CPS-390 Spike: Define and Agree DMI Plugin REST Interface

References

CPS-390 - Getting issue details... STATUS

RESTful API Design Specification

Issues and Decisions

#	Issue	Notes	Decision
1	How will hostname and port be provided when dmiPlugin register itself and its list of cmHandles with NCMP	The team thinks that the information should instead be provided in the form of a 'host-name' and a 'port' (there was some debate on service-name v. host-name but it was settled on host-name) e.g. "dmiPlugin" : { <host-name>, <port> } Where the host-name is unique. (the DB might assign an internal unique ID for each entry but that is just for indexing and x-referencing in a relation DB and this ID is not to be used/ exposed externally)	Instead of using 'host-names' and 'ports' parameters between java applications when in the cloud all we need is 'service-names' . The mapping of service-names to hosts and ports is done as part of the cloud configuration, in our case Kubernetes. And these are dynamic! The client application can then use a simple dns-lookup to connect to an instance of the service. Using service names also allows any plugin to use implement scaling as they see fit e.g. partitioning For the ONAP DMI Plugin which initial have only 1 instance we can simply hard-code the service-name and us the same name in the Kubernetes configuration e.g. "onap.cps.dmiPlugin"
2	Additional information in request body duplicates cmHandleId this is redundant information	Suggested to remove from request body to avoid possible error scenarios.	Only the one with the additionalInformation is needed and remove body
3	No need for Sync method, this is basic standard read operation at the root level for that cmHandle		
4	Use include 'location' property when request yang-module sources	Suggestion: do include it in the request but allow dmiPlugin to decide to use it or now. Location (this leaf is called schema in older RFC7895) is not mandatory to support in YANG library and nodes may not include it. Another alternative presumably used also by ODL itself is the <get-schema> RPC. The key difference is that the YANG module definition is sent directly over the NETCONF channel, not requiring separate file servers and clients. So this is maybe one more reason that the ONAP DMI plugin currently doesn't need the location attribute.	Location is not needed for any plugin and could only lead to ambiguity therefore will NOT be included in this request

5	Inconsistent use of "Operation" and/or HTTP Methods to distinguish write operations	Currently this page proposes to use "Operation=update" request body parameter for restconf "Replace" and "Patch" operations and use the HTTP (RESTFul) operation to distinguish between them. It also proposes to use PUT HTTP for Read and Delete operations 😞 Basically a very confusing an unintuitive use of HTTP operations to distinguish ambiguous operations that instead easily could be defined by just using the 'operation' field in the request body.	**Proposal Toine:** For Consistent (restful) design I would suggest to think as the operation to DMI-Plugin (always with body) as "creating a new order to do something" toward DMI-Plugin. Ie always a HTTP POST (or PUT?) operation. The "operation " in the body can simple be extended to include both "update" and "patch" as required. If the 'operation' is NOT supplied "read" wil be assumed as the default operation See also CPS-NCMP - DMI - SDNC Request and Response Mapping Proposal agreed by stakeholders in meeting 09 Nov 2021 • Only 'POST' method needs to be supported • use term 'Update' instead of 'replace'

DMI URI

Below table shows the proposed interface, actual implementation might deviate from this but can be accessed from

- Gerrit Source
- Read-the-docs: https://docs.onap.org/projects/onap-cps-ncmp-dmi-plugin/en/latest/design.html

DMI URI format to follow below pattern

<OP>dmi/<v{vNumber}>/ch/<cmHandle>/<data|operations|dmiAction>/ds/<datastore>/[rp:]<resourcePath>?<query>

URI	Mandatory or Optional	Description
<OP>	mandatory	the HTTP method
dmi	mandatory	the dmi root resource
<v{vNumber}>	mandatory	version of the dmi interface is the target resource URI is the query parameter list
<cmHandle>	mandatory	unique (string) identifier of a yang tree instance.
<data|operations|dmiAction>	mandatory	yang data, rpc operation or a (non-modeled) dmi action
{datastore}	mandatory	mandatory datastore
<resourcePath>	optional	the path expression identifying the resource that is being accessed by the operation. If this field is not present, then the target resource is the API itself.
<query>	optional	the set of parameters associated with the RESTCONF message; see Section 3.4 of [RFC3986]. RESTCONF parameters have the familiar form of "name=value" pairs. Most query parameters are optional to implement by the server and optional to use by the client. Each optional query parameter is identified by a URI

NCPS-NCMP - DMI Plugin Write Request Flow

See CPS-NCMP - DMI - SDNC Request and Response Mapping

Datastore

If the cmhandle metadata indicates that data is not synched in CPS then the request is forwarded to the dmiPlugin

RESTCONF/NETCONF relationship

HTTP Method	NETCONF Operation	Media Type
POST	create	application/yang.data
PUT	replace	application/yang.data
PATCH	merge	application/yang.data
PATCH	any edit operation	application/yang.patch
DELETE	delete	application/yang.data
POST	any <rpc> operation	application/yang.operation
GET	<get>, <get-config>	application/yang.data
GET	<create-subscription>	text/event-stream

- NETCONF: <config> subtree specifies data node targets
- RESTCONF: request URI specifies target resource

REST Data API

The DMI APIs for data access are similar to corresponding NCMP APIs. The following list is a summary of the main differences:

1. The URI prefix is /dmi instead of /ncmp.
2. For non-passthrough datastores, the resource path will be converted from cpsPath to RESTConfPath
3. The body for each request will contain additional information and any data provided on the NCMP interface (write operations) will be embedded in a larger JSON structure as described in example below.
4. Since all requests will have a message body, in some cases the HTTP method will be different to allow passing data. Thus POST can be used, the actual operation will be read from the body.

Request Format for Data Access

request body

```
{
  “operation”: “<operation>”,         // Valid operations are: “create”, “read”, “update”, “patch” and “delete”.

  "dataType": "<dataType>",           // e.g. "application/yang.data"

    “data”: {                           // Embedded data as a String.
    <data>                            // required for create and update operations. Optional filter-data for
read-operations
  },

  “cmHandleProperties”: {           // Additional properties for CM handle previously added by DMI plugin and
stored in NCMP.
    <properties>
  }
}
```

Below table shows the proposed interface, actual implementation might deviate from this but can be accessed from

- Gerrit Source
- Read-the-docs: https://docs.onap.org/projects/onap-cps-ncmp-dmi-plugin/en/latest/design.html

	Usecase	REST Method	URI

<table>
<tr><td>1</td><td>Add a data resource for a cmHandle</td><td>POST</td><td>{dmiRoot}/dmi/v1/ch/<cmhandle>/data/ds/ncmp-datastore:running/
{parentDataResourceIdentifier}

{
<new-yang-data-resource>
}

Content-Type: application/json

"data" payload : yang-data+json</td></tr>
<tr><td>2</td><td>Delete a data resource for a cmHandle</td><td>PUT</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:running/
{resourceIdentifier}</td></tr>
<tr><td>3</td><td>Patch a data resource for a cmHandle</td><td>PATCH</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:running/
{resourceIdentifier}

{
<yang-data-for-merging>
}

Content-Type: application/json

"data" payload : yang-data+json</td></tr>
<tr><td>4</td><td>Patch multiple child resources for a single cmHandle</td><td>PATCH</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/<dsName>/{resourceIdentifier}

Content-Type: application/json

"data" payload : yang-patch+json</td></tr>
<tr><td>5</td><td>Execute a yang action on a cmhandle instance</td><td>POST</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:operational/
{resourceIdentifier}/{action}

input: {
"param1Name" :"param1Value",
"param2Name" : "param2Value"

}

Note : If the "action" statement has no "input" section, the request message MUST NOT include a message-body</td></tr>
<tr><td>6</td><td>Execute an rpc operation</td><td>POST</td><td>{dmiRoot}/dmi/v1/operations/ch/<cmHandle>/ds/ncmp-datastore:operational/
{module-name}:{action}

{ input: {
"param1Name" : "param1Value",
"param2Name" : "param2Value"
} }

Note: If there is no "input" section, the request MUST NOT include a message-body</td></tr>
<tr><td>7</td><td>Read a filtered set of data under a data resource for a cmHandle</td><td>PUT</td><td>{dmiroot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:operational/
{resourceIdentifier}?fields={fields-expression}

<table><tr><th>Option</th><th>Description</th></tr><tr><td>fields</td><td>Request a subset of the target
resource contents</td></tr></table></td></tr>
<tr><td>8</td><td>Read data resources with specified fields under a given data resource for a given cmHandle</td><td>PUT</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:operational/
{resourceIdentifier}?fields={fields-expression}

<table><tr><th>Option</th><th>Description</th></tr><tr><td>fields</td><td>Request a subset of the target
resource contents</td></tr></table></td></tr>
<tr><td>9</td><td>Get data resource with 'fileds' for a cmhandle with a given scope condition</td><td>PUT</td><td>{dmiRoot}/dmi/v1/ch/{cmHandle}/data/ds/ncmp-datastore:operational/
{resourcepath}?fields={fields}&scope={scope}</td></tr>
</table>

<table>
<tr><td>10</td><td>Read descendant nodes to a given depth for a given cmHandle</td><td>PUT</td><td>{dmiRoot}/dmi/v1/ch/{cmHandle}/data/ds/ncmp-datastore:operational/{resourceIdentifier}?depth={level}

<table><tr><th>Option</th><th>Description</th></tr><tr><td>depth</td><td>Request limited sub-tree depth in the reply content
If '1' then only immediate resource
is retrieved
If '2' then resource plus next level resources are retrieved</td></tr></table></td></tr>
<tr><td>11</td><td>Replace data for a CMHandle</td><td>PUT</td><td>{dmiRoot}/dmi/v1/ch/<cmHandle>/data/ds/ncmp-datastore:running/{resourceIdentifier}

{data : {

.... the complete tree config to be replaced

}}</td></tr>
</table>

DMI Inventory, Model & Data Sync API

This presentation illustrates the API methods #1, #3 and #4 detailed below

Below table shows the proposed interface, actual implementation might deviate from this but can be accessed from

- Gerrit Source
- Read-the-docs: https://docs.onap.org/projects/onap-cps-ncmp-dmi-plugin/en/latest/design.html

*For response output, where applicable the yang-library format and conventions are used 'as is' or extended

#	Use Case	Rest Method	URI	Example*

1	Get module set for a cmhandle	POST	{dmiRoot}/dmi/v1/ch/cmhandle-001/modules	**Header** :
				Content-Type: application/json
				
				Request Body
				
				{
				"operation": "read",
				"cmHandleProperties ": {
				"subSystemId": "system-001"
				}
				}
				
				Response Body
				
				Response:
				"schemas": [
				{
				"moduleName": "example-identifier",
				"revision": "example-version",
				"namespace": "example-namespace"
				},
				{
				...
				}
				]
				}

2	Get yang module source for a list of modules	POST	{dmiRoot}/dmi/v1/ch/<cmHandle>/moduleResources
			
			DMI Plugin will make multiple requests to xNF and combine the result in a list
				Request Body
				
				{
				  "operation": "read",
				  "dataType": "application/json",
				  "data": {
				  "modules": [
				  {
				  "name": "pnf-sw-upgrade",
				  "revision": "2019-12-03"
				  }
				  ]
				  },
				  "cmHandleProperties": {
				  "subSystemId": "system-001"
				  }
				}
				
				Response: a list yang module references and source for each
				
				Response Body
				
				[{
				"name" : "pnf-sw-upgrade",
				"revision" : "2019-12-03",
				"yang-source": "some-source", {...}]

GET Request with body

The HTTP libraries of certain languages (notably JavaScript) don't allow GET requests to have a request body. In fact, some users are surprised that GET requests are ever allowed to have a body.

The truth is that RFC 7231—the RFC that deals with HTTP semantics and content—does not define what should happen to a GET request with a body! As a result, some HTTP servers allow it, and some—especially caching proxies—don't.

The authors of Elasticsearch prefer using GET for a search request because they feel that it describes the action—retrieving information—better than the POST verb. However, because GET with a request body is not universally supported, the search API also accepts POST requests: }

The same rule applies to any other GET API that requires a request body.
See Elasticsearch details here for more info

yang-patch operations (see rfc8072)

"create", "delete", "insert", "merge", "move", "replace", and "remove"

YANG Data Structure Extensions

https://tools.ietf.org/html/rfc8791

References

Follow principles/patterns of RESTCONF RFC-8040 https://datatracker.ietf.org/doc/html/rfc8040
Follow principles/patterns of yang-patch RFC-8072 https://datatracker.ietf.org/doc/html/rfc8040
Follow principles/patterns of RESTCONF NMDA RFC-8527 https://datatracker.ietf.org/doc/html/rfc8527