Ralph Scrooge Documentation

Release 3.0.1

Allegro Group

Dec 05, 2018

Contents

Contents:

CHAPTER 1

Overview

Scrooge module brings billing functionality to Ralph. Using Ralph Scrooge, you can retrieve information on the use of resources from many other systems and charge other services. Scrooge generates flexible and accurate cost reports and let you refer to historical usages and costs.

Ralph Scrooge is a combination of IT management and accounting software. By using Scrooge your company can optimize the costs of internal services and departments, by reviewing their structure and dependencies.

Features:

- costs module, where you can provide invoice costs, staff costs and other costs (licences, supports etc.)
- many plugins to retrieve information on the use of resources from other systems
- charging other services proportionally to the use of resources by their devices
- many full-featured reports (costs per service, costs and usages per single device, dependency structure)
- OpenStack support (tenants, simple usages, ceilometer)
- pluginable architecture - write your own plugins to collect data and calculate costs of your services!
- API to upload information from other systems

CHAPTER 2

Installation

Scrooge contains ralph in its requirements, because it is a plugin for ralph. For more information on how to configure or install ralph, please refer to its documentation.

2.1 Install Scrooge

There are two ways to install Scrooge. One of them is a simple pip installation which is nice and easy. Installation from sources require Scrooge to be downloaded from github and then, installed manually

2.1.1 Install Scrooge from pip

A fast and easy way is to install Scrooge from pip:

```
(scrooge_env)$ pip install scrooge
```

That's it.

2.1.2 Install Scrooge from sources

It is also possible to install Scrooge from sources. To do this, first, you need to download Scrooge from github:

```
(scrooge_env)$ git clone git://github.com/allegro/ralph_pricing.git ralph_scrooge
```

Enter to the project folder:

```
(scrooge_env)$ cd ralph_scrooge
```

and install it:

```
(scrooge_env)$ pip install -e .
```

The Scrooge requirements will be installed automatically

2.2 Upgrade existing installation

To upgrade Scrooge, you need to stop any Scrooge processes that are running. It is good practice not to upgrade the old version, but create a separate virtual environment and install everything from the begin, but if you need to upgrade the old version, be sure that everything is stopped.

2.2.1 Upgrade Scrooge from pip

If you installed Scrooge from pip, then you can simply:

```
(scrooge_env)$ pip install --upgrade scrooge
```

After it is finished, upgrade the static files:

```
(scrooge_env)$ scrooge collectstatic
```

2.2.2 Upgrade Scrooge from sources

First, you need to download Scrooge from github:

```
(scrooge_env)$ git clone git://github.com/allegro/ralph_pricing.git
```

Enter to the project folder:

```
(scrooge_env)$ cd ralph_scrooge
```

and upgrade it:

```
(scrooge_env)$ pip install --upgrade -e .
```

Finally, you need to upgrade the static files:

```
(scrooge_env)$ scrooge collectstatic
```

2.3 Migrate the database

Some of updates require database migrations. To migrate a database, you need to run:

```
(scrooge_env)$ scrooge migrate ralph_scrooge
```

Be sure that you have a backup of your database. Sometimes you can migrate data or create some complicated and unwanted changes.

2.4 Update the settings

Some new features added to Ralph may require additional settings to work properly. In order to enable them in your settings, follow the instructions in the *change log* for the version you have installed.

2.5 Testing if it works

To be sure that everything work fine, is recommended to run unit tests. To do this, run:

```
(scrooge_env)$ test_scrooge test
```

CHAPTER 3

Configuration

Configuration is available in file settings.py and there you should implement your settings.

PLUGGABLE_APPS <list of strings> - List with available applications for ralph

```
PLUGGABLE_APPS += ['scrooge']
```

VIRTUAL_VENTURE_NAMES <list of strings> - venture names for virtual devices

```
VIRTUAL_VENTURE_NAMES = ['venture1', 'venture2']
```

SCROOGE_SENTRY_DSN <string> - Full address with API key to sentry

```
SCROOGE_SENTRY_DSN = 'http://xxxxxx:yyyyyy@sentry/zz'
```

CURRENCY <string> - This currency will be added to each value on report. It is prefix to cost value.

```
CURRENCY = 'PLN'
```

HAMSTER_API_URL <string> - Url to hamster API

```
HAMSTER_API_URL = 'http://xxxxxxx/'
```

RQ_QUEUE_LIST <tuple of strings> - List of queue names

```
RQ_QUEUE_LIST += ('reports', 'reports_pricing')
```

SSH_NFSEN_CREDENTIALS <dict> - Credentials for servers

```
SSH_NFSEN_CREDENTIALS = {
    'xxx.xxx.xxx.xxx': {
        'login': 'xxx',
        'password': 'xxx',
    },
    'yyy.yyy.yyy.yyy': {
```

(continues on next page)

(continued from previous page)

```
        'login': 'yyy',
        'password': 'yyy',
    },
}
```

NFSEN_CHANNELS <list of strings> - Channels like IN or OUT

```
NFSEN_CHANNELS = ['xxx-OUT', 'xxx-IN', 'yyy-OUT', 'yyy-IN']
```

NFSEN_FILES_PATH <string> - Path to nfsen data files on remote server

```
NFSEN_FILES_PATH = 'xxx/yyy/zzz'
```

NFSEN_CLASS_ADDRESS <list of strings> - Available class addresses

```
NFSEN_CLASS_ADDRESS = [
    'xxx.xxx.xxx.x/yy'
    'zzz.zzz.zzz.z/yy'
]
```

OPENSTACK_USER <string> - User login name to openstack

```
OPENSTACK_USER = 'xxx'
```

OPENSTACK_PASSWORD <string> - User password for given user name

```
OPENSTACK_PASSWORD = 'yyy'
```

OPENSTACK_URL <string> - Url to openstack

```
OPENSTACK_URL = 'yyy'
```

OPENSTACK_REGIONS <list of strings> - Datacenter names

```
OPENSTACK_REGIONS = ['xxx', 'yyy']
```

OPENSTACK_EXTRA_QUERIES <list of tuple> - Extra queries for openstack

```
OPENSTACK_EXTRA_QUERIES = [('http://xxx', 'yyy'), ('http://zzz', 'aaa')]
```

SCALEME_API_URL <string> - Url to scaleme

```
SCALEME_API_URL = 'http://xxxxxxx/'
```

SPLUNK_HOST <string> - Splunk host name

```
SPLUNK_HOST = 'http://xxxxxxx/'
```

SPLUNK_USER <string> - Splunk user name

```
SPLUNK_USER = 'xxx'
```

SPLUNK_PASSWORD <string> - Password for splunk user

```
SPLUNK_PASSWORD = 'yyy'
```

CHAPTER 4

Reports

There are a few report types. The main one is called "Ventures report" being a billing report. Scrooge also contains other reports supporting the main report. For example, one of them shows venture usages for each day. The other one shows devices containing a selected venture. As you probably know, cost columns are not always enough as people want to know how, why and where the figures come from. One of the solutions is to create one huuuuge report with many columns and nested tables, but do you think it is will be clear enough for us? In my opinion, an experienced accountant may have a problem to analyse large statements and each mistake can be dangerous for strategic or tactical business plans. The other solution is to create categorized reports – in such case, we can use a few Scrooge reports.

To understand this section better, use a simple info card:

```
Target:
    Description
Actions:
    - First action
    - Second Action
    - ...
```

4.1 Ventures report

```
Target:
    Show total cost of each venture
Actions:
    - Generate financial billing
    - Create Statement
    - Download as CSV
```

You can have the main report, statement of costs or just a financial billing if you wish. Here, you will see all the costs for each venture. Columns contain the total cost of base usage, service usages and extra costs. Additionally, each service cost contains more details about each usage. Details are shown as count and cost column.

This view also gives you a possibility to generate a csv file and create a statement of report options. If you click Create statement or Download csv before the update, a report will be generated, so you don't need to generate a report before creating a statement to download it to csv.

4.2 Devices report

```
Target:
    Show all devices and their cost for single venture
Actions:
    - Generate statement of devices for single venture
    - Create Statement
    - Download as CSV
```

This report shows you details for a single venture. This solution is used when someone comes to you and asks why his costs are so high. Then, you can generate a device report to show which devices are assigned to him and how much they cost. It is very important for big huge companies with e.g. one thousand ventures.

4.3 Ventures daily usages

```
Target:
    Show usage count differences between days
Actions:
    - Generate usages count differences report
    - Create Statement
    - Download as CSV
```

This report is logically linked with device ventures changes, but we have separated it, because the two reports would look like a board with undefined numbers.

As a result, the ventures daily usages report shows you only numbers of changed usages in a selected time interval for all ventures. There are no details on which device was transfereed between two or more ventures. This is only comp of changes in our system and if you see any warnings or errors in changes and you need to verify what was changed, then you need to use the device ventures changes report.

4.4 Device ventures changes

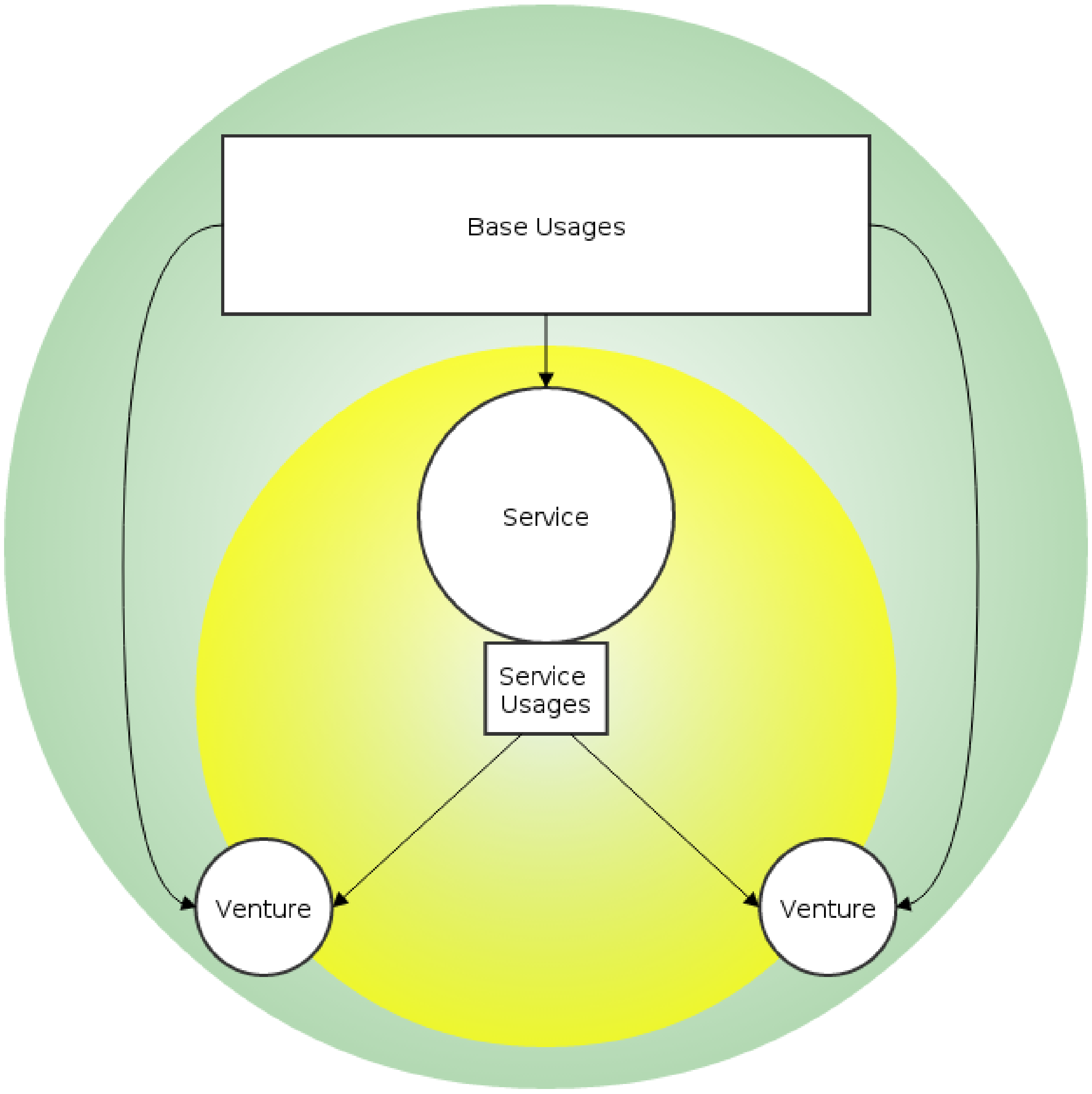
```
Target:
    Show information about transferring devices between ventures
Actions:
    - Generate device transfer report
    - Create Statement
    - Download as CSV
```

This report shows how devices were transferred between ventures. The device ventures changes report is a supplement to the ventures daily usages. It is really useful when ventures contain many devices (for example two thousand) and many devices often change their owners. This report shows you change details for a selected date.

CHAPTER 5

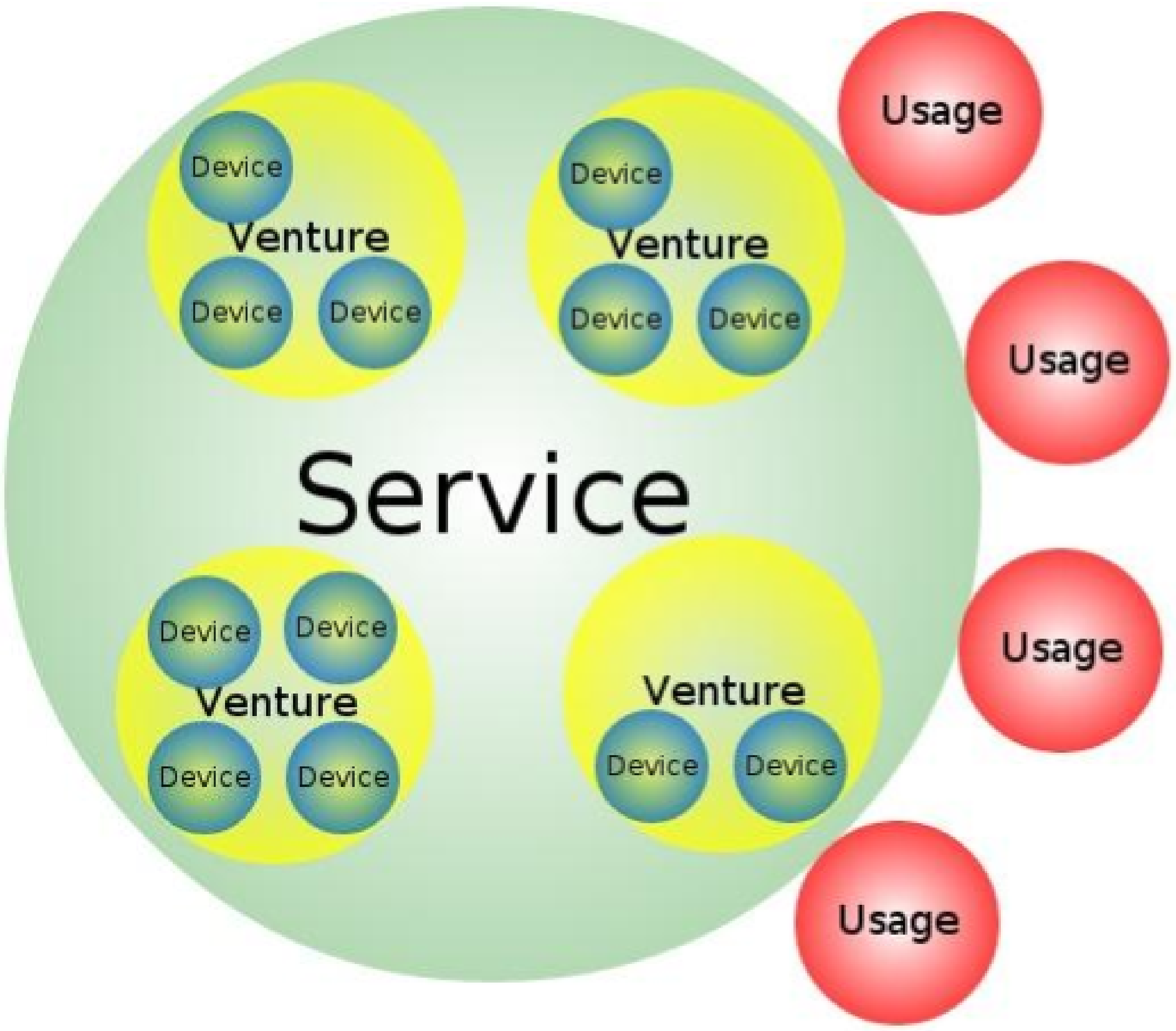
Services & Usages

The billing model implemented in Scrooge is based on services and usages. We can distinguish a service with service usages and base usages. Base usages are used to define a cost of venture/service for hardware. A service is a “bag” used to group many ventures (similarly, ventures are a bag used to group many devices). Service usage is used to calculate how much each client (ventures that used this service) should pay for using the service.



5.1 Services

A venture is the base for calculating costs. Each venture contains a few devices and can use many services. A service is a bag to group many ventures. For example, when we create a service called Datastorage, we can include the following ventures: datastorage_test and datastorage_production to create a “bag” called Datastorage. The costs of datastorage_test and datastorage_production ventures are now included in the total costs of service called Datastorage. Now, we have two more ventures called webserviceA and webserviceB which use our Datastorage service (when I say “use Datastorage”, I mean they use some devices from datastorage_test and datastorage_production ventures).

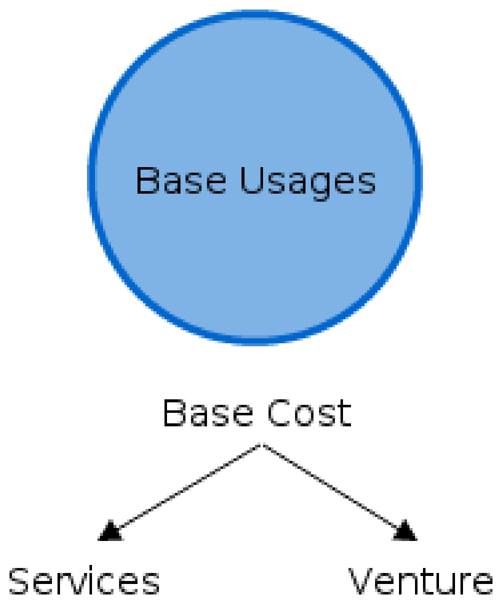


5.2 Usages

Usages contain basic and low-level values required by Scrooge to calculate and create a report.

5.2.1 Base usages

The usage give us information on how much a device costs. As you can imagine, every device in the datacenter uses power and probably internet, as well as space, and requires people to take care of it. Every device has also its purchase cost. To distribute costs, each device must contain information about the energy it consumes and space it occupies, which are its base usages defining how many units each device takes. To calculate the cost of a device, for example electricity cost, energy consumption units are counted to one value and divided by the electricity cost. Then, we have a price per one unit and when we multiply this price by the number of units for a single device we will have a price for this device.



Power consumption

Collect: *Each device model contains information about power consumption entered manually. The collect plugin collects this information every day and saves it as a simple "power consumption" abstract unit.*

Cost: *Accountants enter invoice costs manually.*

Report: *Cost of one unit is calculated by dividing total usages and total cost. The plugin sums all usages per venture and multiplies it by the cost of one unit.*

Height of device

Collect: *This base usage is collected exactly in the same way like power consumption.*

Cost: *Accountants enter invoice costs manually.*

Report: *This base usage cost per venture is calculated exactly in the same way like power consumption.*

Network

Collect: : *Usages are collected from nfsen servers per IP. Information about IT matching to Venture is collected from ralph. The plugin matches IP with Venture and save usages from nfsen as a venture usage. Unknown IPs are matched with the venture from settings.*

Cost: *Accountants enter invoice costs manually.*

Report: *This base usage cost per venture is calculated exactly in the same way like power consumption.*

Depreciation

Collect: *Each asset contains information about costs (more specifically, it is a device cost). The collect plugin collects this information as a daily device imprint.*

Cost: *Logistics specialists enter device (asset) cost manually.*

Report: *The cost for one day is calculated by dividing the device total cost and depreciation days which are calculated based on depreciation rate. The plugin sums all the costs per one day from all the devices and multiplies it by the number of days for which the report is generated.*

Teamwork

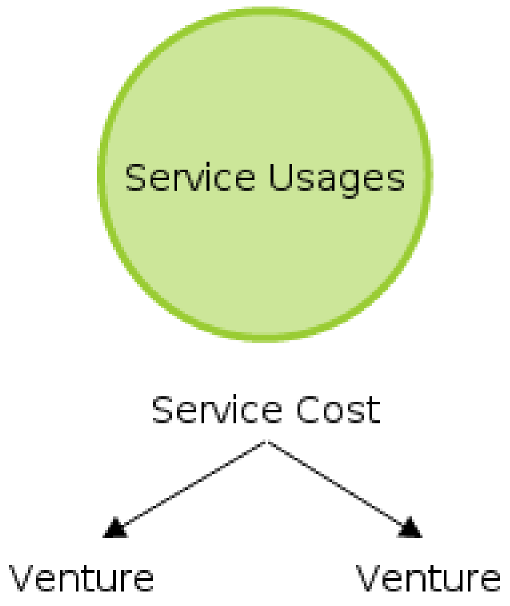
Collect: *Teams are based on collected data, so there is no any additional logic for it.*

Cost: *Accountants enter cost and teams number manually.*

Report: There are 4 ways to calculate teamwork costs.

1. By time
2. By devices and cores count
3. By devices
4. By cost distributed to other teams proportionally to team members count

5.2.2 Service usages



This type of usage is used to distribute service costs to ventures. Every day each service has to supply information to Scrooge on how it was used by each venture. Before supplying such information, service and usages must be defined together with the link from the admin panel. You can find more about how to create services and usages in the How to use section.

In the report, services are represented in columns with count and usage cost and one additional column with the total cost of service. The most important is “to know” that the service cost is the same as the total cost of ventures included in the service. A service is a mechanism to distribute costs of few ventures to other ventures.

One of big wins of Scrooge is its automatic plugin which gives you a possibility to add your service to our system without creating any code line. This plugin uses data that was supplied, services and usages that were created and linked, and returns complete biling information. Of course, if you need to present your data in your own style, you can try to create a dedicated plugin or do it yourself, but it is not recommended.

CHAPTER 6

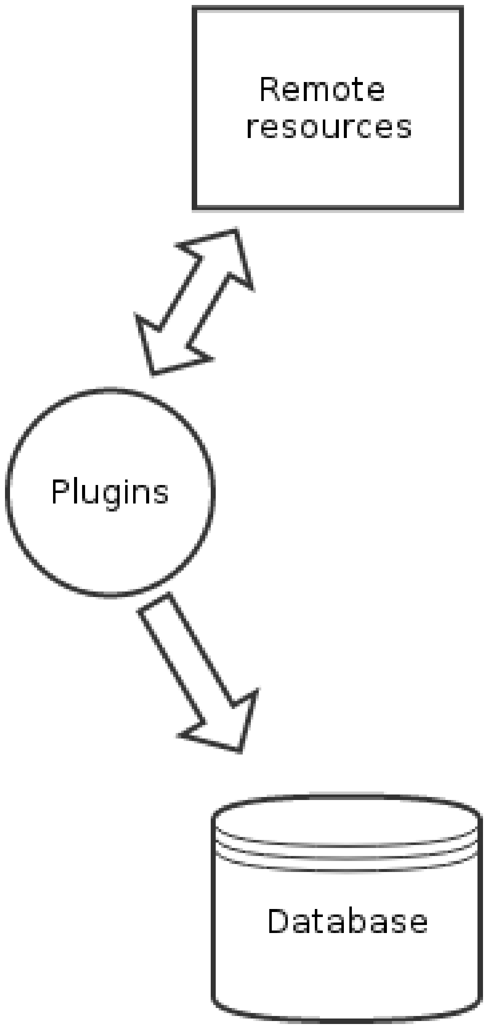
Plugins

There are two types of plugins. One of them is used to collect data from remote applications or servers; the other one called “report” is used to present results in the report.

The main goal of plugins is to create a universal and configurable logic for each new service and handle non-standard cases where the collect or/and report plugin contains more complicated logic and cannot be handled by a universal plugin. For example, if it is not possible to send usages data to Scrooge, you can go to Scrooge administrators and try to ask them to develop the collect plugin for you. This is not recommended, but possible (it is preferred to use push API). The second situation is when your service requires pre-processing before displaying results or contains a really complicated logic – you can also go to Scrooge administrators and ask them for help to create your own plugin, and again you can try to ask them to develop a report plugin for you. It is recommended to use a universal plugin and adjust your account logic to The Scrooge logic model.

Plugins use queues. Before it is run, every plugin is queued and waits for its turn.

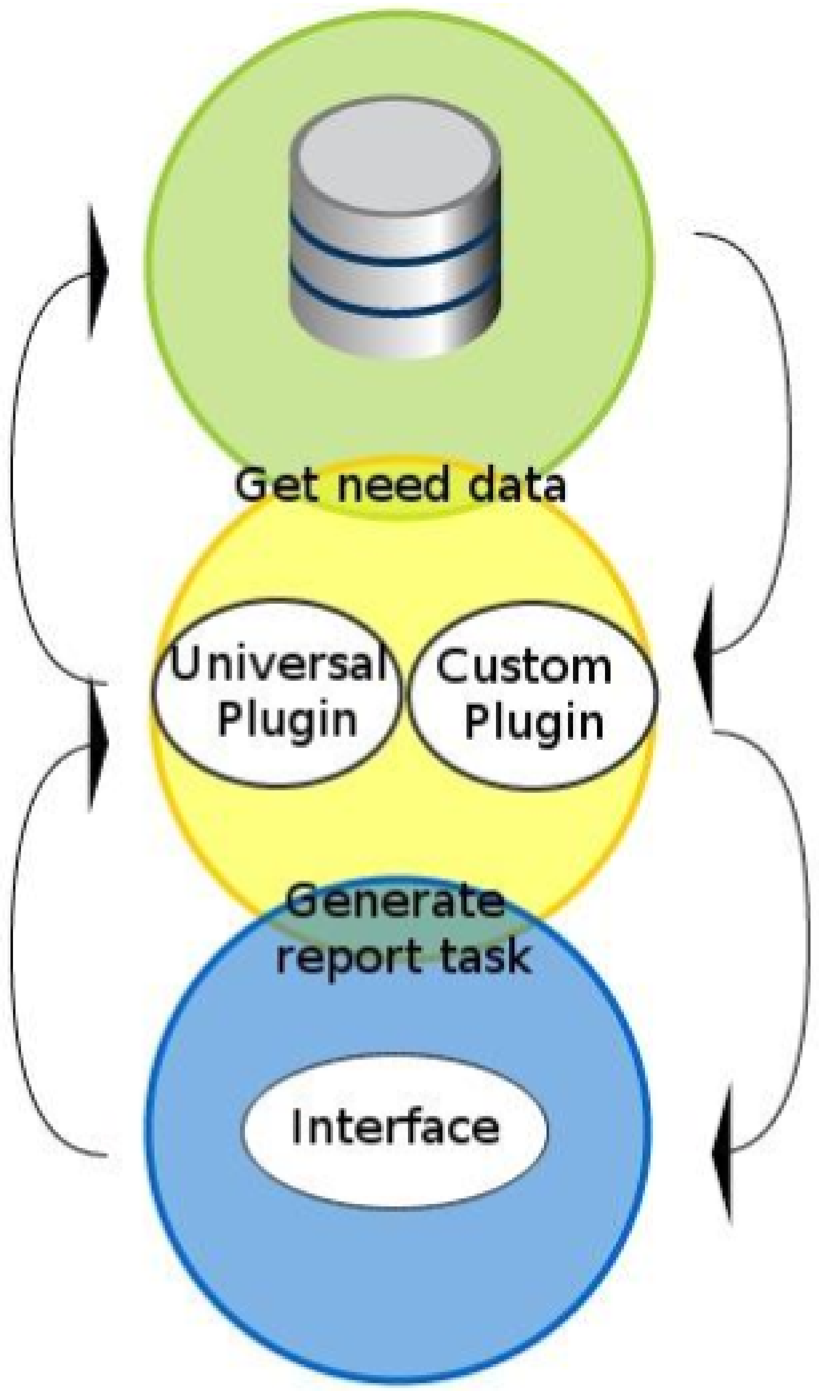
6.1 Collect plugins



Collects plugins are used to collect data from remote sources. More specifically, collect plugins are used to collect base usages. Collect service usages are not advisable, so create plugin for service usages is finality.

Each plugin is launched, for example, at 9 am and collects data all day. There is no base plugin etc. Each collect plugin contains a custom logic and as a result of its work, data is added/updated in the Scrooge database.

6.2 Report plugins



Report plugins are used to generate columns in the report. Each plugin must return 2 pieces of information, column scheme and column value per venture. This is important because the column scheme is used to define and create additional columns in the report, but column value fill this columns for each venture. If a venture has no value for any column, 0.0 will be set as default. For default cases, we have created a universal plugin, so each service can use it when:

1. service was created
2. all usage types for this service were defined
3. service contains usage types
4. service contains percentage for each selected usage. (100 for only 1 usage)
5. service contains selected base usages
6. service contains assigned ventures
7. usages were provided for each day.

Service and usages are configurable parts, so they can be selected from the Scrooge admin panel. For more information on how to do it, refer to the Services & Usages section.

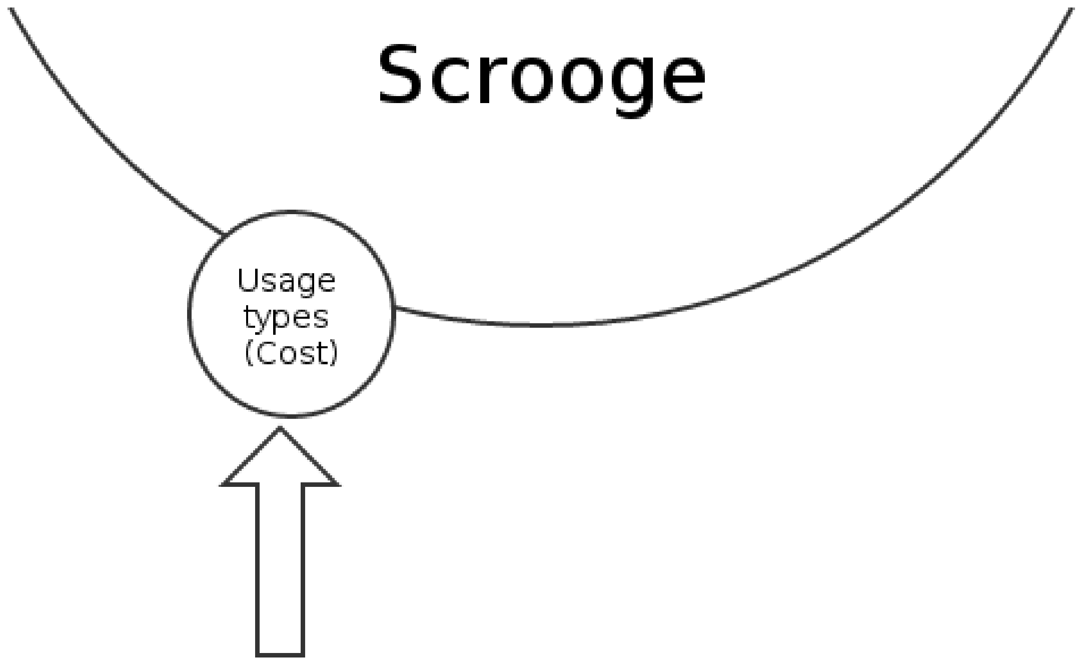
Report plugins used to calculate service costs require the cost of base usages. Base usages without costs are excluded from the total base service cost, so data from the report will be incorrect. More information about costs can be found in the Cost & Extra Costs section.

CHAPTER 7

Costs & Extra costs

To calculate the total cost of venture/usage, we need to add two types of costs. The most important and the first of them is the base usages cost. Base costs are used to calculate the actual (real) cost of ventures or services, so when any of the base costs is not entered or any cost is not correct, the real cost of venture is distorted. The other costs are called extra costs with additional costs of e.g. a software license. The costs and extra costs should not be changed for the past period.

7.1 Costs



This type of costs is used to calculate real costs of usages and ventures. Usually, we have invoice costs, but it is not a rule. When base costs are not entered correctly, all of the calculations are distored (and this is the most important thing to remember). You should also remember about continuity of dates. For example, if you enter an invoice cost from April, you should enter the date from 2014-04-01 to 2014-04-30 and when you enter an invoice cost from May, you should start from 2014-05-01. You may see two error types for costs:

No price It appears when the base usage price is not entered at all for a given time period for a report. For example: someone has forgotten to enter the invoice cost from May.

Incomplete price It appears when a given price has no continuity of dates for a given time period for areport. For example: the cost for April is from 2014-04-01 to 2014-04-20 and the second one is from 2014-04-22 to 2014-04-30. There is no price for 2014-04-21.

These errors will be visible in the report in appropriate cells.

7.2 Extra costs

Extra costs were created for additional costs such as a software license. Extra costs are used to provide any unconventional costs to the report and to the total cost of services. It is important to enter the costs convention. Each extra cost entered to Scrooge is a monthly cost, but the algorithm calculates a daily cost and generates a daily print-out. Let's analyze the following story:

In May the Internal Solutions Department ordered 10 new keyboards for $3,000. For smooth administration, the cost of keyboards started to be calculated on 1 June, so the daily cost of this order is $100 ($3,000/30 days) and $10 per each keyboard. At the same time, The Python Department ordered 20 keyboards for $3,000 (daily cost: 3,000/30=100), but this time each keyboard costs only $5. On 15 June Python software developers made a deal with people from internal solutions to exchange 5 cheaper keyboards for 5 expensive keyboards. As a result, what is the monthly statemenet for the orders of these two departments ?

```
Internal Solutions Department:

1-15 June = $100 * 15 days = $1500                    # daily cost = $100
16-30 June = $50 * 15 days + $25 * 15 days = $1125    # daily cost = $75
Total cost = $2625

Python programmers:

1-15 June = $100 * 15 days = $1500                    # daily cost = $100
16-30 June = $75 * $15 days + $50 * 15 days = $1875   # daily cost = $125
Total cost = $3375
```

Therefore, based on the current monthly cost, the daily cost of each extra cost is calculated and printed. At the beginning (before the exchange), the daily cost was the same (each of departments bought products for $3,000) but after the exchange the daily cost was changed, because Python software developers received 5 expensive keybords and gave away 5 cheaper keyboards and vice versa for the Internal Solutions Department.

CHAPTER 8

REST API

Scrooge provides public REST API described by OpenAPI-compliant schema, which is available for consumption by external programs/clients at `/scrooge/api/schema.json`.

On top of this schema, we provide swagger-ui , which is available at `/scrooge/api` or `/api` endpoint (the latter just redirects to the former). It serves both as documentation and convenience tool, that is meant to facilitate human interaction with our API and improve its discoverability.

8.1 Notes for Scrooge developers/administrators

API schema is defined in file pointed by `API_SCHEMA_FILE` setting (by default `ralph_scrooge.media.api_schema.yaml`). This file, in YAML format is meant for editing and feeding `swagger-ui` during development. For production though, you should generate a JSON file from it. There's a management command `scrooge generate_json_schema`, which is meant exactly for that. So basically, YAML file is a source file, while JSON should be seen as kind of artifact, which is not meant to be edited directly (it is minified BTW), and shouldn't be kept in Scrooge's repo.

There is one important setting related to API schema - `SCROOGE_HOST` - which should be filled if you want to provide Scrooge's REST API schema to your clients (and swagger-ui is one of them) - see the comment in `ralph_scrooge.settings.base` for additional info on this.

CHAPTER 9

Statements

When we start talking about financials and accounting, we may come across definitions of creative accounting, provisions, adjustments or acceptable deviation, but the one think is really stable and unchanged “History”. Scrooge provides an opportunity to create historical statements manually. You may just create a report for the past date but to be sure that the historical raport will be not changed, you may create a statement.

The logic for creating statements is as simple as possible. A report is converted to JSON and saved in the database. JSON cannot be changed by users, so you will be sure that your historical report will be saved and unchanged forever. This simple case protects us from human mistakes like changing an invoice cost for the past date or rebuilding/upgrading custom plugins. This is a really important case when the system is flexible and allows you to create a custom and dedicated logic.

CHAPTER 10

Change Log

10.1 DEV

- [data-migration] changed Scrooge warehouses source to assets datacenters (from assets warehouses)

10.2 3.0.1

Released on April 23, 2015

- Added versioning to static files.
- Added database and Virtual IP as default components.
- Change default settings for collect plugins.
- **Fixes:**
  - Unification of menu items in angular views and old django views.
  - Removed unnecessary ‘Extra costs’ and ‘Usage types’ menu items.

10.3 3.0.0

Released on April 13, 2015

- Project name changed from Ralph Pricing to Ralph Scrooge
- **Redesigned architecure comparing to (old) ralph_pricing**
  - Calculation based on services and environments instead of ventures
  - Base PricingObject model to simplify adding new types of chared objects (ex. Database, Virtual server, Tenant)

  - Costs are (re)calculated and stored in database
  - Many performance improvements
- **New client GUI written in AngularJS**
  - Components: preview of historical objects (server, virtual, database etc) per service for single day
  - Costs card: summary of service costs in single month
  - Allocations: add service or team specific costs and manage it's distribution to other services
  - Costs: detailed costs for each pricing object
- **New charging types:**
  - Dynamic extra costs: specify cost (like Extra cost) and a dynamic way of it's distribution (ex. cores count)
- **New collect plugins:**
  - Collecting Database, VIP, Tenant info from Ralph
  - OpenStack SimpleUsage plugin
  - OpenStack Ceilometer MongoDB plugin
  - Support plugins (from Ralph Assets)
- **New permissions:**
  - Every (active) Ralph user has access to client part (components, allocation, cost card) for services which he owns
  - Admin (ralph_scrooge group) has access to whole system

10.4 2.7.0

- Adjusted ralph_pricing to work in parallel with ralph_scrooge

10.5 2.6.0

- Added regular usages to service
- Rounding value on report
- Added yeasterday flag to pricing_sync
- Added new extracost model

10.6 2.5.4

- Fixed team average with ventures subset

10.7 2.5.3

- Added exclude ventures for base usages
- Added ventures filter and forecast price to ceilometer report plugin
- Changed profit center field length from 75 to 255

10.8 2.5.2

- Logs compatible with python 2.7.3

10.9 2.5.1

- Improved services usages api logging
- Fixed fixed header scroll in reports
- Added DirectoryTimedRotatingFileHandler
- Removed sentry workaround
- Improved default loggers and config
- Separated unknown ventures for shares groups

10.10 2.5.0

- Fixed decimal precision in tests
- Remove back collecting disk share mount
- Fixed report error log text
- Added average team billing model
- Added share multiple groups
- Added san collect plugin
- Added exclude ventures to teams
- Added required permisions to view scrooge
- Renamed package to ralph_scrooge
- Added coverals
- Fixed venture hierarchy, when venture have no parent then venture parent is None
- Added html documentation

10.11 2.4.0

- New devices report
- Devices ventures changes report
- New ceilometer report plugin logic and logging tweakups
- Fixed asset collect plugin (replacing to None)
- Ceilometer collect plugin bugfixes
- Added venture tree rebuild when venture plugin job is finished
- Fixed extra costs - add more than 5 rows (with dynamic adding)
- Fixed header in csv statement
- Improved gitignore and manifest
- When venture have no parent set venture parent as none

10.12 2.3.0

- Fixed report table header on scroll.
- Exception instan error in logging on report plugin run.
- Fixed raise exception 0/0 by team plugins.
- Added extra costs to report as separated column and service to total cost.
- Fixed saving device_id, sn and barcode
- Added monthly statement
- Added plugin to bill cloud 1.0 from ralph
- Fixed ventures daily usages header colspan

10.13 2.2.3

- nfdump get only ips from given network.
- Changed logging to logger in network plugin.
- Only usage types wtih is_manually_type flag are show in menu.
- Fixed calculating price. Massage incomplete_price was incorrect sometime.
- Fixed percent rounding for teams.
- Remove PLN from fields and add it to name of column.
- Average option for usages is now available.
- Fixed is_blade. Now it is truly boolean value.
- Added overwriting in push API.
- Added ventures daily usages report.
- Fixed usages columns width.

10.14 2.2.2

- Fixed nfdump_str, executed command on remote server.
- Added console statistics

10.15 2.2.1

- Upgrade ceilometer collect plugins.
- Added ceilometer report plugin.
- Fixed overwriting configuration by pluggableaps.
- Fixed logging from collect plugins. Now, when venture does not exist log warning.
- Upgrade inserting teams usages. Added total prcent information and button to dynamically add more rows.
- Plugins indentify usages only by symbols. Name and more options are set as defaults.
- Added multiple ventures option for single virtual server usages. settings.VIRTUAL_VENTURE_NAMES must be dict where key is name of groub and value is list of ventures.
- Network cost is by providers.
- Remove teams count table and added count to usage price table.

10.16 2.2.0

- Displayed name changed from Ralph Pricing to Scrooge.
- Added service model and plugin for billing service depending on it's usage types, base usage types and dependent services.
- Change report plugins architecture (change from function to classes, create plugin for base usages (eg. power consumption) and dedicated plugin for depreciation).
- Added teams billing. Teams could be billed in 4 models: by time, by devices count, by devices and cores count or by cost distribution between other teams depending on other teams members count.
- Modified collects virtual plugin for getting usages for more than one virtual systems.
- Created plugin for colleting internet usages per IP address (using nfsen).
- Added height of device usage.
- Removed old AllVentures report and warehouse option from report.
- PUSH API for usages of service resources by ventures.
- New white theme.

10.17 2.1.1

- Added scrooge logger sentry

10.18 2.1.0

- Changes in the architecture. Generate report from plugins for each usage
- Create few plugins for each usage
- Distinguish two groups of plugins, reports and collections
- Rebuild generate reports view and add it as beta venture view
- New report contains separated columns for warehouses for one report
- Increased efficiency of report generation
- Fix splunk plugin
- Used pluggableapps for scrooge config
- Added more logs from logger
- Added separated logger for scrooge
- Openstack ceilometer plugin
- When usage is per warehouse then warehouse must be chosen
- Fix datepicker on report subpage
- Added flag to hide/show usages on report
- Remove TopVenture subpage

10.19 2.0.1

- If assets plugin cannot find device by asset_id then try get device by sn

10.20 2.0.0

- Changes in the architecture. Now devices are taken by asset plugin from assets
- Remove device and cores plugin (This this is a role of asset plugin)
- DailyUsage contains warehouse field
- Added version of usage type price based on cost
- Added price or cost per warehouse
- Now venture reports are generated per warehouse (only colums with flag by_warehouse are different between reports)
- Added forecast prices and costs and possibility to generate forecast reports
- Added cost to price converter used by ‘get_assets_count_price_cost’ method

10.21 1.2.8

Released on December 11, 2013

- F5 devices billing added.

10.22 1.2.7

Released on November 03, 2013

- Added search boxes, filters and additional columns in admin.
- Fixed corner-case bug related to calculation of bladesystems costs.

10.23 1.2.6

Released on August 08, 2013

- Added "show only active" option in the reports
- Added short descriptions to reports templates
- Fixed assets plugin - IntegrityError protection, added new tests
- Show extra costs in the extra costs types admin

10.24 1.0.0

- initial release

CHAPTER 11

License

```
Copyright 2013-2014 Allegro Group

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```