DSL API (BYOQ - Bring Your Own Query)

This page will serve as a reference for how to use the A&AI DSL API and commonly used DSL queries.

Before You Start!

It's important that you engage the AAI team before running any adhoc DSL queries. The current process is to have the AAI team vet the DSL queries to make sure the traversal used is optimal and does not cause performance issues in A&AI. As edge rules and schema may change, it is important for clients to communicate the DSL queries that they use. It's important that, at the very least, we know who is using which queries so we can be cautious of changes in the future. And we can help you find the best way to get the data you need.

Getting Started with the DSL API

To execute a DSL query, a client will perform a PUT on the dsl endpoint and include a payload that includes the dsl query to be run. The version dictates which release's REST API version the output will be based on.

```
PUT /aai/v$/dsl?format={format}
```

When calling the dsl API, the client must specify the output format as a query string. DSL API supports all formats that are supported by custom query API.

Output Formats

count

Provides a count of the objects returned in the query.

```
{
  "results": [
    {
      "generic-vnf": 4,
      "p-interface": 1,
      "vserver": 3,
      "service-instance": 1,
      "tenant": 1,
      "pserver": 1
    }
  ]
}
```

id

Provides an array of objects containing resource-type (A&AI's node type; i.e., generic-vnf) and a resource-link using the vertex ID from A&AI's graph.

```
{
  "results": [
    {
      "resource-type": "generic-vnf",
      "resource-link": "/aai/v16/resources/id/2388112"
    },
    {
      "resource-type": "generic-vnf",
      "resource-link": "/aai/v16/resources/id/4694112"
    },...]

}
```

pathed

Provides an array of objects containing resource-type (A&AI's node type; i.e., generic-vnf) and a resource-link using the A&AI REST API pathed URIs.

```
{
  "results": [
    {
      "resource-type": "generic-vnf",
      "resource-link": "/aai/v16/network/generic-vnfs/generic-vnf/lab20105v"
    },
    {
      "resource-type": "generic-vnf",
      "resource-link": "/aai/v16/network/generic-vnfs/generic-vnf/ro-stack01"
    },...]

}
```

resource

Provides the each object in the results array in the same format as A&AI's REST API with depth = 1 (first level children and cousin relationships).

```
{
  "results": [
    {
      "complex": {
        "physical-location-id": "complex5349-06",
        "complex-name": "complexsa-test",
        "resource-version": "1485403105490",
        "physical-location-type": "Example Location",
        "street1": "123 Example Street",
        "city": "Example",
        "state": "NJ",
        "postal-code": "12345",
        "country": "USA",
        "region": "USA",
        "latitude": "30.395968",
        "longitude": "-84.135344",
        "relationship-list": {
          "relationship": [
            {
              "related-to": "pserver",
              "related-link": "https://aai.onap:8443/aai/v16/cloud-infrastructure/pservers/pserver/pserver5349-
06",
              "relationship-data": [
                {
                  "relationship-key": "pserver.hostname",
                  "relationship-value": "pserver5349-06"
                }
              ],
              "related-to-property": [
                {
                  "property-key": "pserver.pserver-name2"
                }
              ]
            }
          ]
        }
      }
    },...]

}
```

resource_and_url

Provides each object in the results array in the same format as A&AI's REST API with depth = 1 (first level children and cousin relationships) plus the pathed url for the result object in A&AI.

```
 {
    "results": [
        {
            "url": "https://aai.onap:8443/aai/v16/cloud-infrastructure/complexes/complex/EXAMPLE_COMPLEX_ID",
            "complex": {
                "physical-location-id": "EXAMPLE_COMPLEX_ID",
                "complex-name": "Example",
                "resource-version": "1486837035912",
                "physical-location-type": "",
                "street1": "123 Mule St",
                "city": "MOSCOW",
                "state": "RU",
                "postal-code": "0",
                "country": "RUS",
                "region": "EMEA"
            }
        },...]
}
```

simple

Provides each result object in a simplified format. The node-type, graph vertex id, pathed url, object properties, and directly related objects in the graph are all returned. Both direct parent/child objects and cousin objects are included in the related-to array.

```
{
    "results": [
        {
            "id": "739696712",
            "node-type": "generic-vnf",
            "url": "https://aai.onap:8443/aai/v16/network/generic-vnfs/generic-vnf/85f60b5e-6eff-49c8-9a79-
550ee9eb4806",
            "properties": {
                "vnf-type": "WX",
                "service-id": "d7bb0a21-66f2-4e6d-87d9-9ef3ced63ae4",
                "equipment-role": "UCPE",
                "orchestration-status": "created",
                "management-option": "ONAP",
                "ipv4-oam-address": "10.12.14.16",
                "ipv4-loopback0-address": "18.20.22.24",
                "nm-lan-v6-address": "2001:1890:e00e:fffe::1:2806",
                "management-v6-address": "2001:1890:e00e:fffd::773",
                "vcpu": 4,
                "vmemory": 8,
                "vmemory-units": "GB",
                "vdisk": 150,
                "vdisk-units": "GB",
                "in-maint": false,
                "is-closed-loop-disabled": false,
                "resource-version": "1499958805125",
                "vnf-id": "85f60b5e-6eff-49c8-9a79-550ee9eb4806",
                "vnf-name": "VNF_NAME"
            },
            "related-to": [
                {
                    "id": "739700808",
                    "node-type": "license",
                    "url": "https://aai.onap:8443/aai/v16/network/generic-vnfs/generic-vnf/85f60b5e-6eff-49c8-
9a79-550ee9eb4806/licenses/license/EXAMPLE_LICENSE"
                },
                {
                    "id": "411750576",
                    "node-type": "service-instance",
                    "url": "https://aai.onap:8443/aai/v16/business/customers/customer/EXAMPLE_CUSTOMER/service-
subscriptions/service-subscription/EXAMPLE_SERVICE_SUBSCRIPTION/service-instances/service-instance
/EXAMPLE_SERVICE_INSTANCE"
                },
                {
                    "id": "15372328",
                    "node-type": "vnf-image",
                    "url": "https://aai.onap:8443/aai/v16/service-design-and-creation/vnf-images/vnf-image
/78252548-efb6-4b42-9cf7-2b3900c5e7e2"
                },
                {
                    "id": "700920024",
                    "node-type": "vserver",
                    "url": "https://aai.onap:8443/aai/v16/cloud-infrastructure/cloud-regions/cloud-region/Cloudy
/McCloudface/tenants/tenant/EXAMPLE_TENANT/vservers/vserver/EXAMPLE_VSERVER"
                }
            ]
        }
    ]
}
```


Walking back through relationships in the simple format:

Let's say you got back a large tree of output in the simple format and need to go through the list of objects to understand their relationships. For example, the output returned vnfs, vservers, pservers and complexes but you want to only look at the results of a particular complex. First, we'll find the JSON object for the complex by looking at the result objects for one with "node-type": "complex" and "physical-location-id": (the CLLI of the location you want to filter on) within the "properties" object. Next you would check the "related-to" object array for objects with "node-type": "pserver", take the "id"s and search for objects with those IDs in the results object array. You can keep crawling through the results in this way until you reach the objects you need. You can use this method for any property you want to filter on.

```
{
    "results": [
        ...,
        {
            "id": "14147624",
            "node-type": "complex",
            "url": "https://aai.onap:8443/aai/v16/cloud-infrastructure/complexes/complex/EXAMPLE_COMPLEX_ID",
            "properties": {
                ...
                "physical-location-id": "EXAMPLE_COMPLEX_ID",
                ...
            },
            "related-to": [
                {
                    "id": "2134056",
                    "node-type": "pserver",
                    "url": "https://aai.onap:8443/aai/v16/cloud-infrastructure/pservers/pserver/EXAMPLE_PSERVER"
                },
                {
                    "id": "2158632",
                    "node-type": "pserver",
                    "url": "https://aai.onap:8443/aai/v16/cloud-infrastructure/pservers/pserver/EXAMPLE_PSERVER2"

                },...
            ]
        },...
    ]
}
```

graphson

Provides the results using the graphson standard.

```
 {
            "id": 2213998664,
            "label": "vertex",
            "outE": {
                "hasInstance": [
                    {
                        "id": "38lint5-10m5oaw-8ph-z88l3c",
                        "inV": 2130153672,
                        "properties": {
                            "SVC-INFRA": "OUT",
                            "prevent-delete": "NONE",
                            "delete-other-v": "NONE",
                            "contains-other-v": "NONE"
                        }
                    }
                ],
                "runsOnVserver": [
                    {
                        "id": "38lio7d-10m5oaw-15w5-11yxuso",
                        "inV": 2295935016,
                        "properties": {
                            "prevent-delete": "NONE",
                            "SVC-INFRA": "OUT",
                            "delete-other-v": "NONE",
                            "contains-other-v": "NONE"
                        }
                    }
                ]
            },
            "properties": {
                "aai-last-mod-ts": [
                    {
                        "id": "38lijgp-10m5oaw-2rk5",
                        "value": 1499913739139
                    }
                ],
                "service-id": [
                    {
                        "id": "38lilft-10m5oaw-1czp",
                        "value": "d7bb0a21-66f2-4e6d-87d9-9ef3ced63ae4"
                    }
                ],
```

```
"vnf-id": [
    {
        "id": "38lik95-10m5oaw-6uit",
        "value": "2d42aa64-c3e6-455b-af80-d9cc94247dc6"
    }
],
"aai-uri": [
    {
        "id": "38liia1-10m5oaw-kw79",
        "value": "/network/generic-vnfs/generic-vnf/2d42aa64-c3e6-455b-af80-d9cc94247dc6"
    }
],
"prov-status": [
    {
        "id": "38lilu1-10m5oaw-afb9",
        "value": ""
    }
],
"equipment-role": [
    {
        "id": "38lim89-10m5oaw-agw5",
        "value": ""
    }
],
"aai-created-ts": [
    {
        "id": "38ligp5-10m5oaw-c3d1",
        "value": 1499913739138
    }
],
"source-of-truth": [
    {
        "id": "38ligax-10m5oaw-4kcl",
        "value": "SDNC"
    }
],
"vnf-type": [
    {
        "id": "38lil1l-10m5oaw-9og5",
        "value": "SW"
    }
],
"aai-node-type": [
    {
        "id": "38liio9-10m5oaw-1n9h",
        "value": "generic-vnf"
    }
],
"orchestration-status": [
    {
        "id": "38limmh-10m5oaw-bjlx",
        "value": "Created"
    }
],
"in-maint": [
    {
        "id": "38lin0p-10m5oaw-3sp1",
        "value": false
    }
],
"resource-version": [
    {
        "id": "38lij2h-10m5oaw-6io5",
        "value": "1499913739139"
    }
],
"last-mod-source-of-truth": [
    {
        "id": "38lijux-10m5oaw-79j9",
        "value": "SDNC"
    }
],
"is-closed-loop-disabled": [
    {
        "id": "38linex-10m5oaw-6net",
        "value": false
    }
```

```
                ],
                "vnf-name": [
                    {
                        "id": "38liknd-10m5oaw-85xh",
                        "value": "EXAMPLE_VNF_NAME"
                    }
                ]
            }
        },...]
}
```

Optional Query Parameters

Depth

You can pass a depth query parameter on the resource or resource_and_url formats to indicate what level of child objects you want returned. By default the output will be depth = 1 (first level children).

```
PUT /aai/v$/query?format={resource OR resource_and_url}&depth=0
```

Nodes Only

You can pass a nodes only query parameter to have the output only contain the object properties with no relationships. By default the output will be of depth = 1 (first level children and cousin relationships).

```
PUT /aai/v$/query?format={format}&nodesOnly=true
```

Subgraph

You can pass a subgraph query parameter that determines the behavior of the output. By default, a query returns all of the objects from the query and all of their relationships. Using subgraph=prune returns all of the objects from the query and only the edges between those objects. Using subgraph=star returns all of the objects from the query plus all of the objects they relate to.

```
PUT /aai/v$/query?format={format}&subgraph={subgraph}
```

Payload

Typically the dsl payload will include a "dsl" portion that includes the dsl query to be run.

```
{
    "dsl":"customer> service-subscription('service-type','example-service-type') > service-instance > generic-
vnf*('vnf-type','example-vnf-type')('model-version-id-local','1.0')
  "
}
```

DSL Query Format

Syntax

Symbol	Examples	Description
[]	[node1, node2]	Used for creating a union between nodes.
()	node ('property','filter') node(> node-filter)	Used for filtering the current node. Can be used with properties or filtering by related nodes, and multiple filters can be applied at once.
*	node* node* ('property','filter')	Used to return the current node, assuming it meets any filters.
>	node1 > node2	Used to traverse between nodes

DSL Requirements and Best Practices

Requirements

1. Start Node in the DSL Query should include filtered properties that are keys or indexed properties. These come from a special property in oxm (DslStartNodeProperties). The below will error out
    a. cloud-region*
    b. pserver('prov-status','PROV')
    c. ERR.5.4.6149 : DSL Error while processing the query :Non indexed keys sent. Valid keys for cloud-region cloud-owner,cloud-region-id, cloud-type
2. There should be atleast one return node type specified. If the user does not specify a "*" or store('x') on any node type it errors out
    a. generic-vnf('vnf-type','SW')('equipment-role','UCPE') > service-instance > service-subscription
    b. ERR.5.4.6152: No nodes marked for output
3. Check if a valid edge exists between the node types specified and labels
    a. ERR.5.4.6107: No EdgeRule found for passed nodeTypes: cloud-region, l-interface
    b. ERR.5.4.6107: No rules found for EdgeRuleQuery with filter params node type: cloud-region, node type: complex, label: org.onap.relationships.inventory.Belongs, type: TREE, isPrivate: false
4. Check for redundant edges to prevent the possibility of loops to occur from a DSL query.
   A > B, B > C, C > A
    a. ERR.5.4.6151: Validation error :Loop Validation failed
5. Check if the dsl has a max type of vertices returned or traversed - if beyond that then the client should use CQ
    a. ERR.5.4.6151: Validation error :NodeCount Validation failed"
6. Syntax Errors are caught at the grammar level
    a. ERR.5.4.6153: DSL Syntax Error while processing the query
7. Lower Timeouts for DSL Queries
    a. DSL Queries have a lower timeout than custom queries or any other APIs in resources or traversal to prevent the client from running adhoc queries
8. aa

Best Practices

1. Verify if the DSL traversal is optimal. Run .profile() on the underlying gremlin query
2. If available always use keys or unique indexes in the start node or any node in the query
3. Frequently used DSL queries should be encouraged to be converted to custom-queries since the start-node in a custom query has a URI
4. Run the format=count and verify if the traversal can be reversed to get the optimal query

Examples

Node Query(filter based on properties or negation)

Give me a vertex that have the given properties

```
NODE*('key1','value1')('key2','value2')
NODE -> A&AI node type
* -> store(x) or select *

{
    "dsl":"generic-vnf*('vnf-type','example-vnf-type')('model-version-id-local','1.0') "
}
```

Give me a vertex that do not have the given properties

```
NODE*!('key1','value1')('key2','value2')

{
    "dsl":"generic-vnf*('vnf-type','example-vnf-type')!('model-version-id-local','1.0') "
}

Give me a generic-vnf with the given vnf-type and model-version-id is NOT 1.0
```

Traversal Query

Traverse through cousin and children

```
NODE*('key1','value1')('key2','value2') > NODE2 > NODE3*('key3','value3')
> -> edge traversal

{
    "dsl":"customer> service-subscription('service-type','example-service-type') > service-instance > generic-
vnf*('vnf-type','example-vnf-type')('model-version-id-local','1.0')
  "
}
```

Where Query(Filter based on a traversal or a subtraversal)

```
NODE('key1','value1') > NODE2*(>NODE3('key3','value3'))

{
    "dsl":"customer> service-subscription('service-type','example-service-type') > service-instance(> generic-
vnf*('vnf-type','example-vnf-type')('model-version-id-local','1.0') )"
}

Give me service-instances with the given service-type WHERE service-instance is related to a generic-vnf with
the given properties
```

Union Query

```
NODE('key1','value1') > [ NODE2*(>NODE3('key3','value3')), NODE3*]

{
    "dsl":"customer> service-subscription('service-type','example-service-type') > [ service-instance(> generic-
vnf*('vnf-type','example-vnf-type')('model-version-id-local','1.0') ) , tenant*]"
}

Give me service-instances and tenant related to the given service-subscription
```