Orb

User Guide

Table of Contents

1. KEY CONCEPTS

Orb is an integration platform analysis and discovery application. It accepts Snaplogic pipeline definitions to build a database, that Orb users to work with. Rather than keeping the pipeline source code, Orb builds its own data model, based on selected pipeline properties and discovered relations. Orb users access its web-based front-end to browse information in a form of manageable reports.

1.1 An overview

The idea of Orb comes from an assumption, that pipeline definitions are in fact data linked together with relations. If you read all of them, together with accounts and tasks you will be able to build a model representing the entire platform. This perspective helps to answer questions such as:

- What are the most complex pipelines (and why)?
- What would be the impact of a change to a given pipeline?
- What is the difference between two Snaplogic orgs (environments)?
- Given a pipeline, what are similar pipelines and how?
- What pipelines break defined quality standards?
- Who are top contributors for the last week/month?
- What are top oldest/newest/recently changed pipelines?
- What pipelines use a given snap pack?
- What are databases / systems / endpoints used by the platform?

These are difficult to answer just by using the Snaplogic platform alone. Orb takes an approach to bring the data to one unified data store so the analysis can happen.

1.2 How pipeline definitions are interpreted

A pipeline is essentially a JSON document. It means it has a structure and is easily to parse by a programmable logic. Orb takes an advantage of this fact and reads portions of information that can be useful, in particular the following:

- Main properties such as name, path (including the organization name), author and description
- Pipeline properties (both names and default values)
- Snaps along with their settings (all you can see when you configure a Snap) and the information about the snap pack they come from
- Few properties hidden to the Snaplogic user, such as pipeline instance version Snaplogic identifiers (snode_id, instance_id)

What Orb does not read are

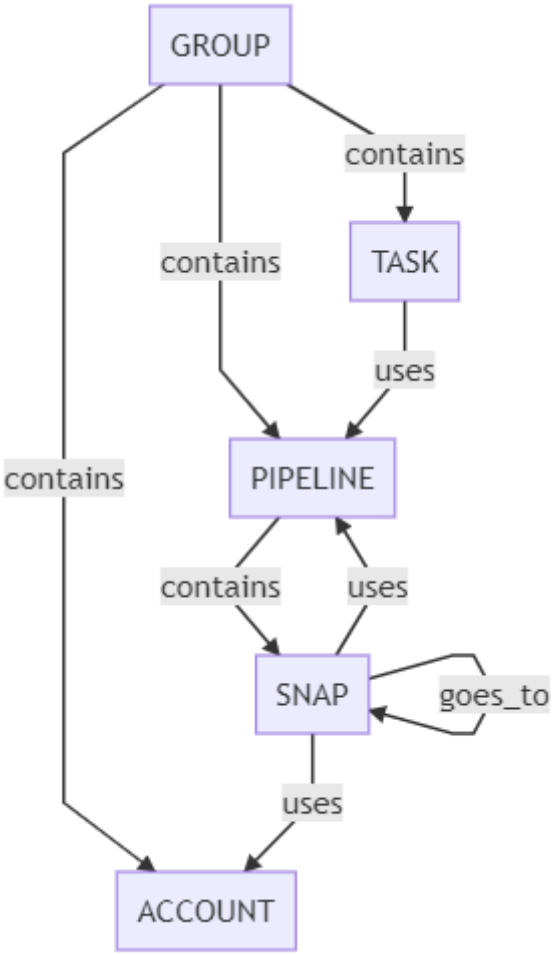
- Transformation logic such as mapping table, json/xml generator templates
- Scripts

Because Orb works with pipeline/account/tasks definitions and has no access to runtime data, all the attribute values that depend on runtime (such as expressions) will have an impact of blurring the end result. Data especially valuable is:

- Pipeline/pipeline and pipeline/account relations
- Endpoints, URLs, connection parameters

1.3 How the data model is structured

We already know that Orb reads Snaplogics definitions, but what does it make out of it? Understanding the application data model is key when searching for answers. The diagram describing it is shown below.



In a nutshell, the database is structured out of nodes and relations.

Each node has a set of properties. These are nothing but key/value pairs representing properties of Snaplogics definitions. These definitions are in fact JSON structures with properties on different levels, and node properties are their flattened versions. One special property `uid` is always present and defines the unique identifier of a node in a context of a parent (node it belongs to). For example, an `uid` of a Pipeline is unique in a context of a Group, and an `uid` of a Snap is unique in a context of a Pipeline.

Orbs underlying database allows nodes to be described by types. These types are nothing but labels and don't imply any schema requirements. A node can be labeled with any number of types.

Orb data structure assumes every node (except od Group) is of type `Entity`. Pipelines, Accounts, Tasks are additionally of type `Component`. Snaps are described by multiple types depending on their snap pack.

Type	Super Types	Relations IN	Relations OUT
Group	-	-	CONTAINS
Pipeline	Component, Entity	CONTAINS, USES	CONTAINS
Account	Component, Entity	CONTAINS, USES	-
Task	Component, Entity	CONTAINS	USES
Snap	Entity	CONTAINS	USES, GOES_TO

Relations are what glues nodes together. Orb user interface divides them into IN and OUT, because the focus is on a single entity (such as Pipeline). For example, Pipeline can contain snaps (OUT relation), but can be also referred by a snap from another pipeline (IN relation). From a data model point of view, to see a relation we need two entities, therefore there is no need to describe them as inbound or outbound. Only Group and Pipeline can `contain` and only Snap `goes_to`. The last relation `uses` is established from Task to Pipeline and from Snap to either Pipeline or Account.

1.3.1 Groups

Every pipeline, tasks and account stored in Orb database has to exist in a context of a group. A group can be anything - from a small scenario for the purpose of an analysis, to a whole Snaplogic organization. It's up to you to decide during the import of the data. Contents that belong to different groups don't relate to each other.

One scenario to follow might be to build a group for each Snaplogic organization your company has access to. This approach helps to understand the difference between environments. Groups can also be used to compare the state of the platform before and after the release.

1.3.2 Pipelines, Tasks, Accounts

Pipelines, Tasks and Accounts are main building blocks of Snaplogic. In Orb, they always belong to a group. Tasks refer to Pipelines. Pipelines contain Snaps. There's no direct relation

between Pipelines and Accounts or Pipelines themselves, but through Snaps. Because Pipelines are made of Snaps, additional information such as statistics are collected.

1.3.3 Snaps

Snaps are always contained within Pipeline. They use Accounts and Pipelines through references. They also refer to each other with `GOES_TO` relation. Snaps can be of various types and belong to snap packs. When Orb reads Pipeline definition and goes through a list of Snaps, it takes `class_id` property and builds a set of labels out of it. Let's take a look at the example of `Mapper` snap. This Snaps value of `class_id` property is `com-snaplogic-snaps-transform-datatransform`. Orb always ignores first three words and turns what's left into Snap types. Following this logic, Mapper Snap from this example will be represented in Orbs database with types: `Snap`, `Transform`, `DataTransform`. Nodes represent Snaps with most of their properties, however complex dynamic structures, such as mapping tables, routing tables or scripts are skipped.

2. SNAPLOGIC ANALYSIS WITH ORB

Orb provides user interface, in a form of web-based application, to browse and manage data. Web user interface can be accessed by three types of roles

Role	Privileges
Admin	Manage users Upload projects Browse and search Tasks, Accounts, Pipelines and Snaps Manage and execute reports
Super User	Upload projects Browse and search Tasks, Accounts, Pipelines and Snaps Manage and execute reports
User	Browse and search Tasks, Accounts, Pipelines and Snaps Manage and execute reports

2.1 Managing data

Things that you upload to Orb through its user interface are Snaplogic projects. A project first has to be exported from the *Manage* tab in Snaplogic. To upload it to Orb, navigate to *Management -> Upload*. Type the name of the group (new or existing one) and select the zip file to be uploaded and choose *Upload*.

Since a single pipeline can be exported from Snaplogic Designer, it can also be imported to Orb. To do so, simply change the scope from *Project* to *Pipeline* in the upload screen.

To delete a group from the database, navigate to *Management -> Storage* and choose *Delete* next to a group you want to remove.

2.2 Navigating through the catalogue

Catalogue is a functionality of Orb user interface, where one can navigate through Pipelines, Accounts and Tasks. In contrast to the overview approach offered by the reports section, this set of views allows to focus on a single element.

Using the top menu, navigate to Catalogue and select the desired type of object. The left pane allows to select the group and desired object. The details pane contains several sections:

2.2.1 Summary

This section shows alle the properties of a catalogue item. Some of them, under *General properties* are named in a user-friendly way, and others, under *Other properties* have their

plain database names displayed. This section allows a user to understand what properties can be found for Pipelines, Accounts and Tasks.

2.2.2 Snaps

This tab lists all Snaps for a Pipeline grouped by their snap pack. Details of each Snap can be viewed after clicking on a Snap.

2.2.3 Snap details

This window shows information gathered by Orb about a Snap.

Details -> General properties and Other properties show all attributes collected for the Snap.

Below Snap properties two sections regarding relations are displayed:

- Uses – whenever a Snap uses an Account or another Pipeline this reference is displayed here.
- Goes To – this relation indicates a Snaps connected to outputs of the detailed Snap. In other words, whenever there is a transition from this Snap to another in the Pipeline, there is a corresponding relation here.

Occurrences tab shows a global list (including all groups) of Pipeline, where this Snap occurs. There are two occurrence types:

- Identical – identically configured and named Snap
- Similar – identically configured Snap but named differently.

2.2.4 Deployment

This tab contains Pipeline properties with their values.

2.2.5 Similarity

This tab contains a list of Pipelines that Orb considers similar to the one displayed. Similarity is defined by the count of identical and similar Snaps between two Pipelines. Two Snaps are identical, when they have the same name and configuration. They are considered similar, when configuration appears the same, but there is a difference in naming.

2.2.6 References

Whenever a Pipeline uses an Account, or another Pipeline through its Snap configuration, it is listed in this tab. A separate list of relations is displayed for all Pipeline using this Pipeline or Account. These two lists are named *Relations Out* and *Relations In*.

2.3 Searching for snaps

To quickly scan the database for a specific type of Snap, navigate to Search in the top menu. You can either search by name of the Snap, or the type (extracted from the snap pack). Results are always in a context of a group. They contain the list of pipelines containing Snaps matching the criteria. Snaps details can be viewed the same similar to Catalogue.

2.4 Working with reports

Reports allow user to build custom scenarios into the application. Report definitions are based on Cypher query language. Cypher queries are executed directly on the underlying Neo4j database instance.

Reports are organized in groups. To create or remove a group, navigate to *Management -> Report* groups.

To create or modify a report, navigate to *Management -> Reports*. When creating a report, two fields require an explanation:

- Query – Cypher query to be executed directly on the underlying database. Query has to return a table structure. Every field defined in a query will result in a report result column in Orb, when the report is executed.
- Parameters – coma separated parameters used in the query. Every parameter will result in an input field to be filled in before the report is executed.

To execute a report, navigate to *Reports* in the top menu bar. All reports defined in the *Management* section of the application are visible here. An executed report can be downloaded as CSV or XSL file.

3. ADVANCED TOPICS

3.1 Accessing Neo4j database

Orb uses Neo4j as its underlying database to store Snaplogic components. To work on new reports, or go beyond what the user interface offers, accessing Neo4j instance is the way to go.

Orb installation offers a separate endpoint to access Neo4j web console. Access to this console should be shared wisely, as there is no control over permissions a user can be assigned with. To gain this functionality, please refer to Neo4j licensing page.

Neo4j uses Cypher as its query language. It's distant from what one may know but it's human readable. Cypher user reference guide may be found at

https://neo4j.com/docs/cypher-manual/current

3.2 Creating report queries

With knowledge of Cypher, access to Neo4j web console and understanding of Orb data model, one can create new report queries to extend existing functionality. This chapter explains basics to start with.

Cypher query language can return either node structure (with relations), or table structure. The latter is very similar to what SQL databases got us accustomed to. Orb reports work only with the table structure.

Let's start with creating a simple report query that returns all groups:

```
match(g:Group) return g.uid as GROUP_NAME
```

This example shows two required parts of a Cypher query – the match part and the return part. The match part specifies criteria for nodes and relations to search by, and the return part extracts information to display in the report.

Let's try another one:

```
match(g:Group{uid:'DEV'})-[:CONTAINS]->(c:Pipeline) return c.uid as
PIPELINE_NAME
```

The result would be a list of pipelines that belong to a 'DEV' group. We can see a relation between a group and a pipeline. We can also see a predicate within the match part to narrow down the result to one group only.

Something more complex:

```
match (g:Group)-[:CONTAINS]->(a:Account)<-[:USES]-(n)
with
      g.uid as g_id,
      a.name as a_id,
      count(n) as u_ct
where
      a_id is not null
return
      g_id as Group_Id,
```

```
        a_id as Account,
        u_ct as Use_Count
    order by
          Use_Count desc
```

This query returns most used accounts across the database. Pay attention to the match section first. What we are looking for is 'n'. Now, right below it, the with section allows to chain transformations and subsequent match sections (here: the count of n is what we are after). We can also see a where clause, just like in a SQL. It can be used instead of putting predicates inside the match section. Right after the return section, order by can be applied to sort the result.

With the power of Cypher, complex queries can be created. Examples above are only to get the user going and progressing with Cypher in the context of report definitions in Orb.

3.3 Working with Orb API

Orb provides an API layer to allow both populating and querying the database. The API can be tested with a swagger console available at `http://{host}:{port}/swagger-ui.htm`

The API is protected by an OAuth2 authentication layer. Use details below to acquire the bearer token:

Auth URL: `http://{host}:{port}/auth/realms/orb-realm/protocol/openid-connect/auth`

Token URL: `http://{host}:{port}/auth/realms/orb-realm/protocol/openid-connect/token`

Grant type: `client_credentials`

API consists of four sections:

3.3.1 Browse

Browse endpoints allow to retrieve data either by specifying an identifier or executing a report

GET `/api/browse`

Retrieves the list of groups and externals. Externals are not supported for Snaplogic platform at the moment. Each group entry gives the requestor information about the id and the number of components (Snaplogic Pipelines/Accounts/Tasks) that belong to it.

Parameter	Type	Required	Description
No parameters			

Sample response:

```
{
    "groups": [
```

```
        {
            "id": "acme_Prod",
            "nodeId": 0,
            "componentCount": 127
        },
        {
            "id": "acme_Dev",
            "nodeId": 1450,
            "componentCount": 404
        },
        {
            "id": "analysis_task_1209",
            "nodeId": 3795,
            "componentCount": 132
        }
    ],
    "externals": []
}
```


GET /api/browse/group/{id}

Returns a list of components contained by a group identified by {id}. This mandatory path parameter stands for an id of a database node object.

Parameter	Type	Required	Description
id	Path	Y	Id of a group to be retrieved

Sample response:


```
{
    "id": "acme_Test",
    "nodeId": 0,
    "components": [
        {
            "id": "9b74f5fc-f37d-4f7c-9997-5bc819bb73be",
            "nodeId": 1446,
            "labels": [
                "Account",
                "Binary",
                "Binarybasicauthaccount"
            ]
        },
        {
            "id": "8fe94976-9e1e-40d5-951d-7bab501f82f4",
            "nodeId": 1416,
            "labels": [
                "Account",
                "Sqlserverdatabaseaccount",
                "Accounts",
                "Sql"
            ]
        },
```

```
        {
            "id": "/acme_Test/HR/Payroll/ROUTER.pipeline",
            "nodeId": 1158,
            "labels": [
                "Pipeline"
            ]
        },
        {
            "id": "/acme_Test/HR/Payroll/SPLITTER.pipeline",
            "nodeId": 1023,
            "labels": [
                "Pipeline"
            ]
        }
    ]
}
```


GET /api/browse/component/{id}

Returns an information about a component. {id} is required and represents the database node. The response object can either represent Pipeline, Account or Task. It contains a set of attributes, a list of labels to determine and a list of entities.

Parameter	Type	Required	Description
id	Path	Y	Id of a component to be retrieved

Sample response:


```
{
    "id": "/acme_Dev/EDI/DOC/IN_AX2012.pipeline",
    "nodeId": 5438,
    "labels": [
        "Component",
        "Pipeline"
    ],
    "attributes": {
        "instance_fqid": "819d73b6-32d0-47d8-a3ed-2e076f70f7db_67",
        "property_map.info.pipeline_doc_uri": "null",
        "statistics.route_count": "0",
        "class_id": "com-snaplogic-pipeline",
        "partition_snode_id": "60534f2420443c1c5e1a36ce",
        "snode_id": "608bf0335a93a616ae21e810",
        "link_serial": "136",
        "rid": "5438",
        "_id": "819d73b6-32d0-47d8-a3ed-2e076f70f7db",
        "statistics.mapping_count": "16",
        "property_map.info.purpose": "null"
    },
    "entities": [
        {
            "id": "31645150-dbbd-4671-aae3-67bc1ccdc8d3",
```

```
            "nodeId": 5439,
            "name": "DST AX AIF",
            "labels": [
                "Snap",
                "Flow",
                "Pipeexec"
            ]
        },
        {
            "id": "6559d184-9bf6-4e83-81cd-96b9d8b16ad6",
            "nodeId": 5440,
            "name": "Union",
            "labels": [
                "Snap",
                "Flow",
                "Union"
            ]
        }
    ]
}
```


GET /api/browse/entity/{id}

Returns details of an entity. A response object can be either a snap, if an entity belongs to a pipeline, or a configuration object, when contained by an account. Every entity, a part of its attributes, contains a list of inbound and outbound relations.

Parameter	Type	Required	Description
id	Path	Y	Id of an entity to be retrieved

Sample response:


```
{
    "id": "/acme_Test/System/acme_ftp.account",
    "nodeId": 1447,
    "name": "/acme_Test/System/acme_ftp.account",
    "labels": [
        "AccountConfiguration",
        "Binary",
        "Binarybasicauthaccount"
    ],
    "attributes": {
        "original.parent_snode_id": "5f19729c746e5163dd2f2b44",
        "original.partition_snode_id": "5e164d3e212a7b3db0224455",
        "instance_fqid": "9b74f5fc-f37d-4f7c-9997-5bc819bb73be_2",
        "original.original.asset_id": "null",
        "original.original.time_leased": "null",
        "property_map.settings.username": "ext217",
        "class_id": "com-snaplogic-snaps-binary-binarybasicauthaccount",
        "original.original.metadata.pattern": "false",
        "class_build_tag": "main6403"
```

```
    },
     "relationsIn": [
          {
              "type": "CONTAINS",
              "reference": "9b74f5fc-f37d-4f7c-9997-5bc819bb73be"
          }
     ],
     "relationsOut": []
}
```


GET /api/browse/external/{id}

Not available for Snaplogic distribution of Orb.

POST /api/browse/report/execute/{id}

Executes a report by its {id} and returns the result. Accepts several parameters to control the output of the report.

Parameter	Type	Required	Description
id	Path	Y	Report identifier
parameters	Body	N	Json object containing key/value parameters to filter the result of the report.
orderBy	Body	N	Json collection of strings representing fields to order the result by.
limit	Body	N	The number of results to limit the response to.
page	Body	N	Page number to be set when the limit is smaller than the total number of records returned.

Sample request:


```
{
  "limit": 0,
  "page": 0,
  "parameters": {}
}
```


Sample response:


```
{
     "reportName": "Complexity",
     "parameters": {},
     "orderBy": null,
     "limit": 0,
     "page": 0,
```

```
    "count": 393,
    "fields": [
        "Group",
        "Pipeline",
        "Complexity_Score",
        "Snaps",
        "Flow",
        "Transform",
        "Mappings",
        "Routes"
    ],
    "results": [
        [
            "acme_Test",
            "/amce_Test/HR/WD/HR.pipeline",
            18,
            103,
            27,
            63,
            372,
            32
        ],
        [
            "acme_Test",
            "/acme_Test/HR/WD/HW2.pipeline",
            18,
            102,
            27,
            62,
            367,
            32
        ]
    ]
}
```


POST `/api/browse/report/export/{id}`

Returns the report execution result in a coma separated format. Allows to filter the same way the execution does. Doesn't allow pagination and sorting the result.

Parameter	Type	Required	Description
id	Path	Y	Report identifier
parameters	Body	N	Json object containing key/value parameters to filter the result of the report.

Sample request:

```
{
  "parameters": {}
}
```

Sample response:

```
Group,Pipeline,Complexity_Score,Snaps,Flow,Transform,Mappings,Routes
acme_Test,/acme_Test/HR/WD/HR.pipeline,18,103,27,63,372,32
acme_Test,/acme_Test/HR/WD/WD2.pipeline,18,102,27,62,367,32
```

POST `/api/browse/report/excel/{reportName}`

Same as two endpoints above, but the result is an excel file.

3.3.2 Collect

Collect endpoints populate Orb database with data.

POST `/api/collect/{vendor}/{product}`

Creates a node structure in Orb database based on the definition send in the request body. The content of a pipeline/task/account is sent as a file in a form/multipart request along with necessary parameters

Parameter	Type	Required	Description
vendor	Path	Y	Vendor name. Use “snaplogic”.
product	Path	Y	Product name. Use “iip”.
group	Query	Y	Name of a group
username	Query	Y	Name of the user to indicate who performed the import. This value is not validated and serves the audit function only.
Content-Type	Header	Y	form/multipart

Sample request: a file content as a request form part.

POST `/api/collect/{vendor}/{product}/metadata`

Creates a node structure in Orb database that extends a pipeline/task/account. A request body is a set of key/value pairs to be attached to a database sub-node. This data cannot be viewed through Orb UI.

Parameter	Type	Required	Description
vendor	Path	Y	Vendor name. Use “snaplogic”.
product	Path	Y	Product name. Use “iip”.
group	Query	Y	Name of a
username	Query	Y	Name of the user to indicate who performed the import. This value is not validated and serves the audit function only.
component	Query	Y	Uid of a pipeline/account/task to extend with metadata information

Sample request:

```
[
    {
        "key": "myNewParam",
        "value": "myNewValue"
    }
]
```

3.3.3 Group

Group endpoint allows requestor to register a new group

POST `/api/group`

Creates a group. Importing data to Orb using API does not create a non-existing group by its own. This operation has to be executed first. It is idempotent, so doesn’t fail when executed for an existing group.

Parameter	Type	Required	Description
group	Query	Y	Name of a group to be created
vendor	Query	Y	Vendor name. Use “Snaplogic”.
product	Query	Y	Product name. Use “IIP”.
username	Query	Y	Name of the user to indicate who performed the import. This value is not validated and serves the audit function only.

3.3.4 Inventory

Inventory endpoints expose management of all Orb data, that is not a graph representation of Snaplogic definitions.

GET `/api/inventory/audit`

Retrieves a collection of audit entries. Each entry represents an operation performed on either a pipeline, account, or a task.

Parameter	Type	Required	Description
No parameters			

Sample response:


```
{
    "entries": [
        {
            "entryId": 1,
            "username": "abuksztaler@sii.pl",
            "dttm": "2021-05-12T11:41:12.044654",
            "group": "acme_Dev",
            "component": "5d9ce4d62c06b765a59b1771.slp",
            "action": "ADD",
            "entryType": null,
            "attributes": null
        },
        {
            "entryId": 2,
            "username": "abuksztaler@sii.pl",
            "dttm": "2021-05-12T11:41:12.015754",
            "group": "acme_Dev",
            "component": "5e2aae4a3aa2eb1a30fcfe71.sla",
            "action": "ADD",
            "entryType": null,
            "attributes": null
        }
    ]
}
```


POST `/api/inventory/audit`

Creates new audit entry.

Parameter	Type	Required	Description

No parameters

Sample request

```
{
   "username": "abuksztaler@sii.pl",
   "group": "acme_Dev",
   "component": "5e2aae4a3aa2eb1a30fcfe71.sla",
   "action": "ADD",
}
```

GET /api/inventory/audit/{filter}

Retrieves a collection of audit entries based on a {filter}. The filter is a phrase to search for in username and audit entry attributes.

Parameter	Type	Required	Description
Filter	Path	Y	Phrase to search for in audit entry attributes.

Sample response:

```
{
    "entries": [
        {
            "entryId": 1,
            "username": "abuksztaler@sii.pl",
            "dttm": "2021-05-12T11:41:12.044654",
            "group": "acme_Dev",
            "component": "5d9ce4d62c06b765a59b1771.slp",
            "action": "ADD",
            "entryType": null,
            "attributes": null
        },
        {
            "entryId": 2,
            "username": "abuksztaler@sii.pl",
            "dttm": "2021-05-12T11:41:12.015754",
            "group": "acme_Dev",
            "component": "5e2aae4a3aa2eb1a30fcfe71.sla",
            "action": "ADD",
            "entryType": null,
            "attributes": null
        }
    ]
}
```

PUT /api/inventory/audit/{id}

Modifies an audit entry identified by {id}.

Parameter	Type	Required	Description
Id	Path	Y	Audit entry unique identifier

DELETE `/api/inventory/audit/{id}`

Deletes audit entry identified by {id}

Parameter	Type	Required	Description
Id	Path	Y	Audit entry unique identifier

GET `/api/inventory/reportgroups`

Retrieves a collection of report groups.

Parameter	Type	Required	Description
No parameters			

Sample response:


```
{
    "reportGroups": [
        {
            "id": 1,
            "name": "General",
            "description": "General report group ",
            "editable": false,
            "createdDate": "2021-05-10T08:21:43.722076",
            "modifiedDate": "2021-05-10T08:21:43.722076",
            "createdBy": "SYSTEM",
            "modifiedBy": "SYSTEM",
            "fullName": "General / General report group "
        },
        {
            "id": 2,
            "name": "Snaplogic",
            "description": "SnapLogic generic reports",
            "editable": true,
            "createdDate": "2021-05-10T08:26:52.727011",
            "modifiedDate": "2021-05-10T08:26:52.727011",
            "createdBy": null,
            "modifiedBy": null,
            "fullName": "Snaplogic / SnapLogic generic reports"
        }
    ]
```

```
}
```

POST `/api/inventory/reportgroups`

Creates a new report group.

Parameter	Type	Required	Description
No parameters			

Sample request:

```
{
    "name": "My group",
    "description": "My report group"
}
```

DELETE `/api/inventory/reportgroups/{id}`

Removes a report group identified by {id}

Parameter	Type	Required	Description
Id	Path	Y	Report group unique identifier

PUT `/api/inventory/reportgroups/{id}`

Modifies a report group identified by {id}

Parameter	Type	Required	Description
Id	Path	Y	Report group unique identifier

Sample request:

```
{
    "name": "My changed group",
    "description": "My changed report group"
}
```

GET /api/inventory/reports

Retrieves a collection of reports

Parameter	Type	Required	Description
No parameters			

Sample response:


```
{
    "reports": [
        {
            "id": 52,
            "name": "Complexity",
            "description": "Complexity report for SnapLogic pipelines",
            "query": "match (g:Group)-[:CONTAINS]->(p:Pipeline)-
[:CONTAINS]->(sn:Snap) with g,p,count(sn) as s optional match (p)-
[:CONTAINS]->(sn:Snap:Flow) with g,p,s,count(sn) as s1 optional match (p)-
[:CONTAINS]->(sn:Snap:Transform) with g,p,s,s1,count(sn) as s2,
toInteger(coalesce(p.`statistics.mapping_count`, '0')) as mc,
toInteger(coalesce(p.`statistics.route_count`, '0')) as rc return   g.uid
as Group,   p.uid as Pipeline,   ((s*100)+(mc*20)+(rc*20))/1000 as
Complexity_Score,   s as Snaps,   s1 as Flow,   s2 as Transform,   mc as
Mappings,   rc as Routes   order by Snaps desc",
            "parameters": "",
            "group": {
                "id": 2,
                "name": "Snaplogic",
                "description": "SnapLogic generic reports",
                "editable": true,
                "createdDate": "2021-05-10T08:26:52.727011",
                "modifiedDate": "2021-05-10T08:26:52.727011",
                "createdBy": null,
                "modifiedBy": null,
                "fullName": "Snaplogic / SnapLogic generic reports"
            },
            "createdDate": "2021-05-10T08:28:15.077095",
            "modifiedDate": "2021-05-10T08:35:58.671764",
            "createdBy": null,
            "modifiedBy": null
        }
    ]
}
```


POST /api/inventory/reports

Creates a new report definition.

Parameter	Type	Required	Description
No parameters			

Sample request:

```
{
    "name": "Complexity",
    "description": "Report for SnapLogic pipelines",
    "query": "match (n:Entity) return n.uid",
    "parameters": "",
    "groupName": "MyGroup"
}
```

DELETE `/api/inventory/reports/{id}`

Removes a report definition identified by {id}

Parameter	Type	Required	Description
Id	Path	Y	Report definition unique identifier

PUT `/api/inventory/reports/{id}`

Updates a report definition identified by {id}

Parameter	Type	Required	Description
Id	Path	Y	Report definition unique identifier

Sample request:

```
{
    "description": "Report for SnapLogic pipelines",
    "query": "match (n:Entity) return n.uid",
    "parameters": "",
    "groupName": "MyGroup"
}
```

GET `/api/inventory/reports/{name}`

Retrieves a report definition by {name}

Parameter	Type	Required	Description
Name	Path	Y	Report definition name

Sample response:

```
{
```

```
    "id": 54,
    "name": "Most used pipelines",
    "description": "",
    "query": "match (g:Group)-[:CONTAINS]->(p:Pipeline)<-[:USES]-(n) with g.uid as g_id, p.name as p_id, count(n) as u_ct return  g_id as Group_Id, p_id as Pipeline,  u_ct as Use_Count  order by Use_Count desc",
    "parameters": "",
    "groupName": "Snaplogic",
    "createdDate": "2021-05-10T08:36:48.345572",
    "modifiedDate": "2021-05-10T08:45:37.464541",
    "createdBy": null,
    "modifiedBy": null
}
```


GET /api/inventory/users

Returns a list users defined in Orb.

Parameter	Type	Required	Description
No parameters			

Sample response:


```
{
    "users": [
        {
            "id": "fe358034-a718-4e42-b10a-df2ead0769cf",
            "username": "adam_b",
            "firstName": "Adam",
            "lastName": "Buksztaler",
            "enabled": false,
            "emailVerified": true,
            "email": "abuksztaler@sii.com",
            "roles": [
                "orb-user",
                "orb-superuser"
            ],
            "createdDate": "2020-11-11T08:19:49.69"
        },
        {
            "id": "58bf4653-fcd1-4cd0-a23a-046eb3365e80",
            "username": "default_admin",
            "firstName": "Default",
            "lastName": "Admin",
            "enabled": true,
            "emailVerified": true,
            "email": "orb@sii.pl",
            "roles": [
                "orb-admin"
            ],
            "createdDate": "2020-11-08T20:08:56.094"
        }
    ]
}
```

GET `/api/inventory/users/{username}`

Returns a user identified by {username}

Parameter	Type	Required	Description
Username	Path	Y	Username

Sample response:


```
{
    "userId": "58bf4653-fcd1-4cd0-a23a-046eb3365e80",
    "username": "default_admin",
    "firstName": "Default",
    "lastName": "Admin",
    "email": "orb@sii.pl",
    "enabled": true,
    "emailVerified": true,
    "roles": [
        "orb-admin"
    ],
    "createdDate": "2020-11-08T20:08:56.094"
}
```