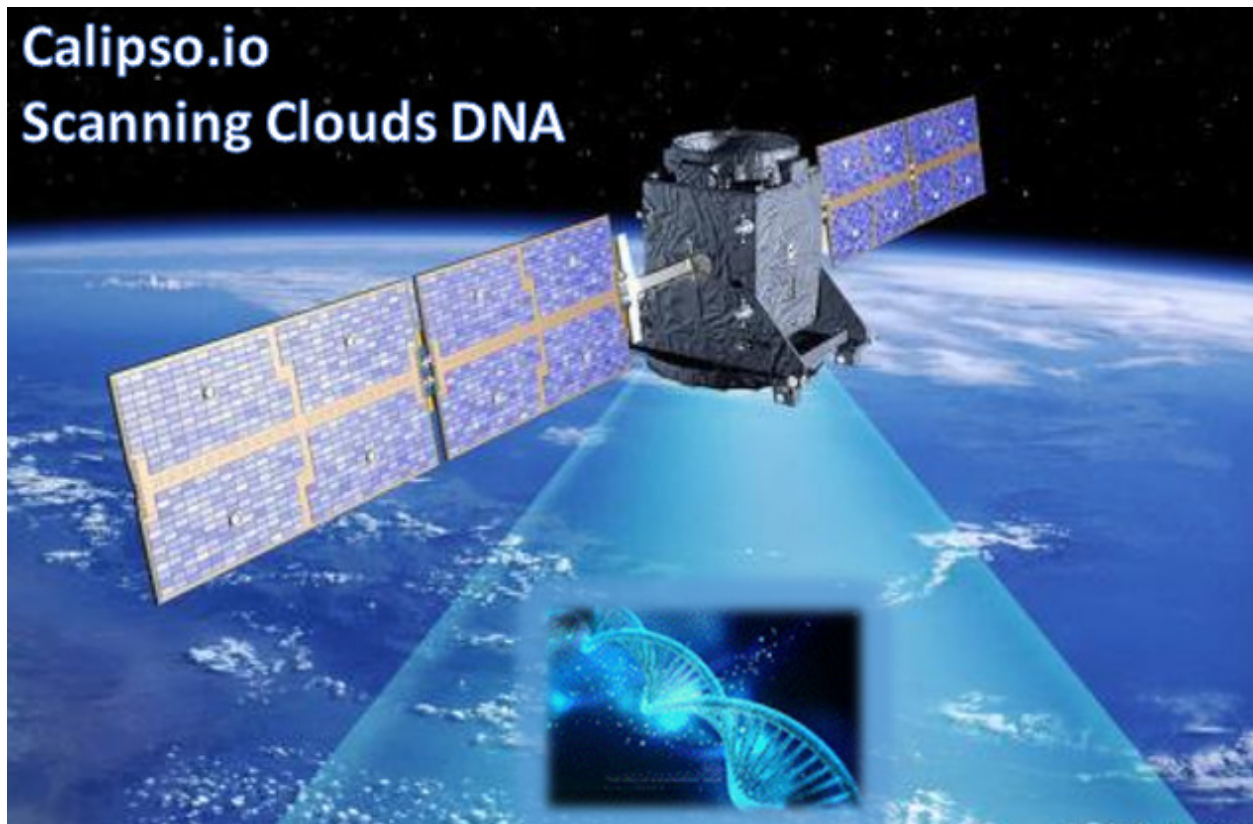# Calipso.io
Objects Model

Project "Calipso" tries to illuminate complex virtual networking with real time operational state visibility for large and highly distributed Virtual Infrastructure Management (VIM).

Calipso provides visible insights using smart discovery and virtual topological representation in graphs, with monitoring per object in the graph inventory to reduce error vectors and troubleshooting, maintenance cycles for VIM operators and administrators.

Calipso model, described in this document, was built for multi-environment and many VIM variances, the model was tested successfully (as of Aug 27th) against 60 different VIM variances (Distributions, Versions, Networking Drivers and Types).

Table of Contents

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

        A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

1 Environments config

Environment is defined as a certain type of Virtual Infrastructure facility the runs under a single unified Management (like an OpenStack facility).

Everything in Calipso application rely on environments config, this is maintained in the **"environments_config"** collection in the mongo Calipso DB.

Environment configs are pushed down to Calipso DB either through UI or API (and only in OPNFV case Calipso provides an automated program to build all needed environments_config parameters for an 'Apex' distribution automatically).

When scanning and discovering items Calipso uses this configuration document for successful scanning results, here is an example of an environment config document:

```
{
    "name": "DEMO-ENVIRONMENT-SCHEME",
    "enable_monitoring": true,
    "last_scanned": "filled-by-scanning",
    "app_path": "/home/scan/calipso_prod/app",
    "type": "environment",
    "distribution": "Mirantis",
    "distribution_version": "8.0",
    "mechanism_drivers": ["OVS"],
    "type_drivers": "vxlan"
    "operational": "stopped",
    "listen": true,
    "scanned": false,
    "configuration": [
        {
            "name": "OpenStack",
            "port":"5000",
            "user": "adminuser",
            "pwd": "dummy_pwd",
            "host": "10.0.0.1",
            "admin_token": "dummy_token"
        },
        {
            "name": "mysql",
            "pwd": "dummy_pwd",
            "host": "10.0.0.1",
            "port": "3307",
            "user": "mysqluser"
        },
        {
            "name": "CLI",
            "user": "sshuser",
            "host": "10.0.0.1",
            "pwd": "dummy_pwd"
        },
        {
```

```
      "name": "AMQP",
      "pwd": "dummy_pwd",
      "host": "10.0.0.1",
      "port": "5673",
      "user": "rabbitmquser"
    },
    {
      "name": "Monitoring",
      "ssh_user": "root",
      "server_ip": "10.0.0.1",
      "ssh_password": "dummy_pwd",
      "rabbitmq_pass": "dummy_pwd",
      "rabbitmq_user": "sensu",
      "rabbitmq_port": "5671",
      "provision": "None",
      "env_type": "production",
      "ssh_port": "20022",
      "config_folder": "/local_dir/sensu_config",
      "server_name": "sensu_server",
      "type": "Sensu",
      "api_port": NumberInt(4567)
    },
    {
      "name": "ACI",
      "user": "admin",
      "host": "10.1.1.104",
      "pwd": "dummy_pwd"
    }
  ],
  "user": "wNLeBJxNDyw8G7Ssg",
  "auth": {
    "view-env": [
      "wNLeBJxNDyw8G7Ssg"
    ],
    "edit-env": [
      "wNLeBJxNDyw8G7Ssg"
    ]
  },
}
```

Here is a brief explanation of the purpose of major keys in this environment configuration doc:

Distribution: captures type of VIM, used for scanning of objects, links and cliques.

Distribution_version: captures version of VIM distribution, used for scanning of objects, links and cliques.

Mechanism_driver: captures virtual switch type used by the VIM, used for scanning of objects, links and cliques.

Type_driver: captures virtual switch tunneling type used by the switch, used for scanning of objects, links and cliques.

Listen: defines whether or not to use Calipso listener against the VIM BUS for updating inventory in real-time from VIM events.

Scanned: defines whether or not Calipso ran a full and a successful scan against this environment.

Last_scanned: end time of last scan.

Operational: defines whether or not VIM environment endpoints are up and running.

Enable_monitoring: defines whether or not Calipso should deploy monitoring of the inventory objects running inside all environment hosts.

Configuration-OpenStack: defines credentials for OpenStack API endpoints access.

Configuration-mysql: defines credentials for OpenStack DB access.

Configuration-CLI: defines credentials for servers CLI access.

Configuration-AMQP: defines credentials for OpenStack BUS access.

Configuration-Monitoring: defines credentials and setup for Calipso sensu server (see monitoring-guide for details).

Configuration-ACI: defines credentials for ACI switched management API, if exists.

User and auth: used for UI authorizations to view and edit this environment.

App-path: defines the root directory of the scanning application.

2 Inventory objects

Calipso's success in scanning, discovering and analyzing many (60 as of 27th Aug 2017) variances of virtual infrastructures lies with its objects model and relationship definitions (model was tested even against a vSphere VMware environment).
Those objects are the real-time processes and systems that are built by workers and agents on the virtual infrastructure servers.
All Calipso objects are maintained in the **"inventory"** collection.
Here are the major objects defined in Calipso inventory in order to capture the real-time state of networking:

2.1 Host

It's the physical server that runs all virtual objects, typically a hypervisor or a containers hosting machine.

It's typically a bare-metal server, in some cases it might be virtual (running "nesting" VMs as second virtualization layer inside it).

2.2 physical NIC (pNIC)

It's the physical Ethernet Network Interface Card attached to the Host, typically several of those are available on a host, in some cases few of those are grouped (bundled) together into etherchannel bond interfaces.

For capturing data from real infrastructure devices Calipso created 2 types of pNICs: host_pnic (pNICs on the servers) and switch_pnic (pNICs on the physical switches). Calipso currently discovers host to switch physical connections only in some types of switches (Cisco ACI as of Aug 27th 2017).

2.3 Bond

It's a logical Network Interface using etherchannel standard protocols to form a group of pNICs providing enhanced throughput for communications to/from the host.

Calipso currently maintains bond details inside a host_pnic object.

2.4 Instance

It's the virtual server created for running a certain application or function. Typically it's a Virtual Machine, sometimes it's a Container.

2.5 virtual Service (vService)

It's a process/system that provides some type of networking service to instances running on networks, some might be deployed as namespaces and some might deploy as VM or Container, for example: DHCP server, Router, Firewall, Load-Balancer, VPN service and others. Calipso categorized vServices accordingly.

2.6 Network

It's an abstracted object, illustrating and representing all the components (see below) that builds and provides communication services for several instances and vServices.

2.7 virtual NIC (vNIC)

There are 2 types - instance vNIC and vService vNIC:
- Instance vNIC: It's the virtual Network Interface Card attached to the Instance and used by it for communications from/to that instance.
- vService vNIC: It's the virtual Network Interface Card attached to the vService used by it for communications from/to that vService.

2.8 Port

It's an abstracted object representing the attachment point for an instance or a vService into the network, in reality it's fulfilled by deployment of vNICs on hosts.

2.9 virtual Connector (vConnector)

It's a process/system that provides layer 2 isolation for a specific network inside the host (isolating traffic from other networks). Examples: Linux Bridge, Bridge-group, port-group etc.

2.10 virtual Edge (vEdge)

It's a process/system that provides switching and routing services for instances and/or vServices running on a specific host. It function as an edge device between virtual components running on that host and the pNICs on that host, making sure traffic is maintained and still isolated across different networks.

Examples: Open Virtual Switch, Midonet, VPP.

2.11 Overlay-Tunnel-Endpoint (OTEP)

It's an abstracted object representing the end-point on the host that runs a certain tunneling technology to provide isolation across networks and hosts for packets leaving and entering the pNICs of a specific host. Examples: VXLAN tunnels endpoints, GRE tunnels endpoints etc.

2.12 Network_segment

It's the specific segment used inside the "overlay tunnel" to represent traffic from a specific network, this depends on the specific type (encapsulation) of the OTEP.

Calipso currently maintains segments details inside a network object.

2.13 Network_Agent

It's a controlling software running on the hosts for orchestrating the lifecycle of the above virtual components. Examples: DHCP agent, L3 agent, OVS agent, Metadata agent etc.
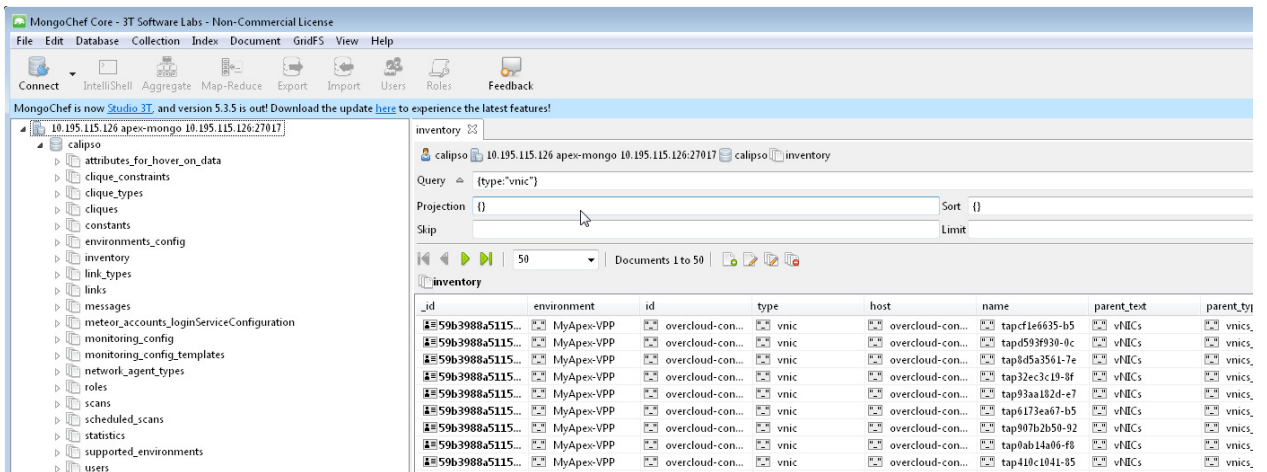
2.14 Looking up Calipso objects details

As explained in more details in Calipso admin-guide, the underlying database used is mongoDB. All major objects discovered by Calipso scanning module are maintained in the "inventory" collection and those document includes detailed attributes captured from the infrastructure about those objects, here are the main objects quarries to use for grabbing each of the above object types from Calipso's inventory:

{type:"vnic"}
{type:"vservice"}
{type:"instance"}
{type:"host_pnic"}
{type:"switch_pnic"}
{type:"vconnector"}
{type:"vedge"}
{type:"network"}
{type:"network_agent"}
{type:"otep"}
{type:"host"}
{type:"port"}

All Calipso modules (visualization, monitoring and analysis) rely on those objects as baseline inventory items for any further computation.
Here is an example of a query made using mongo Chef Client application:

* See Calipso API-guide for details on looking up those objects through the Calipso API.

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

The following simplified UML illustrates the way Calipso objects relationships are maintained in a VIM of type OpenStack:
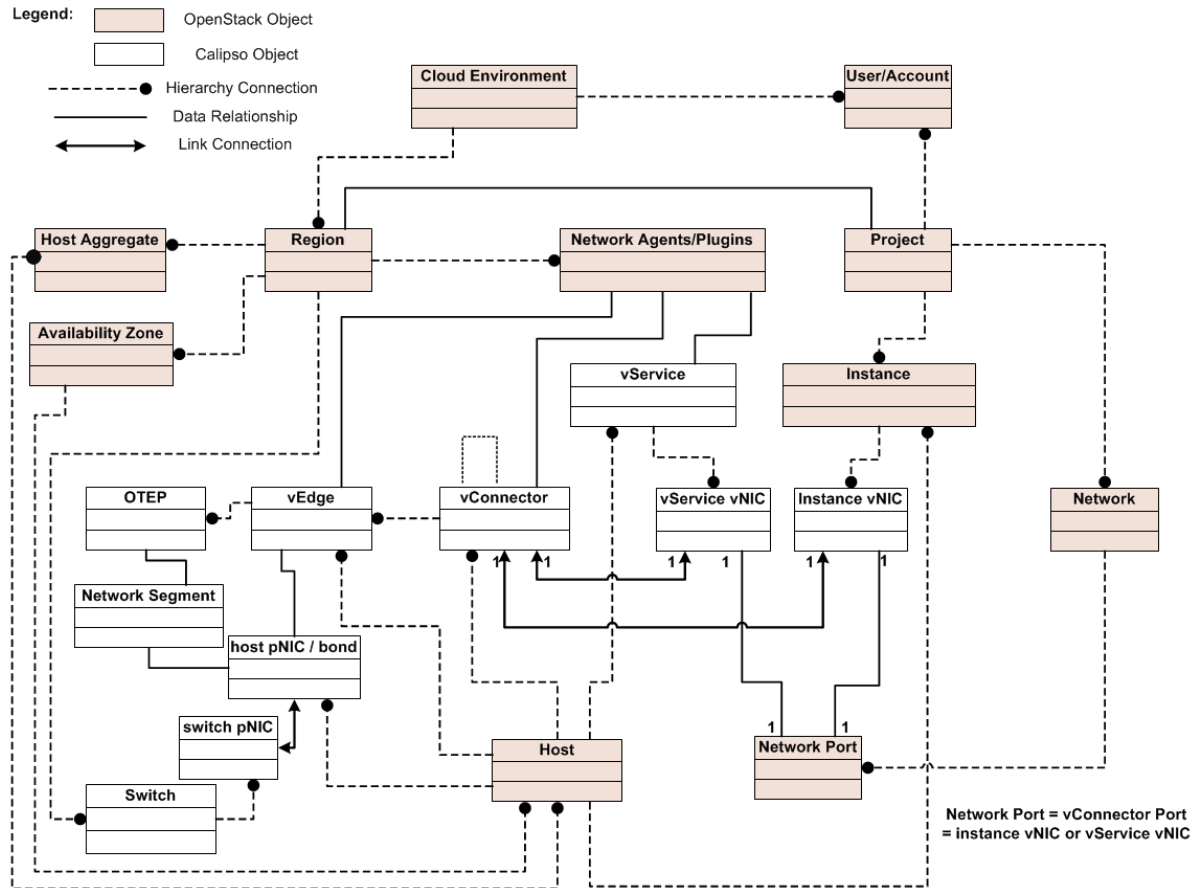
3 Link Objects

Calipso analyzes all objects in its inventory for relationships, finding in real-time, which object is attached to which object and then creates a link object representing this relationship. This analysis finds a link that is "single hop away" - a connection from certain object to certain object that is attached to it directly.
Derived relationships (A to B and B to C = A to C) is maintained as 'cliques'.
Links objects are maintained in the **"links"** collection.

3.1 Link types

Based on the specific VIM distribution, distribution version, mechanism driver and type driver a set of links are discovered automatically by Calipso scanning module.

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

Each link type is bi-directional, it means that if a connection is discovered from A to B, a connection also exists from B to A.

Here is the list of link types that might be discovered from a certain environment in the current release:

{"link_type": "instance-vnic"}
{"link_type": "vnic-vconnector"}
{"link_type": "vconnector-vedge"}
{"link_type": "vedge-host_pnic"}
{"link_type: "host_pnic-network"}
{"link_type": "vedge-otep"}
{"link_type": "otep-vconnector"}
{"link_type": "otep-host_pnic"}
{"link_type": "vconnector-host_pnic"}
{"link_type": "vnic-vedge"}
{"link_type": "vservice-vnic"}
{"link_type": "switch_pnic-host_pnic"}
{"link_type": "switch_pnic-switch_pnic"}
{"link_type": "switch_pnic-switch"}

A successful completion of scanning and discovery means that all inventory objects, link objects and clique objects (see below) are found and accurately representing real-time state of the virtual networking on the specific environment.

4 Clique objects

Cliques are lists of links. Clique represent a certain path in the virtual networking infrastructure that an administrator is interested in, this is made to allow easier searching and finding of certain points of interest ("focal point").

4.1 Clique types

Based on the specific VIM distribution, distribution version, mechanism driver and type driver variance, Calipso scanning module search for specific cliques using a model that is pre-populated in its **"clique_types"** collection, and it depends on the environment variance, here is an example of a clique_type:

```
{
    "environment" : "Apex-Euphrates",
    "link_types" : [
        "instance-vnic",
```

```
        "vnic-vconnector",
        "vconnector-vedge",
        "vedge-otep",
        "otep-host_pnic",
        "host_pnic-network"
    ],
    "name": "instance_clique_for_opnfv",
    "focal_point_type": "instance"
}
```

The above model instruct the Calipso scanner to create cliques with the above list of link types for a "focal_point" that is an "instance" type of object. We believe this is a highly customized model for analysis of dependencies for many use cases. We have included several clique types, common across variances supported in this release.

The cliques themselves are then maintained in the **"cliques**" collection.

To clarify this concept, here is an example for an implementation use case in the Calipso UI module:

When the user of the UI clicks on a certain object of type=instance, he expresses he's wish to see a graph representing the path taken by traffic from that specific instance (as the root source of traffic, on that specific network) all the way down to the host pNIC and the (abstracted) network itself.

A successful completion of scanning and discovery means that all inventory objects, link objects and clique objects (based on the environment clique types) are found and accurately representing real-time state of the virtual networking on the specific environment.

5 Supported Environments

As of Aug 27th 2017, Calipso application supports 60 different VIM environment variances and with each release the purpose of the application is to maintain support and add more variances per the VIM development cycles. The latest supported variance and the specific functions of Calipso available for that specific variance is captured in the **"supported_environments"** collection, here are two examples of that 'supported' model:

1.
```
{
    "environment" : {
        "distribution" : "Apex",
        "distribution_version" : ["Euphrates"],
        "mechanism_drivers" : "OVS",
        "type_drivers" : "vxlan"
    },
    "features" : {
        "listening" : true,
```

```
        "scanning" : true,
        "monitoring" : false
    }
}

2.
{
  "environment" : {
    "distribution" : "Mirantis",
    "distribution_version": ["6.0", "7.0", "8.0", "9.0", "9.1", "10.0"],
    "mechanism_drivers" : "OVS",
    "type_drivers" : "vxlan"
  },
  "features" : {
    "listening" : true,
    "scanning" : true,
    "monitoring" : true
  }
}
```

The examples above defines for Calipso application that:

1. For an 'Apex' environment of version 'Euphrates' using OVS and vxlan, Calipso can scan/discover all details (objects, links, cliques) but is not yet monitoring those discovered objects.

2. For a "Mirantis" environment of versions 6.0 to 10.0 using OVS and vxlan, Calipso can scan/discover all details (objects, links, cliques) and also monitor those discovered objects.

With each calipso release more "supported_environments" should be added.

6 System collections

Calipso uses other system collections to maintain its data for scanning, event handling, monitoring and for helping to operate the API and UI modules, here is the recent list of collections not covered yet in other written guides:

6.1 Attributes_for_hover_on_data

This collection maintains a list of documents describing what will be presented on the UI popup screen when the use hover-on a specific object type, it details which parameters or attributed from the object's data will be shown on the screen, making this popup fully customized.

6.2 Clique_constraints

Defines the logic on which cliques are built, currently network is the main focus of the UI (central point of connection for all cliques in the system), but this is customizable.
When building a clique graph, Calipso defaults to traversing all nodes edges (links) in the graph.
In some cases we want to limit the graph so it will not expand too much (nor forever).
For example: when we build the graph for a specific instance, we limit the graph to only take objects from the network on which this instance resides - otherwise the graph will show objects related to other instances.
The constraint validation is done by checking value V from the focal point F on the links.
For example, if an n instance has network X, we check that each link we use either has network X (attribute "network" has value X), or does not have the "network" attribute.

6.3 Connection_tests

This collection keeps requests from the UI or API to test the different adapters (API, DB, and CLI etc) and their connections to the underlying VIM, making sure dynamic and real-time data will be maintained during discovery.

6.4 Messages

Aggregates all loggings from the different systems, Source_System of logs currently defined as "OpenStack" (the VIM), "Sensu" (the Monitoring module) and "Calipso" (logs of the application itself. Messages have 6 levels of severity and can be browsed in the UI and through Calipso API.

6.5 Network_agent_types

Lists the types of networking agents supported on the VIM (per distribution and version).

6.6 Roles, Users

Basic RBAC facility to authorize calispo UI users for certain calipso functionalities on the UI.

6.7 Statistics

Built for detailed analysis and future functionalities, used today for traffic analysis (capturing samples of throughputs per session on VPP based environments).

6.8 Constants

This is an aggregated collection for many types of documents that are required mostly by the UI and basic functionality on some scanning classes ('fetchers').

6.9 Constants-env_types

Type of environments to allow for configuration on sensu monitoring framework.

6.10   Constants-log_levels

Severity levels for messages generated.

6.11   Constants-mechanism_drivers

Mechanism-drivers allowed for UI users.

6.12   Constants-type_drivers

Type-drivers allowed for UI users.

6.13   Constants-environment_monitoring_types

Currently only "Sensu" is available, might be used for other monitoring systems integrations.

6.14   Constants-link_states

Provides statuses for link objects, based on monitoring results.

6.15   Constants-environment_provision_types

The types of deployment options available for monitoring (see monitoring-guide for details).

6.16   Constants-environment_operational_status

Captures the overall (aggregated) status of a curtained environment.

6.17   Constants-link_types

Lists the connections and relationships options for objects in the inventory.

6.18   Constants-monitoring_sides

Used for monitoring auto configurations of clients and servers.

6.19   Constants-object_types

Lists the type of objects supported through scanning (inventory objects).

6.20   Constants-scans_statuses

During scans, several statuses are shown on the UI, based on the specific stage and results.

6.21   Constants-distributions

Lists the VIM distributions.

6.22   Constants-distribution_versions

Lists the VIM different versions of different distributions.

6.23   Constants-message_source_systems

The list of systems that can generate logs and messages.

6.24   Constants-object_types_for_links

Object_types used only for link popups on UI.

6.25 Constants-scan_object_types

Object_types used during scanning, see development-guide for details.

6.26 Constants-configuration_targets

Names of the configuration targets used in the configuration section of environment configs.