AAI REST API

Contents

1. Overview

The AAI REST API provides access to the AAI active inventory graph. The API is largely configured off of models and configuration files. Each vertex in the graph has an API that can be called separately or, if part of a tree structure, as a nested element with one or more generations (parent, grandparent, etc.).

The edges of the graph are provisioned using a relationship list construct. For PUT methods, a relationship contains the vertex type or category (related-to) and a list of relationship data which captures the key pieces of data required to uniquely identify the resource. On a GET method, the above information and a URL are returned. The URL can be used to GET all the details of that object. The URL returned is suitable for retrying failed commands but should not be expected to be cacheable for very long periods (e.g., the version of the URL may get deprecated when the release changes).

Concurrency control for AAI is in place. The assumptions made for the implementation are as follows:

- A client always gets a resource before updating through PUT or deleting it.
- All resource updates and deletions are done via the AAI REST APIs
- This solution will apply to PUT and DELETE operations.
- The resource-version attribute is now in every container
- The update API is not subject to concurrency control because it is only intended to be used by clients who are the definitive source of truth for the attributes they are changing. An update through the update API will however reset the resource-version so clients using PUT and DELETE will not risk updating with stale data.
- The PATCH REST verb is not subject to concurrency control because it is only intended to be used by clients who are the definitive source of truth for the attributes they are changing. An update through the update API will however reset the resource-version so clients using PUT and DELETE will not risk updating with stale data.

2. How to Use this Document

When you click on the API documentation, you will see the Summary of APIs broken down by namespace (e.g., cloud-infrastructure, business, network, service-design-and-creation).   You can search for **Tag:** (matching the explicit case) to move from namespace to namespace through the Summary.

Search for **Paths** to skip past the Summary section where there will be more detail about each API.   Query parameters are provided here, as well as links to our error codes.

Search for **Schema definitions** to see the definitions of the payloads. In your browser URL, you can type /#/definitions/node-name at the end of the html address to skip directly to a payload definition.

Note that the schema definitions now contain information about the delete scope of a node (also referenced in this document) and some related node information (also reference in this document as Edges).

Once AAI has a model and configured it, the AAI development server can be used to generate sample XML and JSON payloads, according to the Accept header passed in the request. This is done by calling the "plural" version of an API followed by the word example (e.g., /vserver/vservers/example). This returns a GET result array with one entry. That single entry can be sent in a PUT request with actual data (the resource-id does not need to be in the PUT payload as it is on the URL).

3. Document Conventions

Information that is largely guidance or aspirational will be show in *gray italicized text*. This information should not be relied upon or referenced at this point except with the understanding that it WILL change.

Bold blue text will be used to cover communication to our clients that may not be enforced by AAI. The sources of truth (our clients) populate AAI and are expected to send the correct information, having applied business rules that live in the client systems.

4. Deprecation Warnings and History

V11

API retirements:

- The actions/update API will be retired. Clients must switch to PATCH. There is one grandfathered usage for vpe update flows which will be retired in v11.
- The edge tag query will be retired.

Notable attribute and/or valid value changes (generally also impacts events):

- The persona-model-id and persona-version will be replaced with model-invariant-id (same value as persona-model-id) and model-version-id (the UUID of the specific version of a model). Persona-model-customization-id will be replaced by model-customization-id.
- The operational-state attribute will be replaced by operational-status and the only valid values will be in-service-path and out-of-service-path
- The vpn-binding object will be split in two to reflect more than one route-target per binding. The route-target will be a child of vpn-binding and some attributes will move from vpn-binding to route-target.
- The following license related attributes will be removed from generic-vnf: license-key, entitlement-assignment-group-uuid, entitlement-resource-uuid, license-assignment-group-uuid, and license-key-uuid due to the introduction of the entitlement and license children.

Event Specific:

- Normal impacts due to renaming or adding attributes, splitting objects, etc. Please see swagger documentation for objects of interest.
- In v11, clients that require lineage, children, or relationship information need to subscribe to a different DMaaP topic than the current one.

Relationship List

- The related-link will be a URI and thus not contain https://{serverroot} (impacts events)
- Thhe related-link will be used on a PUT as the "first choice" to identify the related resource. The relationship-data structure, which contains the unordered set of keys, is still an acceptable way to relate two objects but, if both the relationship-data and the related-link are passed, and they don't

agree, the related-link will be used without warning that the data is inconsistent.

- The relationship-data will be ignored on PUT.

5. AAI API Definition

The API structure is composed of:

- The HTTP command, which indicates the operation to perform
- The HTTP URI, which defines what object this operation is related to
- The HTTP version, which MUST be 1.1

Available HTTP commands are:

- PUT: used to create or update an object
- DELETE: used to delete an object or a set of objects
- GET : used to query an object or set of objects
- PATCH : used to update specific fields owned by the client doing the update

The HTTP URI is built according to this pattern:
https://{serverRoot}/{namespace}/{resource}

- (serverRoot} refers to the server base url: hostname+port+base path+version. Port and base path are OPTIONAL but AAI will use port 8443 and base path aai. The Amsterdam release version will be v11.
- {namespace} refers to the API namespace. Supported namespaces are cloud-infrastructure, business, service-design-and-creation, and network
- {resource} refers to how the object is identified according to the namespace specifications.

Example
GET https://{hostname}:8443/aai /v11/cloud-infrastructure/cloud-regions/cloud-region/{cloud-owner}/{cloud-region-id}

The GET requests support a depth query parameter allowing a query to stop after it has reached a certain point in the graph. This allows clients to minimize the data that is returned to them. A depth=0 returns the resource itself and none of its children.

5.1 Data Assumptions

Given AAI is largely a correlation engine among disparate inventory types, AAI will accept values as they are sent, without validating the format or value of the input. It is incumbent upon the source of truth to provide valid information to AAI.

Clients should do a GET prior to a PUT and change only the data that they mean to affect. The REST APIs expect the payload passed to replace the resource in AAI. **This is vital in our concurrency scheme. The client will be returned an opaque value per entity which needs to be returned back in the PUT. AAI**

will reject the PUT or DELETE if the opaque value doesn't match what AAI has stored for that entity.

If a leaf has been added to a model in vN+1, and a GET/PUT of a vN resource is done, AAI should not affect the new leaf (i.e., it should be left unchanged).

5.2 PUT and Lists

The PUT verb is used to both create and replace a resource. A given resource may have child resources (e.g., customers have service subscriptions; tenants have vservers and vservers have volumes).

The following convention will be followed:
If a resource is replaced and there are no tags for children, the children that exist will be left alone.
If a resource is replaced and there are tags for children, the children will be replaced by the list passed. If the list is empty, then children will be deleted.

Note that the relationship list is a type of child resource. The same conventions are followed. It is especially critical to ensure that you do not send an incomplete relationship list and therefore remove edges in the graph. See section 5.10 for more information on relationship lists.

5.3 PATCH

To move towards industry standards and to make our APIs easier to use by clients who own specific attributes and do not require AAI to enforce concurrency control around them, the PATCH verb has been introduced.

- RFC Algorithm implemented JSON Merge PATCH https://tools.ietf.org/html/rfc7386
- HTTP Verb = PATCH
- PATCH requires a Content-Type of "application/merge-patch+json" in your HTTP headers.
- PATCH does not support XML
- PATCH does not require a resource version to preform these modifications
- Clients should only send what they wish to modify and whose value they "own"

Example:

PATCH https://<hostname>:8443/aai/v11/network/generic-vnfs/generic-vnf/cscf0001v
```
   {
     "vnf-id": "cscf0001v", 🡐 This key needs to be here but you cannot modify the
key
     "regional-resource-zone": null,
     "ipv4-oam-address": "1.2.3.4"
}
```

This payload would result in the generic-vnf with the vnf-id = cscf0001v having ipv4-oam-address set to "1.2.3.4" and regional-resource-zone having its value removed from the database.

5.4 Referential Integrity

AAI is primarily a view to the relationships between customers, products, services, physical and virtual components, etc. It stores just the details it needs to be efficient to its tasks and knows how to get more details if needed.

As such, a transaction sent to AAI may be refused if would break referential integrity. The referential integrity rules of AAI are still evolving as we understand the services and customers that will use us.

AAI uses a graph database on a NoSQL data store. The following are true for AAI:

- Some vertices are exposed to the outside world through APIs, others are internal to how we store the data (i.e., it may look like one resource to our customers but it is expressed as more than one vertex in our graph)
- Vertices that are internal to AAI will be deleted when the parent vertex is deleted, if deletion of the parent leaves the child vertex orphaned
- Vertices that are exposed need to be managed using specific rules for each vertex.
- Vertices may have more than just parent/child relationships. One example is a vserver, which will be owned by a tenant and used by a VNF.

5.5 Delete Rules

The following options are available as actions to be take upon deletion of a resource:

- ERROR_IF_ANY_EDGES – If the resource being deleted has any edges at all, an error should be returned
- ERROR_IF_ANY_IN_EDGES – if the resource being deleted has any edges that point IN towards it, an error should be returned
- THIS_NODE_ONLY – delete the vertex being requested by first deleting its edge to other vertices, but do not delete the other vertices. Note, the delete will be rejected if the deletion target has DEPENDENT children (e.g., tenants that have vservers)
- CASCADE_TO_CHILDREN – cascade the delete through vertices who have a parentOf relationship to the vertex being deleted, as long as the vertex is orphaned by the delete of its parent
- ERROR_4_IN_EDGES_OR_CASCADE – error if there are any in edges and, if not, cascade to children

5.6 Security

All REST APIs must be called using https.

The current release is configured to support BasicAuth. 2-way SSL using client certificates should be configured for production deployments or as needed.

5.7 Headers

The following will be used for logging and interface diagnostic purposes.

- X-FromAppId Unique Application ID assigned to the user of these APIs
- X-TransactionId Unique ID that identifies an API request

The X-FromAppId will be assigned to each application by the AAI team. The X-TransactionId must be unique to each transaction within the context of an X-FromAppId.

OpenECOMP components that call AAI use the Java UUID class to generate unique ids for X-TransactionId.

The Accept header should be set to either application/json or application/xml.

Client	X-FromAppId
Policy	Policy
Master Service Orchestrator	MSO
SDN Controller	SDNC
Application Controller	APPC

5.8 Response Codes and Error Handling

HTTP response codes and error codes are described in the API documentation.

5.9 URLs Sent To and Retrieved From AAI

AAI receives URLs from clients that point back to that client in order to get more details about the data sent to AAI. AAI expects the URLs sent by clients (e.g., self links) to be URL encoded (UTF-8) and AAI will store them unchanged.

URLs that AAI constructs that point to AAI resources will be returned URLEncoded (UTF-8) to clients. This affects URLs in relationship lists and search results.

AAI expects space to be %20, and not plus(+).

5.10 The Relationship-List

The REST interface does not lend itself to creating more than parent-child relationships and the backend structure of AAI is a graph. A goal of AAI is to do as little coding as possible to introduce a new service into the service design and creation environment.

To that end, we've introduced a relationship-list structure. AAI will ask its clients to provide certain data in the relationship-list structure.

Each relationship has a related-to attribute and a list of key/value pairs. The related-to attribute identifies the node type that the resource being acted on is to be related to using the data in the key/value pairs. AAI will encode a set of rules for each resource type to verify that only valid edges are being made. AAI will keep the name of the edge itself, the directionality and cardinality, and the edge attributes within its own logic.

If an attempt is made to add a relationship to a node that doesn't exist (e.g., from a vserver to a vnf, and the vnf doesn't exist), a unique message Id (3003) will be returned with a specific error code (ERR.5.4.6129). Arguments will tell the client which node type was missing (e.g., generic-vnf) and the key data for that node type (generic-vnf.vnf-id).

Single relationships can be PUT to the graph in the following way:

https://{serverRoot}/{namespace}/{resource} /relationship-list/relationship

or

https://{hostname}:8443/aai/v11/cloud-infrastructure/pservers/pserver/pserver-123456789-01/p-interfaces/p-interface/p-interface-name-123456789-01/l-interfaces/l-interface/l-interface-name-123456789-01/relationship-list/relationship

with a payload containing the relationship information.

XML:
```
<relationship xmlns="http://org.openecomp.aai.inventory/v11">
        <related-to>logical-link</related-to>
        <relationship-data>
                <relationship-key>logical-link.link-name</relationship-key>
                <relationship-value>logical-link-123456789-01</relationship-value>
        </relationship-data>
</relationship>
```

JSON:
```
"related-to": "logical-link",
        "relationship-data": [
        {
                "relationship-key": "logical-link.link-name",
                "relationship-value": " logical-link-123456789-01"
        }
        ]
}
```

6. Edges

The following are the properties used for edge definitions. T is true, F is false

- From and To are the node types for the ends of the edges.

- EdgeLabel is the name of the label within the graph.
- Direction shows the direction of the edge.
- Multiplicity shows the multiplicity rule between two nodes. This helps govern what AAI does when modifying relationships between edges using the relationship REST APIs
- ParentOf indicates whether From is a parent of To.
- UsesResource specifies whether the From node uses resources of the To node, to be able to view the data in the context of "what uses what".
- hasDelTarget specifies whether to try to delete the To node when the From node is deleted.
- SVC-INFRA (deprecated)

The configuration for different edges supported by the AAI model are defined in the DbEdgeRules.java class.

7. Indexed Attributes

AAI supports query parameters on its indexed attributes.

As an example, if you wanted to GET a tenant by tenant-name, you would do something like

/aai/vX/cloud-infrastructure/cloud-regions/cloud-region/cloud_owner_1/cloud-region_1/tenants/tenant?tenant-name=value

The properties that are indexed are defined in the aai-schema.

8. Namespaces

8.1 Util Domain

The util domain is where AAI locates utility functions. There is currently one utility function, echo, which serves as a ping test that authenticated authorized clients can call to ensure there is connectivity with AAI.

The URL for the echo utility is:

https://load-balanced-address:8443/aai/util/echo

If the response is unsuccessful, an error will be returned following the standard format.

The successful payload returns the X-FromAppId and X-TransactionId sent by the client.

8.1.1 Successful XML Response Payload

```
<Info>
  <responseMessages>
    <responseMessage>
      <messageId>INF0001</messageId>
      <text>Success X-FromAppId=%1 X-TransactionId=%2 (msg=%3)
(rc=%4)</text>
      <variables>
        <variable>XYZ</variable>
        <variable>XYZ123</variable>
        <variable>Successful health check:OK</variable>
        <variable>0.0.0002</variable>
      </variables>
    </responseMessage>
  </responseMessages>
</Info>
```

8.1.2 Successful JSON Response Payload

```
{"responseMessages": {"responseMessage": [{
  "messageId": "INF0001",
  "text": "Success X-FromAppId=%1 X-TransactionId=%2 (msg=%3) (rc=%4)",
  "variables": {"variable":    [
    "XYZ",
    "XYZ123",
    "Successful health check:OK",
    "0.0.0002"
  ]}
}]}}
```

8.2 Cloud Infrastructure Domain

The Cloud Infrastructure domain (cloud-infrastructure) represents the assets managed within a cloud infrastructure site. This includes the physical servers, tenants, vservers and cloud-region.

8.3 Network Domain

The network namespace contains virtual and physical network resources as well as connection resources such as physical links, logical links, etc.

8.4 Business Domain

The business namespace captures customers, service subscriptions, and service instances. This domain is immature and will be evolving as service design and creation starts to gel.

8.5 Service Design and Creation

The service design and creation namespace captures data we invented based on what we thought SDC would eventually provide.

To date, there are only five containers:

1. Service-capabilities capture the pairings of service to resources.
2. Service captures the service model instances and this will be deprecated in the future as things mature
3. Models captures model definitions (subgraph definitions using the AAI widgets)
4. named-queries capture subgraph definitions that allow different data to be retrieved for a given type of asset