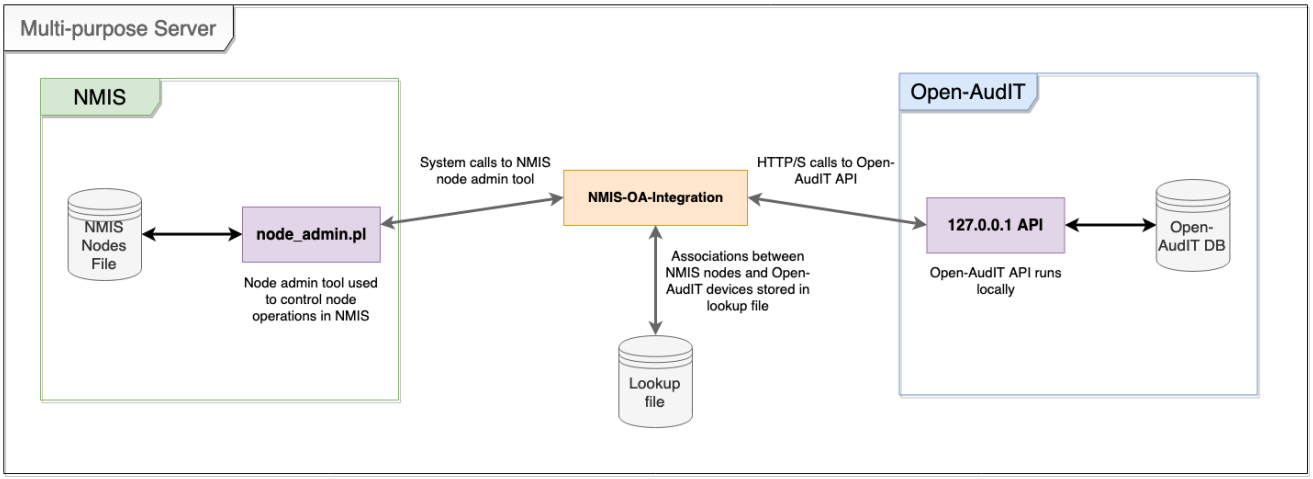
OpenAudIT-NMIS Integration

Architecture

This integration sits between and communicates with Open-AudIT and NMIS. It is designed to work as its own unit of software and makes very few assumptions about your unique setup. The integration only requires as much information about NMIS and Open-AudIT as is necessary to communicate with them.

The following diagram shows the integration running on a single, shared server running both NMIS and Open-AudIT.

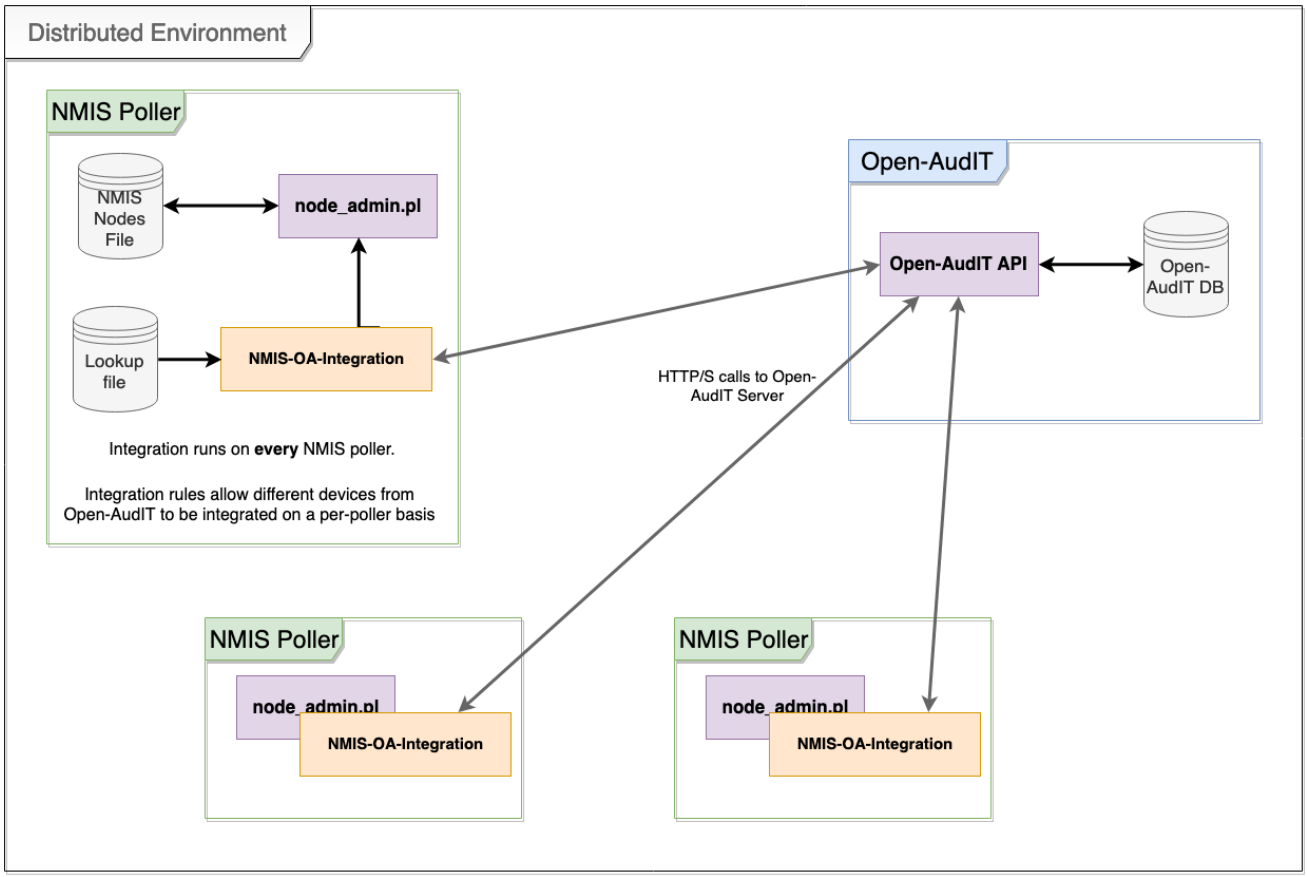
Single Server



As shown above, the integration needs to know authentication details for the Open-AudIT server, as well as the location of the NMIS node admin tool. After that, it manages its own state through a local lookup file.

As your network grows and you add NMIS pollers, you can continue to integrate with Open-AudIT on a per-poller basis by including the integration software on each poller. Each integration has its own set of rules, so it can point to any Open-AudIT server and request only the devices that poller requires. The diagram below shows an example of this architecture.

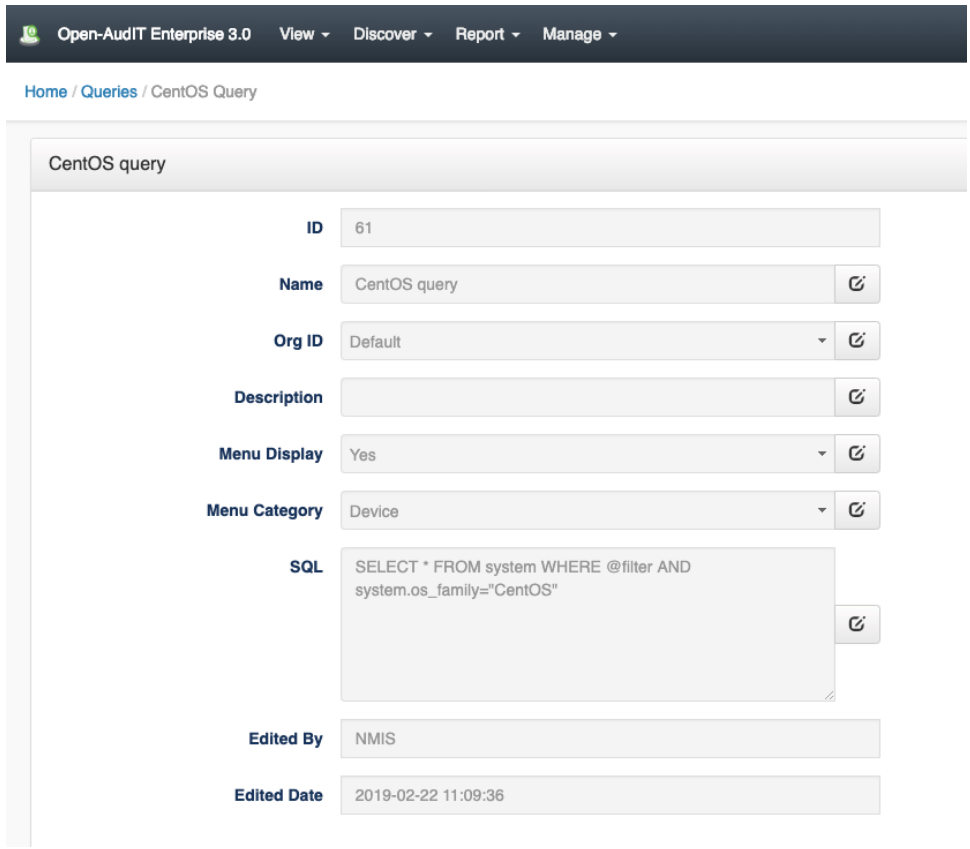
Multi Server



Choosing Devices for Integration

The integration leverages Open-AudIT's **query** feature as a way of controlling the devices you want to integrate with NMIS. You can create a query specific to your needs, then tell the integration to use that query (by referencing the name or ID). For example, if you wanted to integration only devices running CentOS, you could use the following query in Open-AudIT:

```
SELECT
* FROM
system
WHERE
@filter
AND
system.
os_famil
y="
CentOS"
```



In a scenario where your integration is distributed across multiple NMIS pollers, it's a good idea specify the poller that each device should integrate with. Open-AudIT 3.0.0+ provides an **nmis_poller** field on devices for this purpose. You could set your devices' **nmis_poller** field to be the name of a poller (perhaps using Bulk Edit) and then create a query for each poller that matches that poller's name.

You can read about Open-AudIT queries in detail here.

Integration Description

An integration run has the following steps:

1. Retrieve devices from Open-AudIT.
2. Create a node file suitable for passing into **node_admin.pl**.
3. Create a new node if one does not exist, or update an existing node if it does.
4. Update the mapping of Open-AudIT devices to NMIS nodes.
5. Update the device on the Open-AudIT server if necessary.

Configuration

The following is a sample configuration file for the integration. The configuration is written in the **.nmis** format common to other Opmantek products.

conf/nmisIntegration.nmis

```
%hash = (
  'log_path' => 'log/nmisintegration.log',
  'node_admin_path' => '/usr/local/nmis8/admin/node_admin.pl',
  'node_file_path' => '/usr/local/nmis8/conf/Nodes.nmis',
  'open_audit_details' => {
    'host' => 'https://demo.opmantek.com',
    'log_path' => 'log/openauditapi.log',
    'password' => 'OA_password',
    'user' => 'OA_user'
  },
  'open_audit_lookup_path' => 'conf/oa_nmis_lookup.nmis',
  'integration_rules_path' => 'conf/integration_rules.nmis',
  'open_audit_query_ids' => [60, 77],
);
```

Key	Type	Description
log_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path for the integration log. This log will contain a summary of actions that are taken for each integration, including system calls to **node_admin.pl.**
node_admin_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path to the executable for the node admin tool. All interactions with NMIS are controlled via the node admin tool.
node_file_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path to the Nodes.nmis file. This will be backed up before every integration run and saved as **[filename].integration.bak**.
integration_rules_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path to the rules file for the integration. The rules file defines how values from Open-AudIT devices should translate into NMIS nodes. It is recommended to give this file a **.nmis** extension, as it is written in the **.nmis** format.
open_audit_lookup_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path to the lookup file for the integration. The lookup file stores mappings between Open-AudIT devices and NMIS nodes. It is recommended to give this file a **.nmis** extension, as it is written in the **.nmis** format.
open_audit_query_ids	An **array of integers.** OR An **array of integers and strings.** (Open-AudIT 3.0.0+ only.)	The ID/s of the Open-AudIT queries to be used for this integration. See the section about Open-AudIT queries for more. If using Open-AudIT 3.0.0 or greater, you may also specify query **names** as well as IDs. For example, this value could be `'open_audit_query_ids' => ['centos_query', 11, 63, 'extra query name'],`
open_audit_details.host	A **string** representing a URL.	The Open-AudIT server you wish to target for the integration. This should include the protocol, but should NOT include any path.
open_audit_details.user	A **string**.	The username of the Open-AudIT user the integration will run under. You should ensure the chosen user has access to all the devices you wish to integration with NMIS.
open_audit_details.password	A **string**.	The password of the Open-AudIT user.
open_audit_details.log_path	A **string** representing either a fully-qualified path or a path relative to the integration's execution.	The path to the Open-AudIT log. This log will contain records of all the requests that are made to the Open-AudIT server (excluding authentication requests).

Integration Rules

The rules file defines how values from Open-AudIT devices should translate into NMIS nodes. The configuration is written in the **.nmis** format common to other Opmantek products.

Within the rules file, you define NMIS fields and give them a list of candidate values (a **ruleset**). These values can be either constants (strings), or they can refer to fields from the device itself. The integration will use the first value it can resolve or the empty string if it cannot resolve any values.

To refer to fields from the device, use the format **$DEVICE.fieldname**.

There is also the option to provide pre-built **transformations** to fields that apply after they are resolved.

An example rules file is provided below, along with an explanation of how the rulesets might resolve.

conf/integration_rules.nmis

```
%hash = (
 'nmis' => {

   # Different rules can be defined for when an NMIS node is created and when
   # it is updated. This lets you avoid overwriting values the you edit in NMIS.
   'create' => {

     # version will resolve to the 'os_version' field in the device if it exists or
     # an empty string otherwise.
     'version' => ['$DEVICE.os_version'],

     # roleType and group try to use a value from the device if it exists, but
     # it will fall back to a constant if it does not.
     'roleType' => ['$DEVICE.nmis_role', 'core'],
     'group' => ['$DEVICE.nmis_group', 'Open-AudIT'],

     # host checks multiple fields from the device, choosing the first one
     # that has a value (or the empty string if no values are found).
     'host' => ['$DEVICE.ip', '$DEVICE.hostname','$DEVICE.dns_hostname', '$DEVICE.fqdn'],

     # active, ping, and model just use a default value when any new node is created.
     'active' => ['true'],
     'ping' => ['true'],
     'model' => ['automatic'],

     # name uses an array of candidates like all the other fields, but it also applies
     # a set of transform functions to the value after a candidate is chosen.
     'name' => {
       'candidates' => ['$DEVICE.name'],
       'transforms' => ['trim_whitespace'],
     }
   },

   # Only the node fields defined here will be considered for an update.
   'update' => {

     'roleType' => ['$DEVICE.nmis_role', 'core'],
     'group' => ['$DEVICE.nmis_group', 'Open-AudIT'],

   }
 }
);
```

Transform functions

Transform functions provide additional functionality by transforming fields in some predefined way. If an invalid transform function is provided, the integration will fail. A list of valid transform functions are listed below.

Transform Function	Effect
trim_whitespace	Removes leading and trailing whitespace from the field.

Transform Example

The following table shows an Open-AudIT device, a set of transform rules, and the node structure generated after applying the transform rules. These documents have been simplified to make the changes easy to understand.

Open-AudIT Device	Integration Rules	Resulting NMIS node

<pre>{
        "name": "dbdev-1",
        "sysName": "postgres-dev-01",
    "nmis_group": "",
    "os" : "Debian GNU/Linux 9.4 (stretch)"
}</pre>
	<pre>'nmis' => {
	        'create' => {
	                'name' => ['$DEVICE.sysName'],
	                'active' => ['true'],
	                'group' => ['$DEVICE.nmis_group', 'Open-AudIT'],
	                'notes' => [],
	        }
	}</pre>
		<pre>{
		        "name": "postgres-dev-01",
		        "active": "true",
		        "group": "Open-AudIT",
		        "notes": ""
		}</pre>

Usage

To run the integration, simply invoke the executable and pass it a configuration file as described in the previous section. You can also invoke the tool by itself, which will look for a configuration file at **conf/nmisIntegration.nmis** by default.

While most of the integration is driven by the options set in the configuration file, additional options can be passed at runtime. These options can be seen in the usage instructions for the integration script.

```
# Calling the tool with a custom-named configuration file
./bin/oa-nmis-integration.pl conf=conf/my_custom_config.nmis

# Calling the tool by itself (uses conf/nmisIntegration.nmis)
./bin/oa-nmis-integration.pl

# Show additional options
./bin/oa-nmis-integration.pl -h
```

Special Cases

This section documents the behaviour that can be expected from the integration in special cases such as a node being deleted from one system or another.

Case	Behaviour
Previously integrated node is deleted from NMIS	A new node will be created in NMIS (to avoid this, update the query/device in Open-Audit)
Previously integrated device is deleted from Open-AudIT	If the flag 'delete_missing_nodes' is provided at execution time, a delete operation will run on the matching NMIS node. Note that this will also apply if the
Previously integrated node is renamed in NMIS	Similar to a node being deleted in NMIS - the integration cannot find the node it is supposed to be updating and so a new node is created.

An equivalent node existed in NMIS before the integration took place	The integration will treat it as a new node and attempt to create it in nmis, most likely resulting in a message like: `Call to node admin tool to create new node snorri failed with errmsg: Node snorri already exist.`