Monitoring Using the REST API

About Monitoring Using the REST API

Monitoring APIC Using the REST API

Proactive monitoring is an important piece of the network administrator's job but is often neglected because solving urgent problems in the network usually takes priority. However, the Application Policy Infrastructure Controller (APIC) will save network administrators time and frustration because it makes it easy to gather statistics and perform analyses. Because statistics are gathered automatically and policies are used and can be re-used in other places, human effort and error are minimized.

The following examples using the REST API can be used to drill into APIC fabric and switch components.

APIC

Monitoring APIC CPU and Memory Usage Using the REST API

The easiest way to quickly verify the health of the controllers is the APIC. Controllers provide information regarding the current status of CPU and memory utilization by creating instances of the *procEntity* class. The procEntity is a container of processes in the system. This object holds detailed information about various processes running on the APIC. The procEntity objects contain the following useful properties:

- *cpuPct*—CPU utilization
- *maxMemAlloc*—The maximum memory allocated for the system
- *memFree*—The maximum amount of available memory for the system

Procedure

Retrieve information about CPU and memory usage using the following REST API call:

Example:

```
https://apic-ip-address/api/node/class/procEntity.xml?
```

Monitoring APIC Disk Utilization Using the REST API

There are several disks and file systems present on the APIC. The REST API provides ready access to disk space utilization of all partitions on the system and can be used for monitoring this information.

Procedure

Monitor the disk and file systems on the APIC, by sending a REST API post, such as the following:

Example:

```
https://apic-ip-address/api/node/class/eqptStorage.xml?
```

Monitoring Physical Interface Statistics and Link State Using the REST API

You can use the REST API interface to poll for interface statistics. Several counters are available (for example, RX/TX, input/output / duplex, 30 second rates, 5 minute rate, unicast packets, multicast packets). Using the parent managed object, the children can be derived from it. To do this, you must have a good understanding of the object model and be able to navigate through the model to obtain the information desired using the example below.

Procedure

Step 1 Use the following base API call to get physical interface statistics:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/phys-[eth1/1].json
```

Step 2 To determine the total ingress bytes on Leaf 101 port Eth1/1, you can issue the following API call:

Example:

```
/topology/pod-1/node-101/sys/phys-[eth1/1].json
```

Step 3 Visore allows you to dig deeper into the hierarchical tree. From the prior command, the operator can see children of the interface object, such as ingress and egress bytes. The child objects include the following:

Example:

```
/topology/pod-1/node-101/sys/phys-[eth1/1]/dbgEtherStats
```

Fabric

Monitoring LLDP and CDP Neighbor Status Using the REST API

The APIC enables you to determine all LLDP or CDP neighbors in a fabric, using the REST API.

Procedure

Step 1 To determine the LLDP neighbors, send a POST such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/lldp/inst/if-[eth1/1]
```

Step 2 To determine the CDP neighbors, send a POST such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/cdp/inst/if-[eth1/1]
```

Monitoring Physical and Bond Interfaces Using the REST API

The APIC uses a bonded interface that is typically dual-homed to two leaf switches for connectivity to the Cisco ACI fabric. It also can use a bonded interface that can be dual-homed to the out-of-band management network.

- *Bond0* is the bond interface used to connect to the fabric itself (to connect to leaf switches that connect into the fabric).
- *Bond1* is the bond interface used to connect to the out-of-band segment (to connect to an OOB segment that allows setup of the APIC itself).

The bond interfaces rely on underlying physical interfaces. It is important to note that the REST API provides link information for both the physical and logical bond interfaces.

Procedure

Collect information about both the bond interfaces by sending a POST request such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-1/sys.json?querytarget=
subtree&target-subtree-class=l3EncRtdIf
```

Monitoring EPG-Level Statistics Using the REST API

To monitor network-related information for an application, you can investigate the aggregate amount of traffic to a specific tier. For example, you can monitor the amount of traffic to the web tier of a given EPG application with the REST API.

Procedure

To monitor the traffic for a new project for the epg-web-epg, send a POST request such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/uni/tn-newproject/ap-app1/epg-web-epg.xml?querytarget=
self&rsp-subtree-include=stats
```

Switches

Monitoring Switch CPU Utilization Using the REST API

Spine and leaf switch CPU utilization can be monitored using the following classes, based on the desired timescale and granularity:

- **proc:SysCPU5min**—A class that represents the most current statistics for system CPU in a 5-minute sampling interval. This class updates every 10 seconds.
- **proc:SysCPU15min**—A class that represents the most current statistics for system CPU in a 15-minute sampling interval. This class updates every 5 minutes.
- **proc:SysCPU1h**—A class that represents the most current statistics for system CPU in a 1-hour sampling interval. This class updates every 15 minutes.
- **proc:SysCPU1d**—A class that represents the most current statistics for system CPU in a 1-day sampling interval. This class updates every hour.
- **proc:SysCPU1w**—A class that represents the most current statistics for system CPU in a 1-week sampling interval. This class updates every day.
- **proc:SysCPU1mo**—A class that represents the most current statistics for system CPU in a 1-month sampling interval. This class updates every day.
- **proc:SysCPU1qtr**—A class that represents the most current statistics for system CPU in a 1-quarter sampling interval. This class updates every day.
- **proc:SysCPU1year**—A class that represents the most current statistics for system CPU in a 1-year sampling interval. This class updates every day.

The following example shows how to use these classes for monitoring:

Procedure

To view the average CPU utilization of all of the fabric switches over the last day, use XML such as in the following example:

Example:

```
https://apic-ip-address//api/node/class/procSysCPU1d.xml?
```

Monitoring Switch Fan Status Using the REST API

The following REST API call(s) and their child objects can be used to monitor the state of the fans on a leaf switch (note that there are 3 slots on this particular switch).

Procedure

To retrieve the status of the fan trays on the leaf switches, use XML such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-1
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-2
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/ftslot-3
```

Monitoring Switch Memory Utilization Using the REST API

Spine and leaf switch memory utilization can be monitored using the following classes, based on the desired timescale and granularity:

- **proc:SysMem5min**—A class that represents the most current statistics for system memory in a 5-minute sampling interval. This class updates every 10 seconds.
- **proc:SysMem15min**—A class that represents the most current statistics for system memory in a 15-minute sampling interval. This class updates every 5 minutes.
- **proc:SysMem1h**—A class that represents the most current statistics for system memory in a 1-hour sampling interval. This class updates every 15 minutes.
- **proc:SysMem1d**—A class that represents the most current statistics for system memory in a 1-day sampling interval. This class updates every hour.
- **proc:SysMem1w**—A class that represents the most current statistics for system memory in a 1-week sampling interval. This class updates every day.
- **proc:SysMem1mo**—A class that represents the most current statistics for system memory in a 1-month sampling interval. This class updates every day.

- **proc:SysMem1qtr**—A class that represents the most current statistics for system memory in a 1-quarter sampling interval. This class updates every day.
- **proc:SysMem1year**—A class that represents the most current statistics for system memory in a 1-year sampling interval. This class updates every day.

The following example shows how to use one of the classes:

Procedure

To monitor memory over the last day, use the following REST call:

Example:

```
https://apic-ip-address/api/node/class/procSysMem1d.xml?
```

Monitoring Switch Module Status Using the REST API

Even though the leaves are considered fixed switches, they have a supervisor component that refers to the CPU complex. From a forwarding perspective, there are two data-plane components: the NFE (Network Forwarding Engine) ASIC, which provides the front panel ports; and the ALE or ALE2 (Application Leaf Engine) ASIC—depending on the generation of switch hardware—which provides uplink connectivity to the spines. The following REST API example can be used to determine the status of the modules in the switch:

Procedure

To monitor the state of the supervisor and the module, use a REST API call such as the following:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/supslot-1/sup
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/lcslot-1/lc
```

Monitoring Switch Power Supply Status Using the REST API

You can use the REST API to retrieve the status of the power supplies on the leaf switches.

Procedure

To monitor the state of the power supplies on a leaf switch, use XML such as the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/psuslot-1
https://apic-ip-address/api/node/mo/topology/pod-1/node-101/sys/ch/psuslot-2
```

Note that there are 2 power supplies on this particular switch.

Monitoring Switch Inventory Using the REST API

You can use the REST API to retrieve switch hardware information such as the model and serial numbers.

Procedure

To retrieve switch hardware information, use the REST API as shown in the following example:

Example:

```
https://apic-ip-address/api/node/mo/topology/pod-1.json?query-target=children&target-subtree-class=fabricNode
```

External Monitoring

Smart Callhome

About Smart Callhome

Smart Callhome provides an email-based notification for critical system policies in a similar way as Callhome. However, Smart Callhome collects a more specific selection of faults to deliver in email messages.

Note Smart Callhome only collects and delivers faults.

The fault triggers that are typical of the Smart Callhome feature correspond to the kind of events that threaten to disrupt your network. Examples are:

- Temperature Faults: The temperature of a sensor exceeds a threshold.
- Fan/ Power Supply Faults: A fan or power supply unit goes offline.
- Disk Utilization Faults: The disk usage of a device exceeds a threshold.

Smart Callhome collects faults and emails them to a network support engineer, a Network Operations Center, or to Cisco Smart Callhome services to generate a case with the Technical Assistance Center (TAC).

Creating a Smart Callhome Destination Group Using the REST API

Procedure

Create a Smart Callhome destination group.

Example:

```
POST https://192.168.1.141/api/node/mo/uni/fabric.json

{
  "callhomeSmartGroup": {
    "attributes": {
      "name": "<destination-group-name>",
      "descr": "<description>"
    },
    "children": [
      {
        "callhomeSmartDest": {
          "attributes": {
            "name": "<destination-name>",
            "email": "<email-address>",
            format="xml"
          },
          "children": []
        }
      },

      {
        "callhomeProf": {
          "attributes": {
            "from": "<email-address>",
            "port": "<number>",
            "replyTo": "<email-address>",
            "email": "<customer-contact-email>",
            "phone": "<contact-phone-number>",
            "contact": "<name>",
            "addr": "<streeet-address>",
            "contract": "<id>",
            "customer": "<id>",¬
            "site": "<id>"
          },
          "children": [
            {
              "callhomeSmtpServer": {
                "attributes": {
                  "host": "<hostname-or-ip-address>"
                },
                "children": [
                  {
                    "file RsARemoteHostToEpg": {
                      "attributes": {
                        "tDn": "uni/tn-mgmt/mgmtp-default/oob- default"
                      },
                      "children": []
                    }
                  }
                ]
              }
            }
          ]
        }
```

```
      }
    ]
  }
}
```

TACACS External Logging

About TACACS External Logging

Terminal Access Controller Access Control System (TACACS) and Terminal Access Controller Access Control System Plus (TACACS+) are simple security protocols that provide centralized validation of users attempting to gain access to network devices. TACACS+ furthers this capability by separating the authentication, authorization, and accounting functions in modules, and encrypting all traffic between the NAS and the TACACS+ daemon.

TACACS external logging collects AAA data from a configured TACACS source (fabric-wide or tenant-only) and delivers it to one or more remote destination TACACS servers, as configured in a TACACS destination group. The collected data includes AAA session logs (`SessionLR`) such as log-ins, log-outs, and time ranges, for every Cisco Application Policy Infrastructure Controller (Cisco APIC) user, as well as AAA modifications (`ModLR`) such as the addition of a new user or a password change. Additionally, all configuration changes are logged and include the user ID and time stamp.

Creating a TACACS External Logging Destination Group Using the REST API

Procedure

Create a TACACS destination group.

Example:

```
POST https://<apic-name>/api/node/mo/uni/fabric/tacacsgroup-<groupname>.json

{
  "tacacsGroup": {
    "attributes": {
      "dn": "uni/fabric/tacacsgroup-<groupname>",
      "name": "<groupname>",
      "rn": "tacacsgroup-<groupname>",
      "status": "created"
    },
    "children": [{
      "tacacsTacacsDest": {
        "attributes": {
          "dn": "uni/fabric/tacacsgroup-<groupname>/tacacsdest-<dest-name>-port-<portno>",
         "host": "<dest-name>",
          "rn": "tacacsdest-<dest-name>-port-<portno>",
          "key": "<server secret>",
          "status": "created"
        },
        "children": [{
          "fileRsARemoteHostToEpg": {
            "attributes": {
              "tDn": "uni/tn-mgmt/mgmtp-default/oob-default",
              "status": "created"
```

```
            },
            "children": []
          }
        }]
      }
    }]
  }
}
```

Creating a TACACS External Logging Source Using the REST API

Procedure

Create a TACACS source.

Example:

```
POST https://<apic-name>/api/node/mo/uni/fabric/moncommon/tacacssrc-<src-name>.json

{
  "tacacsSrc": {
    "attributes": {
      "dn": "uni/fabric/moncommon/tacacssrc-<src-name>",
      "incl": "audits,faults",
      "name": "aaa",
      "rn": "tacacssrc-<src-name>",
      "status": "created",
      "incl":"audit,session"
    },
    "children": [{
      "tacacsRsDestGroup": {
        "attributes": {
          "tDn": "uni/fabric/tacacsgroup-<groupname>",
          "status": "created"
        },
        "children": []
      }
    }]
  }
}
```