Craton Documentation

Release

OpenStack Foundation

Mar 29, 2017

Contents

Craton is a new project planned for OpenStack inclusion. Craton supports deploying and operating OpenStack clouds by providing scalable fleet management:

- Inventory of configurable physical devices/hosts (the fleet)
- Audit and remediation workflows against this inventory
- REST APIs, CLI, and Python client to manage

Support for workflows, CLI, and the Python client is in progress.

CHAPTER 1

Getting Started

Installation

There are several ways that you can install Craton. If you're just getting started, it is recommended that you start with a Docker install.

Docker Install

Installing using Docker

Installation

Installing necessary packages: Ubuntu

- Make sure git is installed:

```
$ sudo apt-get update
$ sudo apt-get install git -y
```

- Clone the Craton repository:

```
$ git clone https://github.com/openstack/craton.git
```

- **To install Docker, follow the instructions found here:** https://docs.docker.com/engine/installation/linux/ubuntulinux/

Installing necessary packages: Fedora/CentOS etc.

- Install a fresh Fedora/CentOS image

- Make sure we have git installed:

```
$ sudo yum update
$ sudo yum install git -y
```

- Clone the repository:

```
$ git clone https://github.com/openstack/craton.git
```

- Follow the correct Docker install guide for your operating system:

```
Fedora: https://docs.docker.com/engine/installation/linux/fedora/
CentOS: https://docs.docker.com/engine/installation/linux/centos/
```

Run the Craton Docker Image

- First, go to craton directory and build the Docker image:

```
$ sudo docker build --pull -t craton-api:latest .
```

- And finally, run the docker image:

```
$ sudo docker run -t --name craton-api -d craton-api:latest
```

Calling into Craton

- Let's get container Id:

```
$ ContainerId=$(docker ps | grep craton-api:latest | awk '{print $1}')
```

- We need the container IP, so we can run an API call against Craton running in the container:

```
$ ContainerIP=$(docker inspect --format '{{ .NetworkSettings.IPAddress }}' $
↪{ContainerId})
```

- Bootstrap credentials are generated at the top of the craton-api logs for initial authentication. You can manually copy the username, api key, and project id from the logs by running:

```
$ docker logs -f craton-api
```

  Or you can grep for them:

```
$ CRATON_PROJECT_ID=$(docker logs craton-api | grep "ProjectId:" | awk '{print $2}
↪' | tr -d '\r')
$ CRATON_USERNAME=$(docker logs craton-api | grep "Username:" | awk '{print $2}'␣
↪| tr -d '\r')
$ CRATON_API_KEY=$(docker logs craton-api | grep "APIKey:" | awk '{print $2}' |␣
↪tr -d '\r')
```

- To generate a sample data set, use the following command:

```
$ python tools/generate_fake_data.py --url http://${ContainerIP}:7780/v1 --user "
↪$CRATON_USERNAME" --project "$CRATON_PROJECT_ID" --key "$CRATON_API_KEY"
```

- Now you can run a curl command like the one below to query Craton:

```
$ curl -i "http://${ContainerIP}:7780/v1/hosts?region_id=1" -H "Content-Type:␣
↪application/json" -H "X-Auth-Token: ${CRATON_API_KEY}" -H "X-Auth-User: $
↪{CRATON_USERNAME}" -H "X-Auth-Project: ${CRATON_PROJECT_ID}"
```

Using wrapper functions

*Some wrapper functions have been included in craton/tools to quickly build, reload, populate, and query craton. * To load the wrapper functions, run the following in the craton parent directory:

```
# source tools/wrapper-functions.sh
```

- To quick start and populate craton in docker, run the following from the craton parent directory:

```
# craton-docker-start
```

- In order to interact with craton, export the bootstrap credentials by running:

```
# export eval $(craton-docker-env)
```

- Populate craton with fake data by running:

```
# craton-fake-data
```

- Run API calls against craton with the following wrappers:

Note: *Requires the installation of httpie*

craton-post v1/regions name=HKG # craton-get v1/hosts # craton-put v1/hosts/3 device_type=container # craton-put v1/hosts/3/variables foo=47 bar:=’[”a”, “b”, “c”]’ # craton-delete v1/hosts/4

Command Cheat-Sheet

- Get the Craton logs:

```
$ docker logs -f craton-api
```

- Open mysql in the Craton container:

```
$ docker exec -it craton-api mysql -ucraton -pcraton craton
```

- Get a bash shell from the Craton container:

```
$ docker exec -it craton-api bash # for a bash shell, etc
```

Basic Install

(Optional) install virtualenv if desired:

```
$ mkvirtualenv craton
$ pip install -r /craton/requirements.txt
$ python setup.py install
```

Setup Developer Environment

Installing and Setting up a Development Environment

Installation

Note: *This is a Python3 project.*

Note: *This project requires MySQL 5.7, until a stable release of MariaDB with JSON function support is available*

Ubuntu 16.04 (Xenial)

- Install a fresh Ubuntu image
- Make sure we have git installed:

```
# apt-get update
# apt-get install git -y
```

- Clone the repository:

```
# git clone https://github.com/openstack/craton.git
```

- Install the prerequisite packages:

```
# apt-get install python3.5 python3.5-dev
# apt-get install python3-pip python3-setuptools
# python3 -m pip install --upgrade pip setuptools
```

- Goto craton directory and install the following:

```
# python3 -m pip install -r requirements.txt
# python3 -m pip install .
```

- Install mysql-server and make sure mysql is running:

```
# apt-get install mysql-server-5.7 mysql-client-5.7
# systemctl enable mysql
# systemctl start mysql
```

- Ensure you have python3-mysqldb installed:

```
# apt-get install python3-mysqldb
```

CentOS 7

- Install a fresh CentOS 7 image
- Make sure we have git installed:

```
# yum update
# yum install git -y
```

- Clone the repository:

```
# git clone https://github.com/openstack/craton.git
```

- Install the prerequisite packages:

```
# yum install python34-devel python34-pip python34-setuptools gcc
# python3 -m pip install --upgrade pip setuptools
```

- Goto craton directory and install the following:

```
# python3 -m pip install -r requirements.txt
# python3 -m pip install .
```

- Install mysql-server community release from MySQL Community Page:

```
# wget https://dev.mysql.com/get/mysql57-community-release-el7-9.noarch.rpm
# rpm -ivh mysql57-community-release-el7-9.noarch.rpm
# yum install mysql-server
# systemctl enable mysqld
# systemctl start mysqld
```

- Ensure you have MySQL-python installed:

```
# yum install MySQL-python
```

- Setup Database User and secure installation:

```
# grep 'temporary password' /var/log/mysqld.log
# mysql_secure_installation
```

Fedora 25

- Install a fresh Fedora 25 image
- Make sure we have git installed:

```
# dnf update
# dnf install git -y
```

- Clone the repository:

```
# git clone https://github.com/openstack/craton.git
```

- Install the prerequisite packages:

```
# dnf install python3-devel python3-pip python3-setuptools gcc redhat-rpm-config
# python3 -m pip install --upgrade pip setuptools
```

- Goto craton directory and install the following:

```
# python3 -m pip install -r requirements.txt
# python3 -m pip install .
```

- Install mysql-server and make sure mysql is running:

```
# dnf install mysql-server
# systemctl enable mysqld
# systemctl start mysqld
```

- Ensure you have python3-mysql installed:

```
# dnf install python3-mysql
```

Database Setup

- Connect to database server as root user:

```
# mysql -u root -p
```

- Create user craton:

```
# CREATE USER 'craton'@'localhost' IDENTIFIED BY 'craton';
```

- Grant proper access to the craton user and flush privileges:

```
# GRANT ALL PRIVILEGES ON craton.* TO 'craton'@'localhost'
identified by 'craton';
# FLUSH PRIVILEGES;
```

- You can verify that the user was added by calling:

```
# select host, user, password from mysql.user;
```

- Create the Craton database:

```
# create database craton CHARACTER SET='utf8';
```

- Logout from the database server:

```
# exit
```

Create etc/craton-api-conf.dev

- Copy the sample config in the etc directory to make a development config file.

  # cp craton-api-conf.sample craton-api-conf.dev

- Make api_paste_config use a fully qualified path (not relative). This will be specific for your machine

Note: Make sure you have the proper path for craton-api-conf.dev

api_paste_config=/home/cratonuser/craton/etc/craton-api-paste.ini

- Add the following line to the [database] section:

  # connection = mysql+pymysql://craton:craton@localhost/craton

- Update the host in the [api] section to match your IP:

  # host = xxx.xxx.xxx.xxx

Run dbsync

- Make sure to run dbsync to get the db tables created:

```
# craton-dbsync --config-file=etc/craton-api-conf.dev version
# craton-dbsync --config-file=etc/craton-api-conf.dev upgrade
```

- Make sure to run dbsync bootstrap to create initial project and root user:: # craton-dbsync –config-file=etc/craton-api-conf.dev bootstrap

  Note: The above command outputs user, project-id and API key to use with python-cratonclient to interact with craton server.

Start the API Service

- To start the API service, run the following command:

```
# craton-api --config-file=etc/craton-api-conf.dev
```

- Some examples of API calls are as below:

Create a Region

- In order to create the region, export the IP address you set in /etc/craton-api-conf.dev:

```
# export MY_IP=xxx.xxx.xxx.xxx
```

- Next create a cloud to which the region is associated to:

```
# curl -i "http://${MY_IP}:7780/v1/clouds" \
       -d '{"name": "Cloud_Sample"}' \
       -H "Content-Type: application/json" \
       -H "X-Auth-Token: demo" \
       -H "X-Auth-User: demo" \
       -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

- To create region, execute the following command:

```
# curl -i "http://${MY_IP}:7780/v1/regions" \
       -d '{"name": "DFW", "cloud_id": 1}' \
       -H "Content-Type: application/json" \
       -H "X-Auth-Token: demo" \
```

```
       -H "X-Auth-User: demo" \
       -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Get created Region

- To get the created region, execute the following command:

```
# curl -i "http://${MY_IP}:7780/v1/regions" \
       -H "Content-Type: application/json" \
       -H "X-Auth-Token: demo" \
       -H "X-Auth-User: demo" \
       -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Get all hosts for Region 1

- To get all hosts for region 1, execute the following command:

```
# curl -i "http://${MY_IP}:7780/v1/hosts?region_id=1" \
       -H "Content-Type: application/json" \
       -H "X-Auth-Token: demo" \
       -H "X-Auth-User: demo" \
       -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Get a particular host

- To get a particular host, execute the following command:

```
# curl -i "http://${MY_IP}:7780/v1/hosts/33" \
       -H "Content-Type: application/json" \
       -H "X-Auth-Token: demo" \
       -H "X-Auth-User: demo" \
       -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Using wrapper functions

Some wrapper functions have been included in craton/tools to quickly build, reload, populate, and query craton.

- To load the wrapper functions, run the following in the craton parent directory:

```
# source tools/wrapper-functions.sh
```

- To start craton directly, run the following from the craton parent directory:

```
# craton-direct-start
```

- The following environment variables must be exported for working with the craton API server:

```
* CRATON_URL
* OS_PROJECT_ID
* OS_USERNAME
* OS_PASSWORD
```

- You can search the logs for these values or run:

```
# export eval $(craton-direct-env)
```

- Populate craton with fake data by running:

```
# craton-fake-data
```

- Run API calls against craton with the following wrappers:

Note: *Requires the installation of httpie*

craton-post v1/regions name=HKG # craton-get v1/hosts # craton-put v1/hosts/3 device_type=container # craton-put v1/hosts/3/variables foo=47 bar:='["a", "b", "c"]' # craton-delete v1/hosts/4

Running Tests

- To run unit tests, execute the following command:

```
# tox
```

- To run functional tests, execute the following command:

```
# tox -e functional
```

Using Keystone for Identity

By default, Craton uses it's own local authentication mechanism. It also supports using Keystone for identity and authentication.

Before you can proceed, you need to first create a user for Craton, e.g.,

```
openstack user create --project service \
                      --description 'Craton Service User' \
                      --password-prompt \
                      --enable \
                      craton
```

And then you must add the admin role to it:

```
openstack role add --user craton \
                   --project service \
                   admin
```

And then you must create the service and endpoints:

```
openstack service create --description 'Craton Fleet Management' \
                         --name 'craton' \
                         --enable \
                         fleet_management
for endpoint_type in "admin internal public" ; do
  openstack endpoint create \
    fleet_management $endpoint_type http://<ip>:<port>/v1 \
```

```
    --region RegionOne
done
```

Then you need to select the `keystone-auth` pipeline and configure the usual Keystone auth token middleware options in the Craton API config file, e.g.,

```
[api]
# ...
paste_pipeline = keystone-auth

[keystone_authtoken]
auth_uri = https://<keystone-ip>:5000
auth_url = https://<keystone-ip>:35357/v3
project_name = service
username = craton
password = aVery_Secure&Complex+Password
project_domain_id = default
user_domain_id = default
auth_type = password
```

You may need to either not use `https` in your URL or set `insecure = True` to avoid SSL errors.

Now with an appropriate identity in Keystone, one can use either the python craton client or another client that can retrieve tokens from Keystone. For example, if you use the openstack client to grab a token, you can use curl to talk to Craton:

```
export AUTH_TOKEN="$(openstack token issue -c id -f value)"
curl -i \
     -H"X-Auth-Token: $AUTH_TOKEN" \
     http://<ip>:<port>/v1/hosts?region_id=1
```

Craton service command-line client

Contents

craton usage

Subcommands:

craton usage Show usages of craton client.

craton project-create Create a new project.

craton project-delete Delete a project.

craton project-list List all projects.

craton project-show Show detailed information about a project.

craton project-update Update information about a project.

craton region-create Create a new region.

craton region-delete Delete a region.

craton region-list List all regions.

craton region-show Show detailed information about a region.

craton region-update Update information about a region.

craton cell-create Create a new cell.

craton cell-delete Delete a cell.

craton cell-list List all cells.

craton cell-show Show detailed information about a cell.

craton cell-update Update information about a cell.

craton device-create Create a new device.

craton device-delete Delete a device.

craton device-list List all devices.

craton device-show Show detailed information about a device.

craton device-update Update information about a device.

craton host-create Create a new host.

craton host-delete Delete a host.

craton host-list List all hosts.

craton host-show Show detailed information about a host.

craton host-update Update information about a host.

craton user-list List the users of a project.

craton help Display help about this program or one of its subcommands.

craton optional arguments

--version
Show program's version number and exit.

-v, --verbose
Print more verbose output.

craton project-create

Create a new project.

```
usage: craton project-create [-n <name>] [-u <uuid>]
```

Optional arguments:

-n <name>, **--name** <name>
Name of the project.

-u <uuid>, **--uuid** <uuid>
UUID of the project.

craton project-delete

Delete a project.

```
usage: craton project-delete <project>
```

Positional arguments:

project
UUID of the project.

craton project-list

List the projects.

```
usage: craton project-list [--detail] [--limit <limit>]
```

Optional arguments:

--detail
Show detailed information about the projects.

--limit <limit>
Maximum number of projects to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

craton project-show

Show detailed information about a project.

```
usage: craton project-show <project>
```

Positional arguments:

project
UUID of the project.

craton project-update

Update information about a project.

```
usage: craton project-update <project> [-n <name>]
```

Positional arguments:

project
UUID of the project.

Optional arguments:

-n <name>, **--name** <name>
New name for the project.

craton region-create

Create a new region.

```
usage: craton region-create [-n <name>]
                            [-u <uuid>]
                            [-p <project>]
                            [--note <note>]
```

Optional arguments:

-n <name>, **--name** <name>
Name of the region.

-u <uuid>, **--uuid** <uuid>
UUID of the region.

-p <project>, **--project** <project>, **--project_uuid** <project>
UUID of the project that this region belongs to.

--note <note>
Note about the region.

craton region-delete

Delete a region.

```
usage: craton region-delete <region>
```

Positional arguments:

region
UUID of the region.

craton region-list

List the regions.

```
usage: craton region-list [--detail] [--limit <limit>]
                          [--sort-key <field>] [--sort-dir <direction>]
                          [--fields <field> [<field> ...]]
```

Optional arguments:

--detail
Show detailed information about the regions.

--limit <limit>
Maximum number of regions to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

--sort-key <field>
Region field that will be used for sorting.

--sort-dir <direction>
Sort direction: “asc” (the default) or “desc”.

--fields <field> [<field> ...]
One or more region fields. Only these fields will be fetched from the server. Can not be used when ‘– detail’ is specified.

craton region-show

Show detailed information about a region.

```
usage: craton region-show <region>
```

Positional arguments:

region
UUID of the region.

craton region-update

Update information about a region.

```
usage: craton region-update <region> [-n <name>]
```

Positional arguments:

region
UUID of the region.

Optional arguments:

-n <name>, **--name** <name>
New name for the region.

craton cell-create

Create a new cell.

```
usage: craton cell-create [-n <name>]
                          [-u <uuid>]
                          [-p <project>]
                          [-r <region>]
                          [--note <note>]
```

Optional arguments:

-n <name>, **--name** <name>
Name of the cell.

-u <uuid>, **--uuid** <uuid>
UUID of the cell.

-p <project>, **--project** <project>, **--project_uuid** <project>
UUID of the project that this cell belongs to.

-r <region>, **--region** <region>, **--region_uuid** <region>
UUID of the region that this cell belongs to.

--note <note>
Note about the cell.

craton cell-delete

Delete a cell.

```
usage: craton cell-delete <cell>
```

Positional arguments:

cell
UUID of the cell.

craton cell-list

List the cells.

```
usage: craton cell-list [--detail] [--limit <limit>]
                        [--sort-key <field>] [--sort-dir <direction>]
                        [--fields <field> [<field> ...]]
                        [--region <region>]
```

Optional arguments:

--detail
Show detailed information about the cells.

-r <region>, **--region** <region>
UUID of the region that contains the desired list of cells.

--limit <limit>
Maximum number of cells to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

--sort-key <field>
Cell field that will be used for sorting.

--sort-dir <direction>
Sort direction: "asc" (the default) or "desc".

--fields <field> [<field> ...]
One or more cell fields. Only these fields will be fetched from the server. Can not be used when '– detail' is specified.

craton cell-show

Show detailed information about a cell.

```
usage: craton cell-show <cell>
```

Positional arguments:

cell
UUID of the cell.

craton cell-update

Update information about a cell.

```
usage: craton cell-update <cell> [-n <name>]
```

Positional arguments:

cell
UUID of the cell.

Optional arguments:

-n <name>, **--name** <name>
New name for the cell.

craton device-create

Create a new device.

```
usage: craton device-create [-n <name>]
                            [-t <type>]
                            [-a <active>]
                            [-u <uuid>]
                            [-p <project>]
                            [-r <region>]
                            [-c <cell>]
                            [--note <note>]
```

Optional arguments:

-n <name>, **--name** <name>
Name of the device.

-t <type>, **--type** <type>
Type of device.

-a <active>, **--active** <active>
Active or inactive state for a device: 'true' or 'false'.

-u <uuid>, **--uuid** <uuid>
UUID of the device.

-p <project>, **--project** <project>, **--project_uuid** <project>
UUID of the project that this device belongs to.

-r <region>, **--region** <region>, **--region_uuid** <region>
UUID of the region that this device belongs to.

-c <cell>, **--cell** <cell>, **--cell_uuid** <cell>
UUID of the cell that this device belongs to.

--note <note>
Note about the device.

craton device-delete

Delete a device.

```
usage: craton device-delete <device>
```

Positional arguments:

device
UUID of the device.

craton device-list

List the devices.

```
usage: craton device-list [--detail] [--limit <limit>]
                          [--sort-key <field>] [--sort-dir <direction>]
                          [--fields <field> [<field> ...]]
                          [--cell <cell>]
```

Optional arguments:

-c <cell>, **--cell** <cell>
UUID of the cell that contains the desired list of devices.

--detail
Show detailed information about the device.

--limit <limit>
Maximum number of devices to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

--sort-key <field>
Device field that will be used for sorting.

--sort-dir <direction>
Sort direction: "asc" (the default) or "desc".

--fields <field> [<field> ...]
One or more device fields. Only these fields will be fetched from the server. Can not be used when '– detail' is specified.

craton device-show

Show detailed information about a device.

```
usage: craton device-show <device>
```

Positional arguments:

device
UUID of the device.

craton device-update

Update information about a device.

```
usage: craton device-update <device> [-n <name>]
```

Positional arguments:

device
UUID of the device.

Optional arguments:

-n <name>, **--name** <name>
New name for the device.

craton host-create

Create a new host.

```
usage: craton host-create [-n <name>]
                          [-t <type>]
                          [-a <active>]
                          [-u <uuid>]
                          [-p <project>]
                          [-r <region>]
                          [-c <cell>]
                          [--note <note>]
```

```
                        [--access_secret <access_secret>]
                        [-i <ip_address>]
```

Optional arguments:

-n <name>, **--name** <name>
Name of the host.

-t <type>, **--type** <type>
Type of host.

-a <active>, **--active** <active>
Active or inactive state for a host: 'true' or 'false'.

-u <uuid>, **--uuid** <uuid>
UUID of the host.

-p <project>, **--project** <project>, **--project_uuid** <project>
UUID of the project that this host belongs to.

-r <region>, **--region** <region>, **--region_uuid** <region>
UUID of the region that this host belongs to.

-c <cell>, **--cell** <cell>, **--cell_uuid** <cell>
UUID of the cell that this host belongs to.

--note <note>
Note about the host.

--access_secret <access_secret>
UUID of the access secret of the host.

-i <ip_address>, **--ip_address** <ip_address>
IP Address type of the host.

craton host-delete

Delete a host.

```
usage: craton host-delete <host>
```

Positional arguments:

host
UUID of the host.

craton host-list

List the hosts.

```
usage: craton host-list [--detail] [--limit <limit>]
                        [--sort-key <field>] [--sort-dir <direction>]
                        [--fields <field> [<field> ...]]
                        [--cell <cell>]
```

Optional arguments:

-c <cell>, **--cell** <cell>
UUID of the cell that contains the desired list of hosts.

--detail
Show detailed information about the host.

--limit <limit>
Maximum number of hosts to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

--sort-key <field>
Host field that will be used for sorting.

--sort-dir <direction>
Sort direction: "asc" (the default) or "desc".

--fields <field> [<field> ...]
One or more host fields. Only these fields will be fetched from the server. Can not be used when '– detail' is specified.

craton host-show

Show detailed information about a host.

```
usage: craton host-show <host>
```

Positional arguments:

host
UUID of the host.

craton host-update

Update information about a host.

```
usage: craton host-update <host> [-n <name>]
```

Positional arguments:

host
UUID of the host.

Optional arguments:

-n <name>, **--name** <name>
New name for the host.

craton user-list

List the users in a project.

```
usage: craton user-list [--detail] [--limit <limit>]
                        [--sort-key <field>] [--sort-dir <direction>]
                        [--fields <field> [<field> ...]]
```

Optional arguments:

--detail
Show detailed information about the users.

--limit <limit>
Maximum number of users to return per request, 0 for no limit. Default is the maximum number used by the Craton API Service.

--sort-key <field>
User field that will be used for sorting.

--sort-dir <direction>
Sort direction: “asc” (the default) or “desc”.

--fields <field> [<field> ...]
One or more user fields. Only these fields will be fetched from the server. Can not be used when ‘– detail’ is specified.

Usage

Import

To use craton in a project:

```
import craton
```

CHAPTER 2

Developer Guide

Fleet Management Service Specifications

All current approved Craton API specifications:

All implemented Craton API specifications:

Listing Devices

https://blueprints.launchpad.net/craton/+spec/list-devices

Craton has separate endpoints for different types of device. Devices of different types can be linked in a parent-child relationship. Craton does not offer a mechanism to easily display devices of different types making queries tracking relationships cumbersome.

Problem description

As a operator I want to be able to list a device's descendants so that I can visualise or operate on a collection of related devices.

Currently Craton supports two types of devices - hosts and network-devices. Devices include the optional attribute parent_id to create a parent-child relationship between two devices. So if one has two network-devices and one is the parent of the other, the network-devices endpoint can be queried to find the child device using the ID of the parent, e.g.

```
GET /v1/network-devices?parent_id=1
```

If a third device is added, this time as a child device of the second device, it is not possible to directly identify it from the root device. A second query would need to be made using the ID of the second device, e.g.

```
GET /v1/network-devices?parent_id=2
```

This means to represent a complete tree could potentially require a large number of queries or the client would need to get all the devices and then link them up itself.

In addition, given that both a host and a network-device can have the same parent, currently both endpoints need to be queried for any parent_id to get all the devices.

Proposed change

To meet the needs of the user story and resolve the problems outlined above, this spec proposes the introduction of a new endpoint for devices to allow for the querying of devices as a whole.

The endpoint will be /v1/devices and will support:

- querying against a set of attributes common to all devices
- optionally including the descendants of any query

Alternatives

- the traversal of the tree could be left to the client, this would likely be a slow process for large deployments
- the existing endpoints, i.e. /v1/hosts and /v1/network-devices, could be allowed to return other types of device however this is likely to be

confusing and lead to mistakes uses the output.

Data model impact

None

REST API impact

Endpoint: /v1/devices Method: GET Description: List project devices Normal response code: 200 Expected error response codes: 400

Parameters schema: { “type”: “object”, “additionalProperties”: False, “properties”: {

“id”: { “type”: “integer”,

}, “region_id”: {

“type”: “integer”,

}, “cell_id”: {

“type”: “integer”,

}, “parent_id”: {

“type”: “integer”,

}, “active”: {

“type”: “boolean”,

}, “descendants”: {

“default”: False, “type”: “boolean”,

},

},

}

Response schema: { “type”: “object”, “additionalProperties”: False, “properties”: {

“devices”: { “type”: “object”, “additionalProperties”: False, “properties”: {

“hosts”: { “type”: “array”, “items”: DefinitionsHost,

}, “network-devices”: {

“type”: “array”, “items”: DefinitionNetworkDeviceResponse,

},

},

}, “links”: DefinitionsPaginationLinks, },

},

}

Example: Request

http://example.com/v1/devices

Response {

“devices”: {

“hosts”: [

{ “active”: true, “cell_id”: 4, “created_at”: “2017-02-16T14:28:55.000000”, “device_type”: “server”, “id”: 20, “ip_address”: “192.168.1.20”, “links”: [

{ “href”: “http://example.com/v1/cells/4”, “rel”: “up”

}

], “name”: “host1.DFW.C0002.C-2.example2.com”, “note”: null, “parent_id”: null, “project_id”: “b9f10eca-66ac-4c27-9c13-9d01e65f96b4”, “region_id”: 2, “updated_at”: null

} ... more hosts ...,

], “network-devices”: [

{ “access_secret_id”: null, “active”: true, “cell_id”: 4, “created_at”: “2017-02-16T14:28:55.000000”, “device_type”: “switch”, “id”: 16, “ip_address”: “10.10.1.1”, “links”: [

{ “href”: “http://example.com/v1/cells/4”, “rel”: “up”

}

], “model_name”: “model-x”, “name”: “switch1.C0002.DFW.example.com”, “os_version”: “version-1”, “parent_id”: null, “project_id”: “b9f10eca-66ac-4c27-9c13-9d01e65f96b4”, “region_id”: 2, “updated_at”: null, “vlans”: null

}, ... more network-devices ...,

],

}, “links”: [

{ “href”: “http://example.com/v1/devices?sort_dir=asc&limit=30&sort_keys=created_at%2Cid”, “rel”: “first”

}, {

"href": "http://example.com/v1/devices?sort_dir=asc&limit=30&sort_keys=created_at%2Cid", "rel": "prev"

}, {

"href": "http://example.com/v1/devices?sort_dir=asc&limit=30&sort_keys=created_at%2Cid", "rel": "self"

}, {

"href": "http://example.com/v1/devices?sort_dir=asc&limit=30&sort_keys=created_at%2Cid&marker=20", "rel": "next"

}

]

}

Example: Request

http://example.com/v1/devices?parent_id=16&descendants=true

Response {

"devices": {

"network-devices": [

{ "access_secret_id": null, "active": true, "cell_id": 4, "created_at": "2017-02-16T14:28:55.000000", "device_type": "switch", "id": 17, "ip_address": "10.10.1.2", "links": [

{ "href": "http://example.com/v1/network-devices/16", "rel": "up"

}

], "model_name": "model-x", "name": "switch2.C0002.DFW.example.com", "os_version": "version-1", "parent_id": 16, "project_id": "b9f10eca-66ac-4c27-9c13-9d01e65f96b4", "region_id": 2, "updated_at": null, "vlans": null

}, {

"access_secret_id": null, "active": true, "cell_id": 4, "created_at": "2017-02-16T14:28:55.000000", "device_type": "switch", "id": 18, "ip_address": "10.10.1.3", "links": [

{ "href": "http://example.com/v1/network-devices/17", "rel": "up"

}

], "model_name": "model-x", "name": "switch3.C0002.DFW.example.com", "os_version": "version-1", "parent_id": 17, "project_id": "b9f10eca-66ac-4c27-9c13-9d01e65f96b4", "region_id": 2, "updated_at": null, "vlans": null

},

], "hosts": [

{ "active": true, "cell_id": 4, "created_at": "2017-02-16T14:28:55.000000", "device_type": "server", "id": 200, "ip_address": "192.168.1.20", "links": [

{ "href": "http://example.com/v1/network-devices/16", "rel": "up"

}

], “name”: “host10.DFW.C0002.C-2.example2.com”, “note”: null, “parent_id”: 16, “project_id”: “b9f10eca-66ac-4c27-9c13-9d01e65f96b4”, “region_id”: 2, “updated_at”: null

},

],

}, “links”: [

{ “href”: “http://example.com/v1/devices?parent_id=16&descendants=true&sort_dir=asc&limit=30&sort_keys=created_at%2Cid”, “rel”: “first”

}, {

“href”: “http://example.com/v1/devices?parent_id=16&descendants=true&sort_dir=asc&limit=30&sort_keys=created_at%2Cid”, “rel”: “prev”

}, {

“href”: “http://example.com/v1/devices?parent_id=16&descendants=true&sort_dir=asc&limit=30&sort_keys=created_at%2Cid”, “rel”: “self”

}, {

“href”: “http://example.com/v1/devices?parent_id=16&descendants=true&sort_dir=asc&limit=30&sort_keys=created_at%2Cid&marker=20”, “rel”: “next”

}

]

}

Security impact

None

Notifications impact

None

Other end user impact

- /v1/devices with need to be supported by the client.

Performance Impact

Given the nature of this new endpoint, there is a strong likelihood that it will be used for most requests where listing devices is required, even if the user is only after one type.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee: - git-harry

Other contributors: - None

Work Items

- add /v1/devices endpoint

Dependencies

None

Testing

A full set of functional and unit tests will need to be added.

Documentation Impact

The repo documentation will require updating but this is handled by the project.

References

None

Pagination of List Resources

https://blueprints.launchpad.net/craton/+spec/pagination-of-resources

Craton is intended to manage large quantities of devices and other objects without sacrificing performance. Craton needs to add pagination support in order to efficiently handle queries on large collections.

Problem description

In the current implementation, a request to one of our collection resources will attempt to return all of the values that can be returned (based on authentication, etc.). For example, if a user and project have access to 5000 hosts then making a `GET` request against `/v1/hosts` would return all 5000. Such large result sets can and likely will slow down Craton's response times and make it unusable.

Proposed change

We propose adding pagination query parameters to all collection endpoints. The new parameters would assume defaults if the user does not include them.

We specifically propose that:

1. Craton choose a default page size of 30 and limit it to being at least 10 items and at most 100 items,
2. Craton choose to make the next page both discoverable *and* calculable. In other words, using "link" hypermedia relations in a response to indicate first, previous, next, and last page URLs that are generated by the server for the client,
3. Craton should assume the defaults for requests that have no query parameters. For example, if someone makes a `GET` request to `/v1/hosts` it would imply an original page size of 30 and that the first 30 results should be returned.

To provide pagination to users, it is suggested that we use `limit` and `marker` parameters to indicate the page size and last seen ID. This allows users to begin pagination after an item, rather than at a particular page. For example, if a user is checking for new hosts in the listing and they know the ID of the last host they encountered they can provide `marker=:id&limit=30` to get the newer hosts. If instead, we used `page` and `per_page` there's the possibility they'd miss items since hosts may have been deleted changing the page number of the last host.

This implies that the default `limit` value would be 30 and the default `marker` would be null (to indicate that no last ID is seen).

This combination of parameters is practically the standard in OpenStack. Operators familiar with OpenStack's existing Compute, Images, etc. APIs will be familiar with these parameters.

In addition to pagination parameters, this spec proposes adding link relations in the Response body - as defined by JSON Hyper-Schema and favored by the API WG

This makes API usage easier for everyone, including, people using the API directly and people writing API wrappers such as python-cratonclient. This does, however, have the downside of affecting our response bodies and JSON Schema

Finally, I'd like to strongly propose that we include these links in each response. Which relation types we include would depend on where in the pagination the user is, but it would do something like this:

1. Include a `self` relation for every page that tells the user exactly what page they're presently on.
2. If there is a page prior to the current one, we would include the `prev` **and** `first` relations. These tell the user what the previous page is and what the first page is.
3. If there is a page after the current one, we would include the `next` **and** `last` relations. These are the opposites to `prev` and `first` respectively.

   It is worth noting that without properly implemented caching the `last` relation, it could become computationally expensive to calculate for every pagination query.

Alternatives

Alternative query parameters to `limit` and `marker` are:

1. Use `page` and `per_page` parameters to indicate the 1-indexed "page number" and number of items on each page respectively. This means that users can change how many items they get on each page request and can resume in arbitrary places by specifying the `page` parameter.

   This would imply that the default `page` value would be 1 and the default `per_page` would be 30.

These two parameters are presently used by a significant number of large APIs at the moment but are not common in OpenStack itself. They provide simplicity in that if the API user wants to, they can just constantly increment the page number to get the next page in the simplest way possible. They don't have to calculate the next value from a combination of values in the response of the last request.

This does, however, prevent users from being able to resume iteration from the last item it received in a list. Further, this adds the potential that users may miss objects due to deletions or other changes in the corresponding collection. Finally, these parameters only provide users an opaque idea as to where in a paginated resource they are and how to resume pagination.

2. Use `limit` and `offset` parameters to provide similar functionality and opacity to `per_page` and `page` respectively.

   The default `limit` would, again, be 30 and the default `offset` would be 0.

   This combination of parameters is also present in a small number of OpenStack projects but has some of the same negative implications as the `page` and `per_page` parameters when compared to `limit` and `marker`.

An alternative way to provide pagination links are:

1. Link headers - as defined in **RFC 6903** - using Relation Types defined in **RFC 5988**.

   These are also commonly used outside of OpenStack and were popular to the creation of including the relations in the response body. The benefit to Craton of using this method is that it doesn't effect our JSON Schema or existing Response bodies. A major problem with this approach is that a relation type can be repeated in a Link header. However, the HTTP library used by the majority of the Python world - Requests - does not parse such links correctly. Further, widespread support for parsing these header values is not known to the author of this specification.

Data model impact

This should have **no** impact on our data model.

REST API impact

This specification will have two impacts on our REST API:

1. It will add `limit` and `marker` query parameters that are identical to a number of existing and future endpoints.
2. It will change the fundamental structure of our list responses in order to accommodate the link relations.

   At the moment, for example, a `GET` request made to `/v1/hosts` has a response body that looks like:


```
[
  {
     "active": true,
     "cell_id": null,
     "device_type": "Computer",
     "id": 1,
     "ip_address": "12.12.12.15",
     "name": "foo2Host",
     "note": null,
     "parent_id": null,
     "region_id": 1
  },
  {
     "active": true,
     "cell_id": null,
```

```
    "device_type": "Phone",
    "id": 2,
    "ip_address": "11.11.11.14",
    "name": "fooHost",
    "note": null,
    "parent_id": null,
    "region_id": 1
  }
]
```

This would need to transform to

```
  {
    "items": [
      {
         "active": true,
         "cell_id": null,
         "device_type": "Computer",
         "id": 1,
         "ip_address": "12.12.12.15",
         "name": "foo2Host",
         "note": null,
         "parent_id": null,
         "region_id": 1
      },
      {
         "active": true,
         "cell_id": null,
         "device_type": "Phone",
         "id": 2,
         "ip_address": "11.11.11.14",
         "name": "fooHost",
         "note": null,
         "parent_id": null,
         "region_id": 1
      }
    ],
    "links": [
      {
        "rel": "first",
        "href": "https://craton.environment.com/v1/hosts?limit=30"
      },
      {
        "rel": "next",
        "href": "https://craton.environment.com/v1/hosts?limit=30&marker=2"
      },
      {
        "rel": "self",
        "href": "https://craton.environment.com/v1/hosts?limit=30&marker=1"
      }
    ]
}
```

Security impact

Pagination suppport reduces the potential attack surface for denial of service attacks aimed at Craton. It alone, however, is not sufficient to prevent DoS attacks and additional measures should be taken by deployers to further mitigate those possibilities.

Notifications impact

Craton does not yet have notifications.

Other end user impact

This will have a minor affect on python-cratonclient. The `list` calls it implements will need to become smarter so they can handle pagination for the user automatically.

Performance Impact

There should not be any performance impact on the service created by this code although it will frequently be called.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee: - icordasc

Other contributors: - None

Work Items

- Add basic pagination support with tests to ensure that functionality works independent of the other features proposed in this specification
- Add link relation support to response bodies

Dependencies

N/A

Testing

This should be tested on different levels, but at a minimum on a functional level.

Documentation Impact

This will impact our API reference documentation

References

- IANA Link Relations Registry
- **RFC 5988**
- **RFC 6903**
- JSON Hyper-Schema
- "Pagination, Filtering, and Sorting" by the OpenStack API WG

Craton URL Structure and Design

Blueprint https://blueprints.launchpad.net/craton/+spec/url-structure-and-design

Craton developers decided to start modifying the URL structure and semantics prior to creating a release. This has led down a number of paths which require documentation and understanding prior to resolving ourselves on one such structure and semantic meaning.

Problem description

Presently, Craton's API requires query parameters for certain calls. For example,

- To list hosts, one **must** specify a region ID:

```
GET /v1/hosts?region_id=1
```

- To list cells, one **must** specify a region ID:

```
GET /v1/cells?region_id=1
```

To make the API easier to use for others, as well as easier to use when performing checks across the inventory, Craton is looking to remove required query parameters.

Proposed change

Query parameters are typically optional and have always been a poorly considered choice for a required parameter. Instead, the Craton team proposes that we adopt a flat URL structure and design while continuing to allow filtering based on attributes that were formerly required.

Now users will be able to list all hosts and cells that their project allows them to view:

```
GET /v1/hosts
GET /v1/cells
```

While also allowing them to filter those based on variables and other attributes:

```
GET /v1/hosts?vars=operating_system:ubuntu
GET /v1/cells?region_id=1
```

This change, however, will increase the priority of completing work around adding pagination support to Craton. As such, adding support for pagination is a work item of this specification.

Alternatives

We could retain our current way of using query parameters. This, however, is unseemly, unusual, and an unpleasant experience for users. If we were to continue requiring parameters, e.g., `region_id`, we would instead be adopting a dfifferent URL structure.

Data model impact

There are no database or data model impacts implied by this change.

REST API impact

This makes the API easier to use and reason about for users new to Craton's API.

Security impact

Proper pagination support is necessary to prevent requests returning large collections of resources.

Notifications impact

Craton does not presently have notifications, so there is no impact.

Other end user impact

This will affect the command-line interface to cratonclient. As region IDs are no longer necessary for listing resources, that requirement will need to be relaxed in our parameter handling.

Performance Impact

With proper pagination, this should have a neglible (if any) impact on Craton's performance.

Other deployer impact

This will not affect people who are deploying Craton.

Developer impact

This has no other developer impact beyond API usage.

Implementation

Assignee(s)

Primary assignee: - git-harry

Other contributors: - icordasc

Work Items

- Refactor API layer to stop requiring parameters in the query string (See also: https://review.openstack.org/408016)
- Add pagination support for endpoints returning collections of resources.

Dependencies

N/A

Testing

We will update and continue to use our current functional testing.

Documentation Impact

This will affect the API reference section of our documentation.

References

- https://review.openstack.org/408016
- https://review.openstack.org/400198
- https://review.openstack.org/401958

Indices and Tables

- search

Contributing

Developer's Guide

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

http://docs.openstack.org/infra/manual/developers.html

Development Workflow

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

http://docs.openstack.org/infra/manual/developers.html#development-workflow

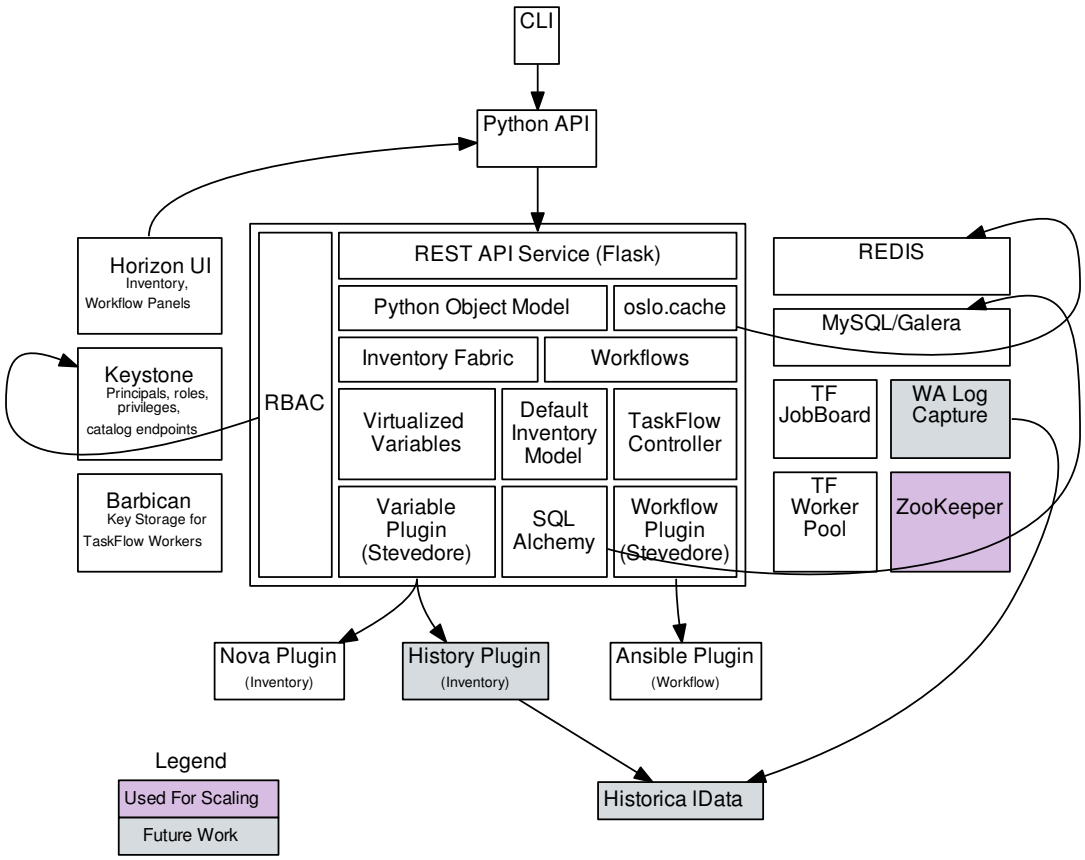
Engagement

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

https://bugs.launchpad.net/craton

Architecture



CLI

TODO: Add Documentation

Python API

TODO: Add Documentation

RBAC

TODO: Add Documentation

REST API Service (Flask)

TODO: Add Documentation

Python Object Model

TODO: Add Documentation

oslo.cache

TODO: Add Documentation

Inventory Fabric

TODO: Add Documentation

Workflows

TODO: Add Documentation

Virtualized Variables

TODO: Add Documentation

Default Inventory Mode

TODO: Add Documentation

TaskFlow Controller

TODO: Add Documentation

Variable Plugin (Stevedore)

TODO: Add Documentation

SQL Alchemy

TODO: Add Documentation

Workflow Plugin (Stevedore)

TODO: Add Documentation

Nova Plugin

TODO: Add Documentation

History Plugin

TODO: Add Documentation

Ansible Plugin

TODO: Add Documentation

Craton's API Reference Guide

Resources:

Cells

Definition of cell

Create Cell

POST /v1/cells

Create a new Cell

Normal response codes: OK(201)

Error response codes: invalid request(400), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the cell
region_id	body	integer	Unique ID of the region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Cell Create

```
curl -i "http://${MY_IP}:7780/v1/cells" \
     -d '{"name": "myCell", "region_id": 1}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
cell	body	object	• id • name • region_id • labels • note • variables
id	body	integer	Unique ID of the cell
name	body	string	Unique name of the cell
region_id	body	integer	Unique ID of the cell's region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "id": 1,
   "name": "myCell",
   "note": null,
```

```
    "region_id": 1
}
```

List Cells

GET /v1/cells?region_id=

Gets all Cells

Normal response codes: OK(200)

Error response codes: invalid request(400), cell not found(404), validation exception(405)

Default response: unexpected error

Request

Name	In	Type	Required	Description
region_id	query	string	Yes	ID of the region to get cells for

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Cell List

```
curl -i "http://${MY_IP}:7780/v1/cells?region_id=1" \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
cells	body	array	Array of cell objects
id	body	integer	Unique ID of the cell
name	body	string	Unique name of the cell
region_id	body	integer	Unique ID of the cell's region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
[
    {
        "id": 2,
        "name": "cellJr",
        "note": null,
        "region_id": 1
    },
    {
        "id": 1,
        "name": "myCell",
        "note": null,
        "region_id": 1
    }
]
```


Todo

Example Unexpected Error

..literalinclude:: ./api_samples/errors/errors-unexpected-resp.json

language javascript

Update Cells

PUT /v1/cells/{id}

Update an existing cell

Normal response codes: OK(200)

Error response codes: invalid request(400), cell not found(404), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the cell
labels	body	string	User defined labels
note	body	string	Note used for governance

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Cell Update

```
curl -i "http://${MY_IP}:7780/v1/cells/1" \
     -XPUT \
     -d '{"name": "changedName"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
cell	body	object	• id • name • region_id • labels • note • variables
id	body	integer	Unique ID of the cell
name	body	string	Unique name of the cell
region_id	body	integer	Unique ID of the cell's region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "id": 1,
   "name": "changedName",
   "note": null,
   "project_id": "717e9a21-6e2d-44e0-bc84-8398563bda06",
   "region_id": 1
}
```

Update Cell Variables

PUT /v1/cells/{id}/variables

Update user defined variables for the cell

Normal response codes: OK(200)

Error response codes: invalid request(400), cell not found(404), validation exception(405)

Request

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data
id	path	integer	Unique ID of the cell to be updated

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Cell Update Variables

```
curl -i "http://${MY_IP}:7780/v1/cells/1/variables" \
     -XPUT \
     -d '{"newKey": "sampleKey"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data

```
{
   "variables":
   {
      "newKey": "sampleKey"
   }
}
```

Delete Cell

DELETE /v1/cells/{id}

Deletes an existing record of a Cell

Normal response codes: no content(204)

Error response codes: invalid request(400), cell not found(404)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the cell to be deleted

Required Header

- Content-Type: application/json

- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

Delete Cell Variables

DELETE /v1/cells/{id}/variables

Delete existing key/value variables for the cell

Normal response codes: no content(204)

Error response codes: invalid request(400), cell not found(404) validation exception(405)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the cell
key	body	string	Identifier to be deleted
value	body	object	Data to be deleted

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

Hosts

Definition of host

Create Host

POST /v1/hosts

Create a new host

Normal response codes: OK(201)

Error response codes: invalid request(400), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the host
cell_id	body	integer	Unique ID of the host's cell
region_id	body	integer	Unique ID of the host's region
parent_id	body	integer	ID of the host's parent
ip_address	body	string	IP address of the host
device_type	body	string	Type of host
active	body	boolean	State of host
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Host Create

```
curl -i "http://${MY_IP}:7780/v1/hosts" \
     -d '{"name": "fooHost", "region_id": 1, "ip_address": "11.11.11.14", "device_type
↪": "Phone"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
host	body	object	• id • name • cell_id • region_id • parent_id • ip_address • device_type • active • labels • note • variables
id	body	integer	Unique ID of the host
name	body	string	Unique name of the host
cell_id	body	integer	Unique ID of the host's cell
region_id	body	integer	Unique ID of the host's region
parent_id	body	integer	ID of the host's parent
ip_address	body	string	IP address of the host
device_type	body	string	Type of host
active	body	boolean	State of host
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "active": true,
   "cell_id": null,
   "device_type": "Phone",
   "id": 1,
   "ip_address": "11.11.11.14",
   "name": "fooHost",
   "note": null,
   "parent_id": null,
   "region_id": 1
}
```

List Hosts

GET /v1/hosts?region_id=

Gets all Host

Normal response codes: OK(200)

Error response codes: invalid request(400), host not found(404), validation exception(405)

Default response: unexpected error

Request

Name	In	Type	Required	Description
region_id	query	integer	Yes	ID of the region to get hosts
limit	query	integer	No	Number of host to return Ranging from 1 - 10000
name	query	string	No	Name of the host to get
cell_id	query	integer	No	Name of the cell to get
ip	query	string	No	IP address of the host to get
device_type	query	string	No	Type of host to get

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Examples Host List

```
curl -i "http://${MY_IP}:7780/v1/hosts?region_id=1" \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
hosts	body	array	array of host
id	body	integer	Unique ID of the host
name	body	string	Unique name of the host
cell_id	body	integer	Unique ID of the host's cell
region_id	body	integer	Unique ID of the host's region
parent_id	body	integer	ID of the host's parent
ip_address	body	string	IP address of the host
device_type	body	string	Type of host
active	body	boolean	State of host
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
[
   {
      "active": true,
      "cell_id": null,
      "device_type": "Computer",
      "id": 2,
      "ip_address": "12.12.12.15",
```

```
        "name": "foo2Host",
        "note": null,
        "parent_id": null,
        "region_id": 1
    },
    {
        "active": true,
        "cell_id": null,
        "device_type": "Phone",
        "id": 1,
        "ip_address": "11.11.11.14",
        "name": "fooHost",
        "note": null,
        "parent_id": null,
        "region_id": 1
    },
]
```


Todo

Example Unexpected Error

..literalinclude:: ./api_samples/errors/errors-unexpected-resp.json

language javascript

Update Hosts

PUT /v1/hosts/{id}

Update an existing host

Normal response codes: OK(200)

Error response codes: invalid request(400), host not found(404), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the host
cell_id	body	integer	Unique ID of the host's cell
region_id	body	integer	Unique ID of the host's region
parent_id	body	integer	ID of the host's parent
ip_address	body	string	IP address of the host
device_type	body	string	Type of host
active	body	boolean	State of host
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables
id	path	integer	Unique ID of the host to be updated

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Examples Host Update

```
curl -i "http://${MY_IP}:7780/v1/hosts/2" \
     -XPUT \
     -d '{"name": "changedName"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
host	body	object	• id • name • cell_id • region_id • parent_id • ip_address • device_type • active • labels • note • variables
id	body	integer	Unique ID of the host
name	body	string	Unique name of the host
cell_id	body	integer	Unique ID of the host's cell
region_id	body	integer	Unique ID of the host's region
parent_id	body	integer	ID of the host's parent
ip_address	body	string	IP address of the host
device_type	body	string	Type of host
active	body	boolean	State of host
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "active": true,
   "cell_id": null,
```

```
    "device_type": "Computer",
    "id": 2,
    "ip_address": "12.12.12.15",
    "name": "changedName",
    "note": null,
    "project_id": "717e9a21-6e2d-44e0-bc84-8398563bda06",
    "region_id": 1
}
```

Update Host variables

PUT /v1/hosts/{id}/variables

Update user defined variables for the host

Normal response codes: OK(200)

Error response codes: invalid request(400), host not found(404), validation exception(405)

Request

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data
id	path	integer	Unique ID of the host to be updated

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Host Variables Update

```
curl -i "http://${MY_IP}:7780/v1/hosts/1/variables" \
     -XPUT \
     -d '{"newVar": "sample variable"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data

```
{
   "variables":
    {
       "newVar": "sample variable"
    }
}
```

Delete Host

DELETE /v1/hosts/{id}

Deletes an existing record of a Host

Normal response codes: no content(204)

Error response codes: invalid request(400), host not found(404)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the host to be deleted

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

Delete Host Variables

DELETE /v1/hosts/{id}/variables

Delete existing key/value variables for the Host

Normal response codes: no content(204)

Error response codes: invalid request(400), host not found(404) validation exception(405)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the host
key	body	string	Identifier to be deleted
value	body	object	Data to be deleted

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

Regions

Definition of region

Create Region

POST /v1/region

Creates a new Region

Normal response codes: OK(201)

Error response codes: invalid request(400), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Region Create

```
curl -i "http://${MY_IP}:7780/v1/regions" \
     -d '{"name": "DFW"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
region	body	object	• id • name • cells • labels • note • variables
id	body	integer	Unique ID of the region
name	body	string	Unique name of the region
cells	body	array	Array of cells
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "id": 1,
   "name": "DFW",
   "note": null
}
```

List Regions

GET /v1/regions

Gets all Regions

Normal response codes: OK(200)

Error response codes: invalid request(400), validation exception(405)

Default response: unexpected error

Request

No parameters

Required Header

- Content-Type: application/json
- X-Auth-Token

- X-Auth-User
- X-Auth-Project

Example Region List

```
curl -i "http://${MY_IP}:7780/v1/regions" \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
regions	body	array	Array of regions
id	body	integer	Unique ID of the region
name	body	string	Unique name of the region
cells	body	array	Array of cells in region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
[
   {
      "id": 1,
      "name": "DFW",
      "note": null
   },
   {
      "id": 2,
      "name": "DFW2",
      "note": null
   },
   {
      "id": 3,
      "name": "fooRegion",
      "note": null
   }
]
```

Todo

Example Unexpected Error

..literalinclude:: ./api_samples/errors/errors-unexpected-resp.json

language javascript

Update Region

PUT /v1/regions/{id}

Update an existing region

Normal response codes: OK(200)

Error response codes: invalid request(400), region not found(404), validation exception(405)

Request

Name	In	Type	Description
name	body	string	Unique name of the region
cells	body	array	Array of cells in region
labels	body	string	User defined labels
note	body	string	Note used for governance
id	path	integer	Unique ID of the region to be updated

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Region Update

```
curl -i "http://${MY_IP}:7780/v1/regions/3" \
     -XPUT \
     -d '{"name": "DFW3"}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
region	body	object	• id • name • cells • labels • note • variables
id	body	integer	Unique ID of the region
name	body	string	Unique name of the region
cells	body	array	Array of cells in region
labels	body	string	User defined labels
note	body	string	Note used for governance
variables	body	object	User defined variables

```
{
   "id": 3,
   "name": "DFW3",
   "note": null,
   "project_id": "717e9a21-6e2d-44e0-bc84-8398563bda06"
}
```

Update Region Variables

PUT /v1/regions/{id}/variables

Update user defined variables for the region

Normal response codes: OK(200)

Error response codes: invalid request(400), region not found(404), validation exception(405)

Request

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data
id	path	integer	Unique ID of the region to be updated

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Example Region Variables Update

```
curl -i "http://${MY_IP}:7780/v1/regions/3/variables" \
     -XPUT \
     -d '{"array": [2]}' \
     -H "Content-Type: application/json" \
     -H "X-Auth-Token: demo" \
     -H "X-Auth-User: demo" \
     -H "X-Auth-Project: 717e9a216e2d44e0bc848398563bda06"
```

Response

Name	In	Type	Description
key	body	string	Identifier
value	body	object	Data

```
{
   "variables":
   {
      "string": "sample text",
      "value": 24,
      "array": [2]
   }
}
```

Delete Region

DELETE /v1/regions/{id}

Deletes an existing record of a Region

Normal response codes: no content(204)

Error response codes: invalid request(400), region not found(404)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the region to be deleted

Required Header

- Content-Type: applicaton/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

Delete Region Variables

DELETE /v1/regions/{id}/variables

Delete existing key/value variables for the region

Normal response codes: no content(204)

Error response codes: invalid request(400), region not found(404) validation exception(405)

Request

Name	In	Type	Description
id	path	integer	Unique ID of the region
key	body	string	Identifier to be deleted
value	body	object	Data to be deleted

Required Header

- Content-Type: application/json
- X-Auth-Token
- X-Auth-User
- X-Auth-Project

Response

No body content is returned on a successful DELETE

API Usage:

Filtering Resources by Variables

This describes how to use variable queries when listing resources. This feature uses a subset of JSON Path supported by MySQL 5.7. Most notably, we do not support the `doubleAsterisk` component.

Supported Syntax

A variable query in our API consists of two main parts, separated by a colon (`:`):

1. The JSON path
2. The JSON value

You may supply as many of these as you like with each discrete query separated by a comma (`,`). For example, the following would all be valid queries against the Craton API:

```
GET /v1/hosts?vars=hardware_profiles.disks[*].manufacturer:"Seagate"
```

and

```
GET /v1/hosts?vars="os-information".release.version:"4.4.0",hardware.core_count:12
```

Path

The JSON Path expression is a series of path legs separated by a period ('.'). Each path leg can consist of the following components:

- A key, which can be either:
  - An ECMAScript identifier, such as `hardware_profiles` or `release`.

  - A JSON string, such as `"hyphenated-key"` or `"this-is-a-json-string"`
- A key and an array wildcard or specific index, like `foo[*]`, `foo.bar[*].key`, or `foo[3]`
- A wildcard character (`*`), to specify all keys at this hierachical level, e.g. : `foo.*.baz`

Value

The value portion of the query can consist of the following JSON data types:

- A JSON string, e.g. `"this-is-a-json-string"`
- A JSON boolean, i.e. `true` or `false`
- A JSON null, i.e. `null`
- A JSON integer, e.g. `42`
- A JSON float, e.g. `3.14`

Putting it All Together

Example 1

With this syntax, you can express powerful variable filters that afford for searching through nested metadata on a resource. Here's a quick example to illustrate the usefulness of this feature. Let's take some arbitrary hardware data that's been stored for each of our hosts:

```
{
    "hardware_profiles": {
        "disks": [
            {
                "manufacturer": "Seagate",
                "capacity_quantity": 2,
                "capacity_unit": "TB"
            },
            {
                "manufacturer": "Western Digital",
                "capacity_quantity": 3,
                "capacity_unit": "TB"
            }
        ]
    }
}
```

Now, let's say we want to find all of the hosts with a Seagate disk, one could accomplish this with the following query:

```
GET /v1/hosts?vars=hardware_profiles.disks[*].manufacturer:"Seagate"
```

Example 2

As another example, let's say we're a root user for Craton (meaning we have access across projects) - what if we wanted to get all hosts that are in, say, any Region that is in some specific data center and the way we're representing that on the Region resource(s) is:

```
{
    "datacenter_info": {
        "id": 543,
        "name": "DFW_DC_0"
    }
}
```

Because of how variables are inherited by child resources, we could query for all of these hosts by simply querying like so:

```
GET /v1/hosts?vars=datacenter_info.id:543
```

Limitations and Schema Considerations

Known Limitations

- Because MySQL 5.7 does not support slicing arrays (`foo[4:10]`, for instance), we do not support them in Craton.
- Although MySQL 5.7 does support the double-asterisk (`prefix**suffix`) in its syntax, we do not. This is due to how *jsonpath-rw*, the library we use for parsing the API response, doesn't include the double-asterisk in its JSON path flavor.
- The first key in the path must be known, because it does not participate in the JSON column search. It is a separate field altogether, really, but we allow one to append it to the beginning for convenience in the syntax.
- You cannot use a colon (`:`) in your JSON path or JSON value, since that is reserved for parsing the query itself.
- You cannot use a comma (`,`) in your JSON path or JSON value, since that is reserved for parsing the query itself.
- When no rows are in the Variables table, JSON Path validation does not occur at the DB.

Schema Considerations

We do not support wildcard values in the Value portion of the variables query. Therefore, it's a good idea to parse and store your data in a more consistent and normalized manner. For instance, take the output of a **:bash:‘uname‘** command in Linux, we'll use `Linux development 4.4.0-66-generic #87-Ubuntu SMP Fri Mar 3 15:29:05 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux` as an example. One could parse this (or, preferably, use the variety of **:bash:‘uname‘** flags that are available) and get several values from it. You may want to store them in a variable on hosts as something like:

```
{
    "hardware": {
        "architecture": "x86_64",
        ...
    },
    "os": {
        "details": "Linux development 4.4.0-66-generic #87-Ubuntu SMP Fri Mar 3␣
↪15:29:05 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux",
        "distribution": "Ubuntu",
        "kernel": {
            "type": "Linux",
            "version": "4.4.0-66-generic",
            ...
```

```
        }
    }
}
```

Inventory

Concepts

The fundamental unit of inventory in Craton is a **device**, which has the following characteristics:

Configurability A device is configurable, either directly over SSH with tooling like Ansible or indirectly via some controller.

Configuration can be further divided as follows:

- Version-controlled configuration, often captured in playbooks. It is the responsibility of workflows to use such configuration.
- Config data stored in the inventory schema.

In either case, the ultimate source may be manual, programmatic, or a combination of the two.

Addressability A device has an IP address (specifically the control plane IP address).

Hierarchy Devices are part of a hierarchy of regions and cells; a region in turn is part of a project.

Labels Devices may be arbitrarily labeled. Such labels can describe physical layout – cabinet, networking, power, etc – along with logical usage – compute service, etc. Labels are not in a hierarchy. (But should they be?)

Some systems like Kubernetes use a key/value scheme for labels. This can be readily supported by the convention `key:value` for the label name.

A **host** is a concrete subclass of device, and corresponds to the equivalent Ansible concept: a host is directly configurable by an Ansible playbook over SSH. Hosts have associated SSH keys for such access.

Principals interact (read and/or write) with inventory about devices. Governance for this interaction is mediated by RBAC and logged with respect to **change records** (including the reason for the change). Third party governance systems like OneOps can further structure these interactions. There are two classes of principals:

Workflows Capture audits about inventory; which are in turn used for any remediation. The pattern of usage is bottom-up.

Workflows are pluggable. They can be refined to a number of levels: Ansible, OpenStack Ansible, a specific workflow for managing patch levels with OSA (TODO does that actually make sense for OSA?).

Note that a workflow can be run any number of times, against different subsets of inventory. Example: migrate a specific cabinet, as specified by a label. Think of this distinction as being like an Ansible playbook vs its play.

Because workflows also know about version-controlled config, perhaps they can be used for queries as well. (TODO Ansible has some limited ways of determining such variables; it's possible OSA might develop this further as well in terms of role-based scheme.)

Users Configure and query inventory. The pattern of usage is top-down, whether that's configuring a specific label or drilling down from a given cell.

Users also can run workflows. This capability implies that workflows can be linked to roles; and that permissions include being the ability to run workflows.

Inventory interactions can be optionally logged. For example, if inventory is configured to use MySQL with InnoDB as its backing store, then all changes can be captured in the write-ahead log and reliably streamed to Kafka for analysis.

Associated with each region, cell, label, and device are **variables**. Here are some aspects of variables:

Description Variables are used to describe a device with config, auditing (possibly a subset of discovered facts), and other information. However variables do not store logging, metric, or monitoring information — because of volume, such storage is best done in a separate database, such as timeseries DB.

Key/value pairs Variables are made of up key/value pairs.

Key Keys are strings; they are additionally restricted to be valid Python identifiers. We usually refer to these as **top-level keys**, because values can be arbitrarily complex JSON values.

Such keys, and their prefixes, also serve as roles in Craton's implementation of RBAC. Keys are in a single namespace that does not differentiate between config or audit variables.

Value Values are of JSON type, and can store arbitrary data as such, including binary data via base64 encoding. Workflows define these specifics.

Scope resolution Variables use hierarchical scope to resolve for a specific device, using the following ordering:

1. Region
2. Cell
3. Label; if a device has multiple labels, the labels are sorted alphanumerically
4. Device

Such resolution overrides at the lowest defined level, which allows for variables to describe a device with the "broadest possible brush". Overrides do not merge values, even if the value has keys embedded in it.

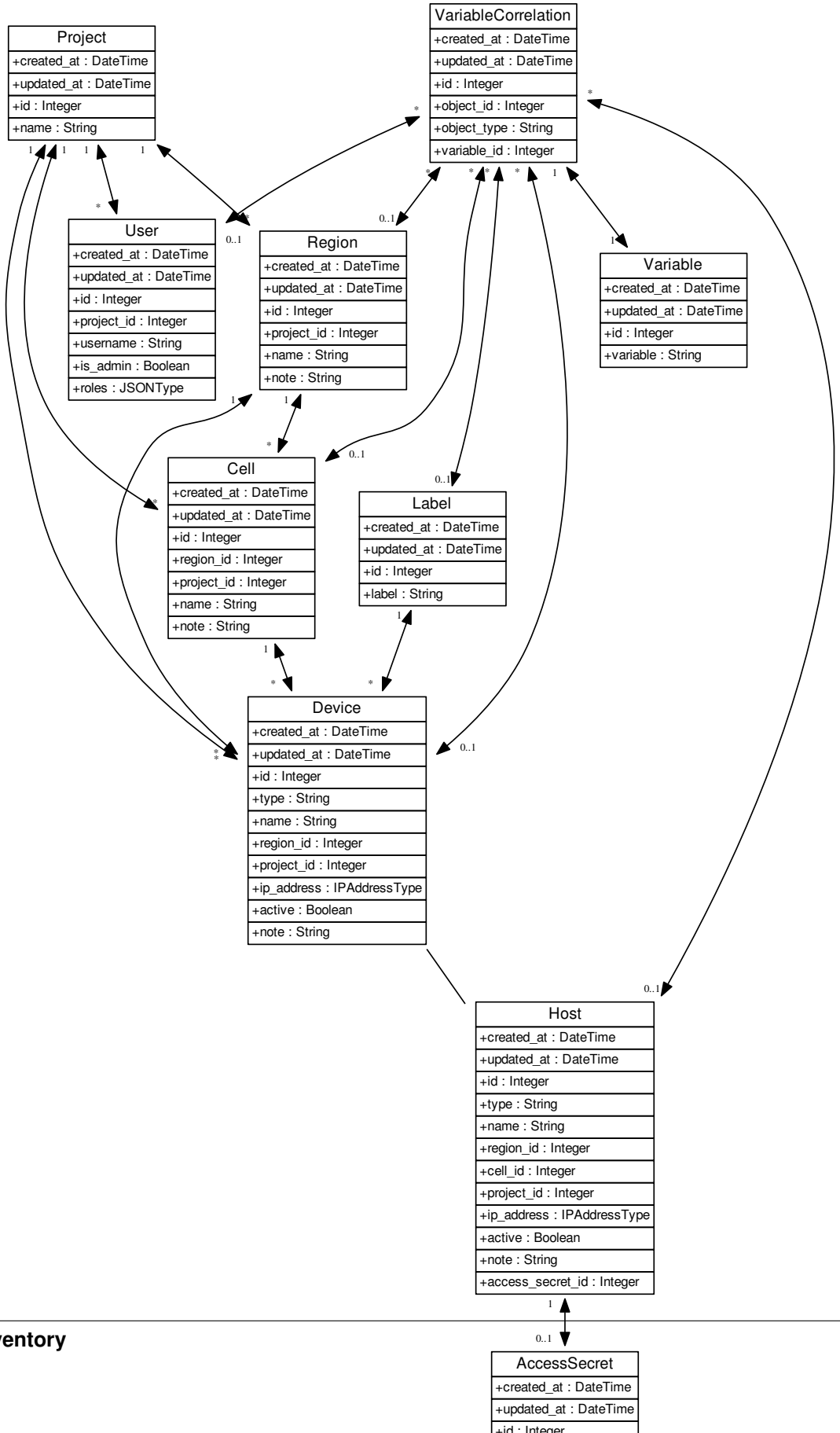
In general, config variables should be set at the highest possible level; whereas audit data should be bottom up from device.

Metadata Variables are also associated with the actor that wrote them, along with a record of change, including a note describing the change.

It may be desirable to track other metadata about a variable: is this intended for config, vs discovered from an audit? But note this might be just a question of which actor wrote this variable: was it a user? (Config.) Or was it a workflow? (Audit/remediation, possibly further identified by workflow metadata.)

Implementation

Craton's inventory is modeled using Python objects, which in turn has a concrete reference implementation using SQLAlchemy:


Project
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+name : String
VariableCorrelation
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+object_id : Integer
+object_type : String
+variable_id : Integer
User
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+project_id : Integer
+username : String
+is_admin : Boolean
+roles : JSONType
Region
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+project_id : Integer
+name : String
+note : String
Variable
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+variable : String
Cell
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+region_id : Integer
+project_id : Integer
+name : String
+note : String
Label
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+label : String
Device
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+type : String
+name : String
+region_id : Integer
+project_id : Integer
+ip_address : IPAddressType
+active : Boolean
+note : String
Host
+created_at : DateTime
+updated_at : DateTime
+id : Integer
+type : String
+name : String
+region_id : Integer
+cell_id : Integer
+project_id : Integer
+ip_address : IPAddressType
+active : Boolean
+note : String
+access_secret_id : Integer
AccessSecret
+created_at : DateTime
+updated_at : DateTime
+id : Integer

TODO(jimbaker): implementation of the inventory concepts is a work in progress, however, the above schema represents the current implementation. Notably missing are principals, including workflows and users, which will be added in the next phase of work.

CHAPTER 3

Indices and Tables

- genindex
- search

CHAPTER 4

License

Craton is licensed under the Apache license

Index

Symbols

C

D

H

P

R