file. The file was stored in the ESC VM and was modified using a text editor. ESC 2.2 and later do not have an *esc-dynamic-mapping* directory and *dynamic_mappings.xml* file. However, if you have an existing dynamic_mapping xml file that you want to add to the ESC VM, do the following:

1. Backup this file to a location outside of ESC, such as, your home directory.
2. Create *esc-dynamic-mapping* directory on your ESC VM. Ensure that the read permissions are set.
3. Install on your ESC VM using the following bootvm argument:

```
 --file
root:root:/opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml:<path-to-local-copy-of-dynamic-mapping.xml>
```

The CRUD operations for mapping the actions and the metrics are available through REST API. Refer to the API tables below for mapped metrics and actions definition.

To update an existing mapping, delete and add a new mapping through the REST API.

Note While upgrading any earlier version of ESC to ESC 2.2 and later, to maintain the VNF monitoring rules, you must back up the *dynamic_mappings.xml* file and then restore the file in the upgraded ESC VM. For more information upgrading monitoring rules, see Upgrading VNF Monitoring Rules section in the *Cisco Elastic Services Controller Install and Upgrade Guide*. Cisco ESC Release 2.3.2 and later, the dynamic mapping API is accessible locally only on the ESC VM.

Table 1: Mapped Actions

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/ <action_name>	GET	N/A	Action XML	Get action by name
Read All	internal/dynamic_mapping/actions	GET	N/A	Action XML	Get all actions defined
Write	internal/dynamic_mapping/actions	POST	Actions XML	Expected Action XML	Create one or multiple actions
Delete	internal/dynamic_mapping/actions/ <action_name>	DELETE	N/A	N/A	Delete action by name
Clear All	internal/dynamic_mapping/actions	DELETE	N/A	N/A	Delete all non-core actions

The response for the actions APIs is as follows:

```
<actions>
    <action>
        <name>{action name}</name>
        <type>{action type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : : :
            </properties>
        </metaData>
    </action>
    : : : : : : : : :
</actions>
```

Where,

{action name}: Unique identifier for the action. Note that in order to be compliant with the ESC object model, for success or failure actions, the name must start with either TRUE or FALSE.

{action type}: Action type in this current release can be either ESC_POST_EVENT, SCRIPT or CUSTOM_SCRIPT.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring the VNFs for more details.

Core and Default Actions List

Table 2: Core and Default Actions List

Name	Type	Description
TRUE esc_vm_alive_notification	Core	Start Service
TRUE servicebooted.sh	Core/Legacy	Start Service
FALSE recover autohealing	Core	Recover Service
TRUE servicescaleup.sh	Core/Legacy	Scale Out
TRUE esc_vm_scale_out_notification	Core	Scale Out
TRUE servicescaledown.sh	Core/Legacy	Scale In
TRUE esc_vm_scale_in_notification	Core	Scale In
TRUE apply_netscaler_license.py	Default	Apply Netscaler License

The core actions and metrics are defined by ESC and cannot be removed or updated.

The default actions or metrics are defined by ESC and exist to supplement core actions or metrics for more complex monitoring capabilities. These can be deleted and modified by the user. The default actions or metrics are reloaded on ESC startup every time an action or a metric with the same name cannot be found in the database.

Metric APIs

Table 3: Mapped Metrics

User Operation	Path	HTTP Operation	Payload	Response	Description
Read	internal/dynamic_mapping/actions/<metric_name>	GET	N/A	Metric XML	Get metrics by name
Read All	internal/dynamic_mapping/metrics/	GET	N/A	Metric XML	Get all metrics defined
Write	internal/dynamic_mapping/metrics/	POST	Metrics XML	Expected Metrics XML	Create one or multiple metrics
Delete	internal/dynamic_mapping/actions/<metric_name>	DELETE	N/A	N/A	Delete metric by name
Clear All	internal/dynamic_mapping/metrics	DELETE	N/A	N/A	Delete all non-core metrics

The response for the Metric APIs is as follows:

```
<metrics>
    <metric>
        <name>{metric name}</name>
        <type>{metric type}</type>
        <metaData>
            <type>{monitoring engine action type}</type>
            <properties>
                <property>
                    <name />
                    <value />
                </property>
                : : : : : : : :
            </properties>
        </metaData>
    </metric>
    : : : : : : : : :
</metrics>
```

Where,

{metric name}: Unique identifier for the metric.

{metric type}: Metric type can be either MONITOR_SUCCESS_FAILURE, MONITOR_THRESHOLD or MONITOR_THRESHOLD_COMPUTE.

{monitoring engine action type}: The monitoring engine type are the following: icmp_ping, icmp4_ping, icmp6_ping, esc_post_event, script, custom_script, snmp_get. See Monitoring for more details.

Core and Default Metrics List

Table 4: Core and Default Metrics List

Name	Type	Description
ICMPPING	Core	ICMP Ping
MEMORY	Default	Memory compute percent usage
CPU	Default	CPU compute percent usage
CPU_LOAD_1	Default	CPU 1 Minute Average Load
CPU_LOAD_5	Default	CPU 5 Minutes Average Load
CPU_LOAD_15	Default	CPU 15 Minutes Average Load
PROCESSING_LOAD	Default	CSR Processing Load
OUTPUT_TOTAL_BIT_RATE	Default	CSR Total Bit Rate
SUBSCRIBER_SESSION	Default	CSR Subscriber Session

ESC Service Deployment

The KPI section defines the new KPI using the monitoring metrics.

```
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
```

```
    <metric_value>20</metric_value>
    <metric_cond>GT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
<kpi>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <metric_value>1</metric_value>
    <metric_cond>LT</metric_cond>
    <metric_type>UINT32</metric_type>
    <metric_occurrences_true>1</metric_occurrences_true>
    <metric_occurrences_false>1</metric_occurrences_false>
    <metric_collector>
        <type>custom_script_count_sessions</type>
        <nicid>0</nicid>
        <poll_frequency>15</poll_frequency>
        <polling_unit>seconds</polling_unit>
        <continuous_alarm>false</continuous_alarm>
    </metric_collector>
</kpi>
```

In the above sample, in the first KPI section, the metric identified by *custom_script_count_sessions* is executed at regular interval of 15 seconds. If the value returned by the metric is greater than 20, then the event name DEMO_SCRIPT_SCALE_OUT is triggered to be processed by the rules section.

In the above sample, in the second KPI section,The metric identified by custom_script_count_sessions is executed at regular interval of 15 seconds. If the value returned by the metric is less than 1, then the event name DEMO_SCRIPT_SCALE_IN is triggered to be processed by the rules section.

The rules section defines rules using the event_name that have been used by kpis. The action tag will define an action that will be executed when the event_name is triggered. In the example below, the action identified by the TRUE ScaleOut identifier is executed when the event DEMO_SCRIPT_SCALE_OUT is triggered.

```
<rule>
    <event_name>DEMO_SCRIPT_SCALE_OUT</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleOut</action>
</rule>
<rule>
    <event_name>DEMO_SCRIPT_SCALE_IN</event_name>
    <action>ALWAYS log</action>
    <action>TRUE ScaleIn</action>
</rule>
```

Script Actions

There are two types of actions supported:

1. Pre-Defined actions
2. Script actions

You can specify script execution as part of the Policy-driven data model. The *script_filename* property is mandatory to script actions, which specifies the absolute path to the script on the ESC VM. The following XML snippet shows a working example of a script action:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
    </properties>
</action>
```

The script timeout is 15 minutes by default. However, you can specify a different timeout value for each script by adding a *wait_max_timeout* property to the properties section. The following example shows how to set the timeout to 5 minutes only for this script:

```
<action>
    <name>GEN_VPC_CHASSIS_ID</name>
    <type>SCRIPT</type>
    <properties>
        <property>
            <name>script_filename</name>
            <value>/opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>
        </property>
        <property>
            <name>CHASSIS_KEY</name>
            <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
        </property>
        <property>
            <name>wait_max_timeout</name>
            <value>300</value>
        </property>
    </properties>
</action>
```

In the above example, GEN_VPC_CHASSIS_ID will have a timeout value of 300 seconds, i.e. 5 mins. ESC also has a global parameter specifying the default timeout time for all the scripts that are being executed, called SCRIPT_TIMEOUT_SEC in the MONA category. This serves as the default value unless a wait_max_timeout property is defined in the script.

Triggering Pre-defined Actions

ESC introduces a new REST API to trigger the existing (pre-defined) actions defined through the Dynamic Mapping API, when required. For more information on the Metrics and Actions APIs, see Metrics and Actions APIs, on page 3.

A sample predefined action is as follows:

```
<actions>
    <action>
        <name>SaidDoIt</name>
        <userlabel>My Friendly Action</userlabel>
        <type>SCRIPT</type>
        <metaData>
            <type>script</type>
            <properties>
```

```
                <property>
                    <name>script_filename</name>
                    <value>/opt/cisco/esc/esc-scripts/do_somethin.py</value>
                </property>
                <property>
                    <name>arg1</name>
                    <value>some_val</value>
                </property>
                <property>
                    <name>notification</name>
                    <value>true</value>
                </property>
            </properties>
        </metaData>
    </action>
</actions>
```

Note A script file located on a remote server is also supported. You must provide the details in the <value> tag, for example,
http://myremoteserverIP:80/file_store/do_somethin.py</value>http://myremoteserverIP:80/file_store/do_somethin.py</value>

The pre-defined action mentioned above is triggered using the trigger API.

Execute the following HTTP or HTTPS POST operation:

POST `http://<IP_ADDRESS>:8080/ESCManager/v0/trigger/action/`

POST `https://<IP_ADDRESS>:8443/ESCManager/v0/trigger/action/`

The following payload shows the actions triggered by the API, and the response received:

```
<triggerTarget>
  <action>SaidDoIt</action>
  <properties>
   <property>
     <name>arg1</name>
     <value>real_value</value>
   </property>
  </properties>
</triggerTarget>
```

The response,

```
<triggerResponse>
    <handle>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</handle>
    <message>Action : 'SAIDDOIT' triggered</message>
</triggerResponse>
```

ESC accepts the request, and returns a response payload and status code.

An http status code of 200 indicates that the action triggered exists, and is triggered successfully. An http status codes of 400 or 404 indicate that the action to be triggered is not found.

You can determine the status using the custom script notifications sent to NB at various lifecycle stages.

ESC sends the MANUAL_TRIGGERED_ACTION_UPDATE callback event to NB with a status message that describes the success or failure of the action execution.

The notification is as follows:

```
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
    <event_type>MANUAL_TRIGGERED_ACTION_UPDATE</event_type>
    <properties>
        <property>
            <name>handle</name>
            <value>c11be5b6-f0cc-47ff-97b4-a73cce3363a5</value>
        </property>
        <property>
            <name>message</name>
            <value>Action execution success</value>
        </property>
        <property>
            <name>exit_code</name>
            <value>0</value>
        </property>
        <property>
            <name>action_name</name>
            <value>SAIDDOIT</value>
        </property>
    </properties>
</esc_event>
```

Note The script_filename property cannot be overwritten by the trigger API request. The trigger API must not contain any additional properties that do not exist in the predefined action.

The new API allows to overrides some of the special properties (of the actions) listed below:

- Notification—Set this if your script generates progress notifications at run time. The default value is false. This value can be set to true in the action or trigger payload.
- wait_max_timeout—Wait for the script to complete the execution before terminating. The default wait timeout is 900 seconds.

Note

- The trigger API supports only script type actions.
- Ensure that the script action located on the ESC VM is copied to the same path on both the Master and Backup HA instances. For more information, see the High Availability chapter in the *Cisco Elastic Services Controller Install and Upgrade Guide*.
- The script execution terminates if there is a failover, shutdown, or reboot of the ESC services.

Configuring Custom Script Metric Monitoring KPIs and Rules

Custom Script Metric Monitoring can be performed as follows:

1. Create Script
2. Add Metric
3. Add Action
4. Define Deployment
5. Update KPI data or Rules

6. Authenticating Remote Server Using KPIs and Rules

The script to be executed has to be compliant with the rules specified for a MONITOR_THRESHOLD action. Threshold crossing evaluation will be based on the exit value from the script execution. In the sample script below, the return value is the number of IP sessions.

```
#!/usr/bin/env python
import pexpect
import re
import sys
ssh_newkey = 'Are you sure you want to continue connecting'
# Functions
def get_value(key):
    i = 0
    for arg in sys.argv:
        i = i + 1
        if arg == key:
            return sys.argv[i]
    return None
def get_ip_addr():
    device_ip = get_value("vm_ip_address")
    return device_ip
# Main
CSR_IP = get_ip_addr()

p=pexpect.spawn('ssh admin@' + CSR_IP + ' show ip nat translations total')
i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==0:
    p.sendline('yes')
    i=p.expect([ssh_newkey,'assword:',pexpect.EOF])
if i==1:
    p.sendline("admin")
    p.expect(pexpect.EOF)
elif i==2:
    pass
n = p.before
result = re.findall(r'\d+', n)[0]
sys.exit(int(result))
```

The ESC monitoring and action engine processes the script exit value.

The script has to be installed into the following ESC VM directory: `/opt/cisco/esc/esc-scripts/`

The following payload describes a metric using a custom_script defined in the script

```
<!-- Demo Metric Counting Sessions -->
<metrics>
    <metric>
        <name>custom_script_count_sessions</name>
        <type>MONITOR_THRESHOLD</type>
        <metaData>
            <properties>
                <property>
                    <name>script_filename</name>
                    <value>/cisco/esc-scripts/countSessions.py</value>
                </property>
                <property>
                    <name>for_threshold</name>
                    <value>true</value>
                </property>
            </properties>
```

```
            <type>custom_script_threshold</type>
        </metaData>
    </metric>
</metrics>
<!-- -->
```

The metric payload has to be added to the list of supported ESC metrics by using the Mapping APIs.

Execute a HTTP POST operation on the following URI:

http://<my_esc_ip>:8080/ESCManager/internal/dynamic_mapping/metrics

The following payload describes custom actions that can be added to the list of supported ESC actions by using the Mapping APIs.

```
<actions>
    <action>
        <name>TRUE ScaleOut</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_Out</value>
                </property>
                <property>
                    <name>esc_config_data</name>
                    <value />
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
    <action>
        <name>TRUE ScaleIn</name>
        <type>ESC_POST_EVENT</type>
        <metaData>
            <type>esc_post_event</type>
            <properties>
                <property>
                    <name>esc_url</name>
                    <value />
                </property>
                <property>
                    <name>vm_external_id</name>
                    <value />
                </property>
```

```
                <property>
                    <name>vm_name</name>
                    <value />
                </property>
                <property>
                    <name>event_name</name>
                    <value />
                </property>
                <property>
                    <name>esc_event</name>
                    <value>VM_SCALE_IN</value>
                </property>
                <properties />
            </properties>
        </metaData>
    </action>
</actions>
```

Execute a HTTP POST operation on the following URI:

http://<IP_ADDRESS>:8080/ESCManager/internal/dynamic_mapping/actions

Custom Script Notification

ESC now supports sending notification to northbound about customized scripts run as part of the deployment at a certain lifecycle stage. You can also determine the progress of the script executed through this notification. To execute a custom script with notification, define action type attribute as *SCRIPT*, and property attribute name as *notification*, and set the value to true.

For example, in the datamodel below, the action is to run a customized script located at `/var/tmp/esc-scripts/senotification.py` with notification, when the deployment reaches POST_DEPLOY_ALIVE stage.

```
<policies>
    <policy>
        <name>PCRF_POST_DEPLOYMENT</name>
        <conditions>
            <condition>
                <name>LCS::POST_DEPLOY_ALIVE</name>
            </condition>
        </conditions>
        <actions>
            <action>
                <name>ANY_NAME</name>
                <type>SCRIPT</type>
                <properties>
                    <property>
                        <name>script_filename</name>
                        <value>/var/tmp/esc-scripts/senotification.py</value>
                    </property>
                    <property>
                        <name>notification</name>
                        <value>true</value>
                    </property>
                </properties>
            </action>
        </actions>
    </policy>
</policies>
```

You can notify northbound about the script execution progress using the following outputs:

- Standard JSON output

- REST API call
- NETCONF Notification

Standard JSON Output

The standard JSON output follows the MONA notification convention. MONA captures entries in this to generate notification.

```
{"esc-notification":{"items":{"properties":
[{"name":"name1","value":"value1"},{"name":"name2","value":"value2"}...]}}}
```

Table 5: Item list

Name	Description
type	Describes the type of notification. `progress_steps | progress_percentage | log | alert | error`
progress **Note** Progress item is required only when the type is progress-steps or progress-percentage.	For progress-steps type, `{current_step}|{total_steps}`
	For progress-percentage type, `{percentage}`
msg	Notification message.

Example JSON output is as follows:

```
{"esc-notification":{"items":{"properties": [{"name":"type",
"value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation
in progress."}]}}}
```

Note If the custom script is written in Python, because standard output is buffered by default, after each notification print statement, the script is required to call sys.stdout.flush() to flush the buffer (for pre Python 3.0). Otherwise MONA cannot process the script stdout in a real-time. `print '{"esc-notification":{"items":{"properties": [{"name":"type", "value":"progress_percentage"},{"name":"progress","value":"25"},{"name":"msg","value":"Installation in progress."}]}}}'sys.stdout.flush()`

REST API Call

```
http://localhost:8090/mona/v1/actions/notification
```

For REST API, the script must accept a script handle as the last parameter. The script handle can be UUID, MONA action or execution job Id. For example, if the script originally accepts 3 command line parameters, to support MONA notification, the script considers an additional parameter for the handle UUID. This helps MONA to identify the notification source. For every notification, the script is responsible for constructing a POST REST call to MONA's endpoint inside the script:

The payload is as follows:

```
{
 "esc-notification" : {
  "items" : {
   "properties" : [{
     "name" : "type",
     "value" : "log",
     "hidden" : false
    }, {
     "name" : "msg",
     "value" : "Log info",
     "hidden" : false
    }
   ]
  },
  "source" : {
   "action_handle" : "f82fe86d-6625-4b13-99f7-89d169e427ad"
  }
 }
}
```

Note The action_handle value is the handle UUID MONA passes into the script.