KPI、ルール、およびメトリック

KPI、ルール、およびメトリック

Cisco Elastic Services Controller VNF モニタリングは、重要業績評価指標（KPI）メトリックの定義に基づいて行われます。コアメトリックには ESC がプリロードされています。プログラム可能なインターフェイスにより、エンドユーザはメトリックを追加および削除できるだけでなく、指定された条件でトリガーされるアクションを定義できます。これらのメトリックとアクションは、展開時に定義されます。

ESC のメトリックとアクションのデータモデルは、次の 2 つのセクションに分かれています。

1. KPI：モニタリングのタイプ、イベント、ポーリング間隔、およびその他のパラメータを定義します。対象には、event_name、しきい値、およびメトリック値が含まれます。event_name はユーザが定義した名前です。metric_values ではしきい値の条件およびその他の詳細を指定します。しきい値の条件に達したときに、イベントがトリガーされます。
2. ルール：KPI モニタリングイベントがトリガーされたときのアクションを定義します。action 要素では、event_name に対応するイベントがトリガーされたときに実行されるアクションを定義します。

ルール

ESC オブジェクトモデルは、vm_group ごとにセクションを定義します。このセクションでは、エンドユーザが、選択した KPI メトリックコレクタの結果に基づいて適用する管理ルールを指定できます。

```
<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>
```

「KPI」セクションで説明したように、KPI とルールの相関は、<event_name> タグの値に基づいて実行されます。

上記の [ルール（Rules）] セクションで、event_name を定義する KPI の結果が VM_ALIVE で、選択されたメトリックコレクタが TRUE の場合、キー TRUE esc_vm_alive_notification によって識別されるアクションが実行用に選択されます。

event_name を定義する KPI の結果が VM_ALIVE であり、選択したメトリックコレクタが FALSE の場合、キー FALSE recover autohealing によって識別されるアクションが実行用に選択されます。

KPI とルールの更新については、KPI とルールの更新を参照してください。

メトリックおよびアクション

ESC メトリックおよびアクション（ダイナミックマッピング）フレームワークは、KPI およびルールセクションの基盤です。「KPI」セクションで説明したように、メトリックタイプはメトリックとそのメタデータを一意に識別します。

メトリックとアクションは次のとおりです。

```
<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>
```

上記のメトリックは、一意の名前 ICMPPING によって識別されます。<type> タグは、メトリックタイプを識別します。

現在、ESC は次の 2 種類のメトリックをサポートしています。

- MONITOR_SUCCESS_FAILURE
- MONITOR_THRESHOLD

<metadata>セクションは、モニタリングエンジンによって処理される属性とプロパティを定義します。

KPI の metric_collector タイプは、次の動作を示します。

ICMPPING 識別子に関連付けられた動作が 3 秒間隔でトリガーされます。ICMPPING メトリックのタイプは MONITOR_SUCCESS_FAILURE です。つまり、モニタリングアクションの結果は成功または失敗となります。上記のサンプルでは、icmp_ping は <metadata> セクションで定義されている <ip_address> フィールドを使用して実施されます。SUCCESS の場合、プレフィックスが TRUE のルールアクションが選択されて実行されます。FAILURE の場合、プレフィックスが FALSE のルールアクションが選択されて実行されます。

```
<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>
```

上記のアクションサンプルは、SUCCESS 値に関連付けられた動作について説明しています。ESC ルールアクション名 TRUE servicebooted.sh esc_vm_alive_notification は、選択するアクションを指定します。アクションを選択すると、<type> ESC_POST_EVENT は、モニタリングエンジンが選択するアクションを識別します。

メトリックおよびアクション API

Cisco ESC リリース 2.1 以前では、データモデルで定義されたアクションおよびメトリックから、モニタリングエージェントで使用可能な有効なアクションおよびメトリックへのマッピン

グは、dynamic_mappings.xml ファイルを使用して有効化されていました。ファイルは ESC VM に保存され、テキストエディタを使用して変更されました。ESC 2.2 以降には、*esc-dynamic-mapping* ディレクトリと *dynamic_mappings.xml* ファイルがありません。したがって、ESC VM に追加する既存の dynamic_mapping.xml ファイルがある場合は、次の手順を実行します。

1. このファイルを、ホームディレクトリなどの ESC 以外の場所にバックアップします。
2. ESC VM で *esc-dynamic-mapping* ディレクトリを作成します。読み取りアクセス許可が設定されていることを確認します。
3. 次の bootvm 引数を使用して、ESC VM にインストールします。

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

アクションとメトリックをマッピングするための CRUD 操作は、REST API を介して実行できます。マッピングされたメトリックとアクションの定義については、以下の API の表を参照してください。

既存のマッピングを更新するには、REST API を使用してそのマッピングを削除して、新しいマッピングを追加します。

（注） 以前のバージョンの ESC を ESC 2.2 以降にアップグレードする場合、VNF モニタリングルールを維持するには、dynamic_mappings.xml ファイルをバックアップしてから、アップグレードした ESC VM でファイルを復元する必要があります。モニタリングルールのアップグレードの詳細については、Cisco Elastic Services Controller インストールおよびアップグレードガイド [英語] の「Upgrade VNF Monitoring Rules」を参照してください。Cisco ESC リリース 2.3.2 以降では、ダイナミックマッピング API は ESC VM でのみローカルにアクセスできます。

表 *1:* マッピングされたアクション

ユーザ操作	パス	HTTP 動作	ペイロード	応答	説明
読み取り	internal/dynamic_mapping/actions/<action_name>	GET	該当なし	アクション XML	名前でアクションを取得する
すべて読み取り	internal/dynamic_mapping/actions	GET	該当なし	アクション XML	定義済みのすべてのアクションを取得する
作成	internal/dynamic_mapping/actions	POST	アクション XML	予期されるアクション XML	1 つまたは複数のアクションを作成する

ユーザ操作	パス	HTTP 動作	ペイロード	応答	説明
削除	internal/dynamic_mapping/actions/ <action_name>	DELETE	該当なし	該当なし	名前でアクションを削除する
すべてクリア	internal/dynamic_mapping/actions	DELETE	該当なし	該当なし	すべての非コアアクションを削除する

アクション API の応答は次のとおりです。

```
<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>
```

それぞれの説明は次のとおりです。

{action name}：アクションの一意の識別子。ESC オブジェクトモデルに準拠するために、成功または失敗のアクションの場合、名前は TRUE または FALSE で始める必要があります。

{action type}：現在のリリースのアクションタイプは ESC _POST_EVENT、SCRIPT、または CUSTOM_SCRIPT です。

{monitoring engine action type}：モニタリングエンジンタイプは、icmp_ping、icmp4_ping、icmp6_ping、esc_post_event、script、custom_script、snmp_get です。詳細については、「VNF のモニタリング」を参照してください。

コアおよびデフォルトアクションリスト

表 *2:* コアおよびデフォルトアクションリスト

名前	タイプ	説明
TRUE esc_vm_alive_notification	コア	サービスの開始
TRUE servicebooted.sh	コア/レガシー	サービスの開始
FALSE recover autohealing	コア	サービスの回復
TRUE servicescaleup.sh	コア/レガシー	スケール アウト

名前	タイプ	説明
TRUE esc_vm_scale_out_notification	コア	スケール アウト
TRUE servicescaledown.sh	コア/レガシー	スケールイン
TRUE esc_vm_scale_in_notification	コア	スケールイン
TRUE apply_netscaler_license.py	デフォルト	NetScaler ライセンスの適用

コアアクションとメトリックはESCによって定義され、削除したり、更新したりできません。

デフォルトのアクションまたはメトリックは ESC によって定義され、より複雑なモニタリング機能のコアアクションまたはメトリックを補完するために存在します。これらは、ユーザが削除および変更できます。デフォルトのアクションまたはメトリックは、同じ名前のアクションまたはメトリックがデータベースで見つからないたびに、ESC の起動時にリロードされます。

メトリック API

表 *3:* マッピングされたメトリック

ユーザ操作	パス	HTTP 動作	ペイロード	応答	説明
読み取り	internal/dynamic_mapping/actions/<metric_name>	GET	該当なし	メトリック XML	名前でメトリックを取得する
すべて読み取り	internal/dynamic_mapping/metrics/	GET	該当なし	メトリック XML	定義されているすべてのメトリックを取得する
作成	internal/dynamic_mapping/metrics/	POST	メトリック XML	予想されるメトリック XML	1 つまたは複数のメトリックを作成する
削除	internal/dynamic_mapping/actions/<metric_name>	DELETE	該当なし	該当なし	名前でメトリックを削除する
すべてクリア	internal/dynamic_mapping/metrics	DELETE	該当なし	該当なし	すべての非コアメトリックを削除する

メトリック API の応答は次のとおりです。

```
<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>
```

それぞれの説明は次のとおりです。

{metric name}：メトリックの一意の識別子。

{metric type}：メトリックタイプはMONITOR_SUCCESS_FAILURE、MONITOR_THRESHOLD、またはMONITOR_THRESHOLD_COMPUTE です。

{monitoring engine action type}：モニタリングエンジンタイプは、icmp_ping、icmp4_ping、icmp6_ping、esc_post_event、script、custom_script、snmp_get です。詳細については、「モニタリング」を参照してください。

コアおよびデフォルトアクションリスト

表 *4:* コアおよびデフォルトアクションリスト

名前	タイプ	説明
ICMPPING	コア	ICMP Ping
MEMORY	デフォルト	メモリのコンピューティング使用率
CPU	デフォルト	CPU のコンピューティング使用率
CPU_LOAD_1	デフォルト	CPU の 1 分間の平均負荷
CPU_LOAD_5	デフォルト	CPU の 5 分間の平均負荷
CPU_LOAD_15	デフォルト	CPU の15 分間の平均負荷
PROCESSING_LOAD	デフォルト	CSR の処理負荷
OUTPUT_TOTAL_BIT_RATE	デフォルト	CSR の合計ビットレート
SUBSCRIBER_SESSION	デフォルト	CSR 加入者セッション

ESC サービスの展開

KPI セクションでは、モニタリングメトリックを使用して新しい KPI を定義します。

```
<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>
```

前述のサンプルでは、最初の KPI セクションで、custom_script_count_sessions で識別されるメトリックが 15 秒間隔で実行されます。メトリックによって返される値が 20 より大きい場合、イベント名 DEMO_SCRIPT_SCALE_OUT がトリガーされ、rule セクションで処理されます。

前述のサンプルでは、2 番目の KPI セクションで、custom_script_count_sessions で識別されるメトリックが 15 秒間隔で実行されます。メトリックによって返される値が 1 未満の場合、イベント名 DEMO_SCRIPT_SCALE_IN がトリガーされ、rule セクションで処理されます。

rule セクションでは、KPI で使用されている event_name を使用してルールを定義します。action タグでは、event_name がトリガーされたときに実行されるアクションを定義します。次の例では、イベント DEMO_SCRIPT_SCALE_OUT がトリガーされると、TRUE ScaleOut 識別子によって識別されるアクションが実行されます。

```
<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>
```

スクリプトアクション

次の 2 種類のアクションがサポートされています。

1. 事前定義されたアクション

2. スクリプトアクション

ポリシー主導型データモデルの一部としてスクリプトの実行を指定できます。*script_filename* プロパティは、ESC VM 上のスクリプトへの絶対パスを指定するスクリプトアクションに必須です。次の XML スニペットは、スクリプトアクションの動作例を示しています。

```
<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>
```

スクリプトのタイムアウトは、デフォルトでは 15 分です。ただし、プロパティセクションに *wait_max_timeout* プロパティを追加することで、スクリプトごとに異なるタイムアウト値を指定できます。次に、このスクリプトにのみタイムアウトを 5 分に設定する例を示します。

```
<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>
```

上記の例では、GEN_VPC_CHASSIS_ID のタイムアウト値は 300 秒、つまり 5 分です。ESC には、実行中のすべてのスクリプトに対してデフォルトのタイムアウト時間を指定するグローバルパラメータもあり、MONA カテゴリの SCRIPT_TIMEOUT_SEC と呼ばれます。スクリプトで wait_max_timeout プロパティが定義されていない限り、これがデフォルト値として機能します。

事前定義されたアクションのトリガー

ESC では、必要に応じて、Dynamic Mapping API で定義された既存の（事前定義済みの）アクションをトリガーする新しい REST API が導入されています。メトリックおよびアクション API の詳細については、メトリックおよびアクション API （3 ページ）を参照してください。

定義済みアクションの例は次のとおりです。

```
<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>
```

（注） リモートサーバにあるスクリプトファイルもサポートされます。<value> タグに詳細を入力する必要があります。例：
http://myremoteserverIP:80/file_store/do_somethin.py</value>http://myremoteserverIP:80/file_store/do_somethin.py</value>

前述の事前定義済みアクションは、トリガー API を使用してトリガーされます。

次の HTTP または HTTPS POST 操作を実行します。

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

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

次のペイロードは、APIによってトリガーされたアクションおよび受信した応答を示します。

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

応答、

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

ESC は要求を受け入れ、応答ペイロードとステータスコードを返します。

HTTP ステータスコード 200 は、トリガーされたアクションが存在し、正常にトリガーされたことを示します。HTTP ステータスコード 400 または 404 は、トリガーされるアクションが見つからないことを示します。

さまざまなライフサイクルステージで NB に送信されるカスタムスクリプト通知を使用して、ステータスを確認できます。

ESC は、MANUAL_TRIGGERED_ACTION_UPDATE コールバックイベントを、アクションの実行の成功または失敗を示すステータスメッセージとともに送信します。

通知は次のとおりです。

```
<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>
```

（注） script_filename プロパティは、トリガー API 要求によって上書きできません。トリガー API には、事前定義済みアクションに存在しない追加のプロパティを含めることはできません。

新しい API では、次に示す（アクションの）特別なプロパティの一部をオーバーライドできます。

- [通知（Notification）]：スクリプトが実行時に進捗通知を生成する場合に設定します。デフォルト値は false です。この値は、アクションまたはトリガーペイロードで true に設定できます。
- wait_max_timeout：この時間が終了するまで、スクリプトの実行完了を待機します。デフォルトは、900 秒です。

（注）
- トリガー API は、スクリプトタイプのアクションのみをサポートします。
- ESC VM にあるスクリプトアクションが、アクティブ HA インスタンスとスタンバイ HA インスタンスの両方で同じパスにコピーされることを確認します。詳細については、『*Cisco Elastic Services Controller Install and Upgrade Guide*』の「高可用性」章を参照してください。
- ESC サービスのフェールオーバー、シャットダウン、または再起動が発生すると、スクリプトの実行は終了します。

カスタム スクリプト メトリック モニタリング KPI およびルールの設定

カスタム スクリプト メトリック モニタリングは、次の手順で実行できます。

1. スクリプトの作成
2. メトリックの追加
3. アクションの追加
4. 展開の定義
5. KPI データまたはルールの更新
6. KPI とルールを使用したリモートサーバの認証

実行するスクリプトは、MONITOR_THRESHOLD アクションに指定されたルールに準拠している必要があります。しきい値超過の評価は、スクリプト実行の終了値に基づいて行われます。次のサンプルスクリプトでは、戻り値は IP セッションの数です。

```
#!/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))
```

ESC モニタリングおよびアクションエンジンは、スクリプトの終了値を処理します。

スクリプトは、ESC VM ディレクトリ /opt/cisco/esc/esc-scripts/ にインストールする必要があります。

次のペイロードは、スクリプトで定義された custom_script を使用したメトリックを示しています。

```
<!-- 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>
<!-- -->
```

メトリックペイロードは、マッピング API を使用してサポートされる ESC メトリックのリストに追加する必要があります。

次の URI で HTTP POST 操作を実行します。

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

次のペイロードは、マッピング API を使用してサポートされる ESC アクションのリストに追加できるカスタムアクションを示しています。

```
<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>
```

次の URI で HTTP POST 操作を実行します。

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

カスタムスクリプト通知

ESCは、特定のライフサイクルステージでの展開の一環として実行される、カスタマイズされたスクリプトに関するノースバウンドへの通知の送信をサポートするようになりました。この通知によって、実行されたスクリプトの進行状況を確認することもできます。通知を使用してカスタムスクリプトを実行するには、アクションタイプ属性を SCRIPT として定義し、プロパティ属性名を notification として定義し、値を true に設定します。

たとえば、次のデータモデルでは、展開が POST_DEPLOY_ALIVE ステージに達したときに、/var/tmp/esc-scripts/senotification.py にあるカスタマイズされたスクリプトを実行します。

```
<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>
```

次の出力を使用して、スクリプトの進行状況をノースバウンドに通知できます。

- 標準 JSON 出力
- REST API コール
- NETCONF 通知

標準 JSON 出力

標準 JSON 出力は MONA 通知規則に従います。MONA は、このエントリをキャプチャして通知を生成します。

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

表 *5:* 項目品目リスト

名前	説明
タイプ	通知のタイプを示します。 `progress_steps | progress_percentage | log | alert | error`
progress （注） 進捗項目は、タイプが progress-steps または progress-percentage の場合にのみ必要です。	progress-steps タイプの場合： `{current_step}|{total_steps}`
	progress-percentage タイプの場合： `{percentage}`
msg	通知メッセージ。

JSON の出力例は次のとおりです。

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

（注） カスタムスクリプトが Python で記述されている場合、標準出力はデフォルトでバッファされるため、各通知の print ステートメントの後に、スクリプトは sys.stdout.flush() を呼び出してバッファをフラッシュする必要があります（Python 3.0 より前）。そうでない場合、MONA はスクリプト stdout をリアルタイムで処理できません。`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 コール

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

REST API では、スクリプトは最後のパラメータとしてスクリプトハンドルを受け入れる必要があります。スクリプトハンドルは、UUID、MONA アクション、または実行ジョブ ID です。たとえば、MONA 通知をサポートするためにスクリプトが元々 3 つのコマンドラインパラメータを受け入れる場合、スクリプトはハンドル UUID の追加パラメータを考慮します。これにより、MONA は通知ソースを識別できます。スクリプトは、通知ごとに、スクリプト内で MONA のエンドポイントへの POST REST コールを作成します。

ペイロードは次のとおりです。

```
{
 "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"
   }
  }
}
```

（注） action_handle 値は、MONA がスクリプトに渡すハンドル UUID です。