Rally Documentation

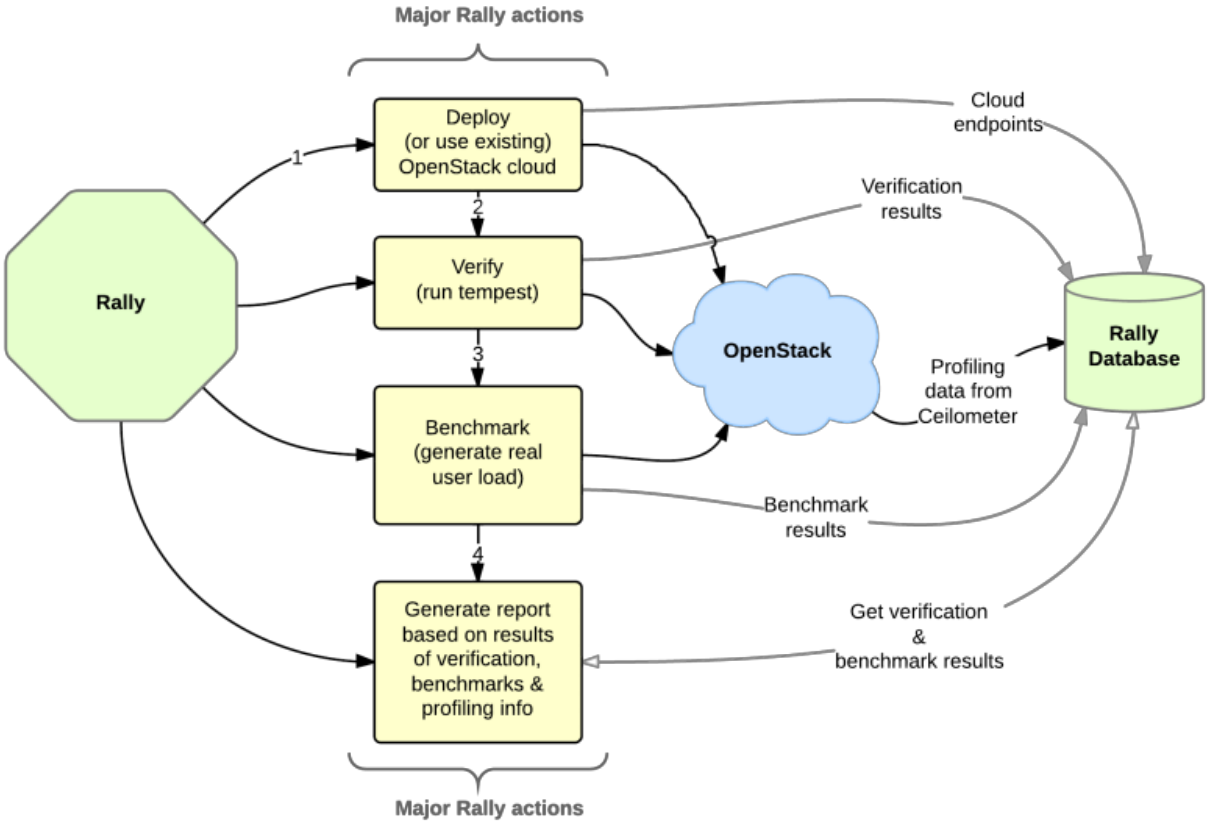
Release 0.0.4

OpenStack Foundation

November 17, 2015

1 Contents **3**
1.1 Overview . . . 3
1.2 Installation . . . 9
1.3 Rally step-by-step . . . 11
1.4 User stories . . . 38
1.5 Rally Plugins . . . 43
1.6 Rally Plugins Reference . . . 50
1.7 Contribute to Rally . . . 119
1.8 Rally OS Gates . . . 121
1.9 Request New Features . . . 124
1.10 Project Info . . . 129
1.11 Release Notes . . . 131

OpenStack is, undoubtedly, a really *huge* ecosystem of cooperative services. **Rally** is a **benchmarking tool** that answers the question: **"How does OpenStack work at scale?"**. To make this possible, Rally **automates** and **unifies** multi-node OpenStack deployment, cloud verification, benchmarking & profiling. Rally does it in a **generic** way, making it possible to check whether OpenStack is going to work well on, say, a 1k-servers installation under high load. Thus it can be used as a basic tool for an *OpenStack CI/CD system* that would continuously improve its SLA, performance and stability.



CHAPTER 1

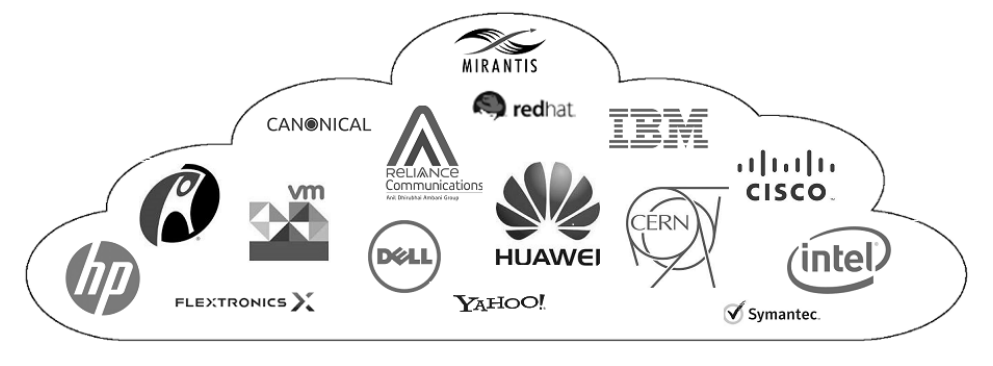
Contents

1.1 Overview

Rally is a **benchmarking tool** that **automates** and **unifies** multi-node OpenStack deployment, cloud verification, benchmarking & profiling. It can be used as a basic tool for an *OpenStack CI/CD system* that would continuously improve its SLA, performance and stability.

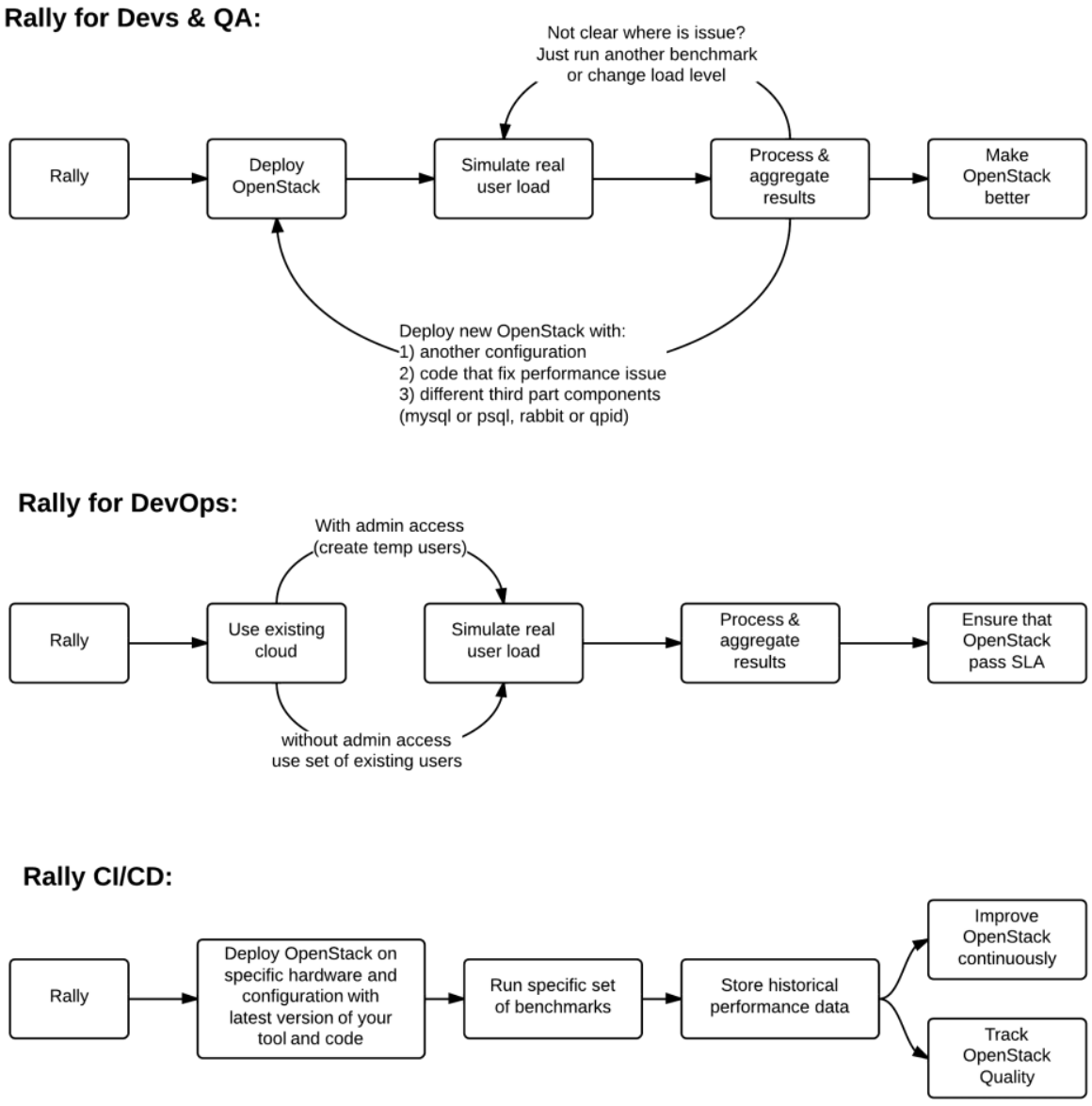
1.1.1 Who Is Using Rally

Here's a small selection of some of the many companies using Rally:



1.1.2 Use Cases

Let's take a look at 3 major high level Use Cases of Rally:



Generally, there are a few typical cases where Rally proves to be of great use:

1. Automate measuring & profiling focused on how new code changes affect the OS performance;
2. Using Rally profiler to detect scaling & performance issues;
3. Investigate how different deployments affect the OS performance:
   - Find the set of suitable OpenStack deployment architectures;
   - Create deployment specifications for different loads (amount of controllers, swift nodes, etc.);
4. Automate the search for hardware best suited for particular OpenStack cloud;

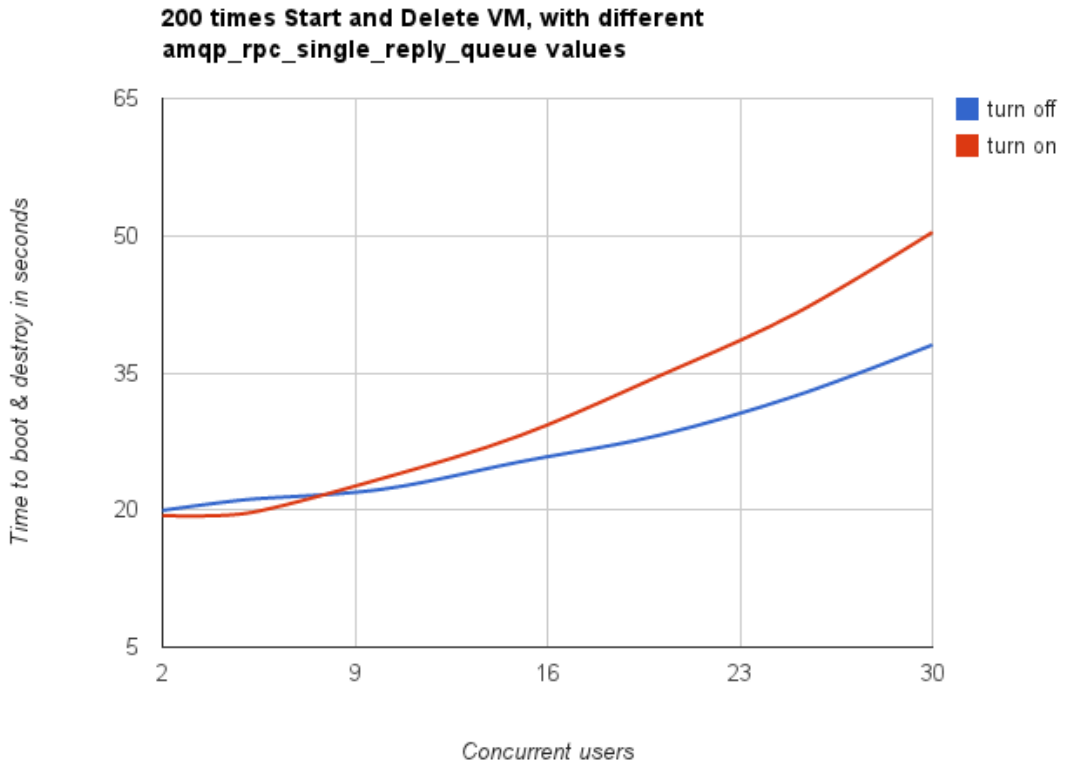
5. Automate the production cloud specification generation:
    - Determine terminal loads for basic cloud operations: VM start & stop, Block Device create/destroy & various OpenStack API methods;
    - Check performance of basic cloud operations in case of different loads.

1.1.3 Real-life examples

To be substantive, let's investigate a couple of real-life examples of Rally in action.

How does amqp_rpc_single_reply_queue affect performance?

Rally allowed us to reveal a quite an interesting fact about **Nova**. We used *NovaServers.boot_and_delete* benchmark scenario to see how the *amqp_rpc_single_reply_queue* option affects VM bootup time (it turns on a kind of fast RPC). Some time ago it was shown that cloud performance can be boosted by setting it on, so we naturally decided to check this result with Rally. To make this test, we issued requests for booting and deleting VMs for a number of concurrent users ranging from 1 to 30 with and without the investigated option. For each group of users, a total number of 200 requests was issued. Averaged time per request is shown below:



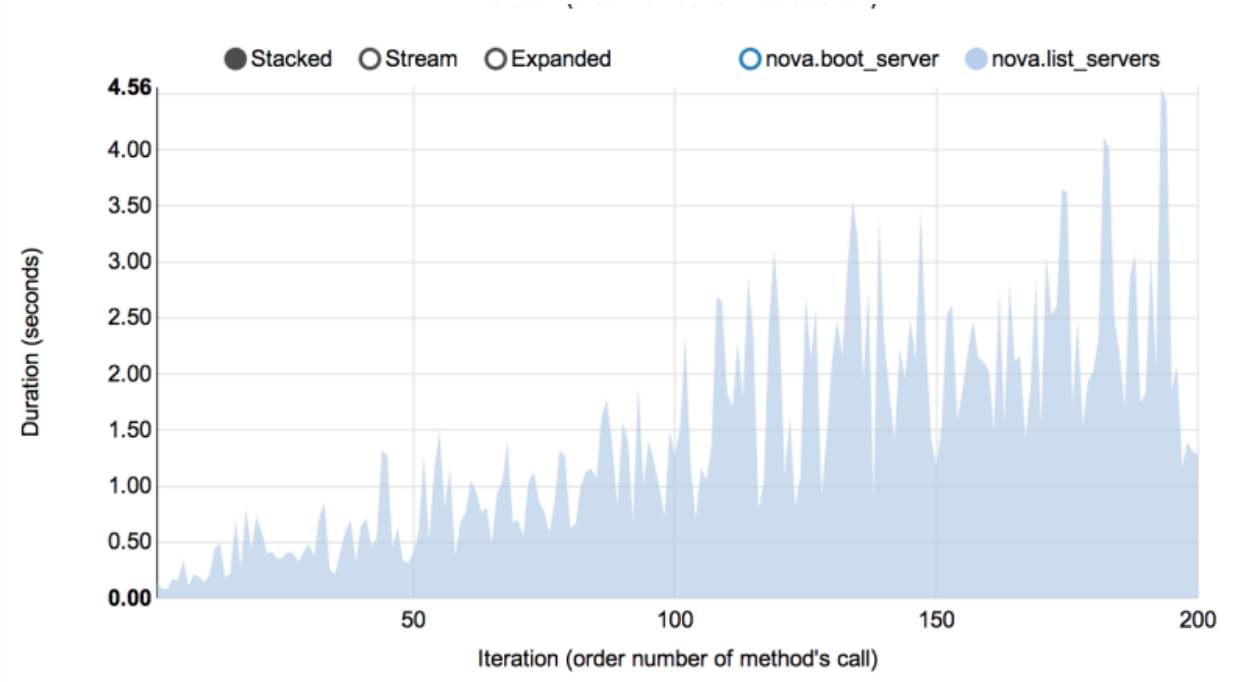
So Rally has unexpectedly indicated that setting the *amqp_rpc_single_reply_queue* option apparently affects the cloud performance, but in quite an opposite way rather than it was thought before.

Performance of Nova list command

Another interesting result comes from the *NovaServers.boot_and_list_server* scenario, which enabled us to we launched the following benchmark with Rally:

- **Benchmark environment** (which we also call **"Context"**): 1 temporary OpenStack user.
- **Benchmark scenario**: boot a single VM from this user & list all VMs.
- **Benchmark runner** setting: repeat this procedure 200 times in a continuous way.

During the execution of this benchmark scenario, the user has more and more VMs on each iteration. Rally has shown that in this case, the performance of the **VM list** command in Nova is degrading much faster than one might expect:

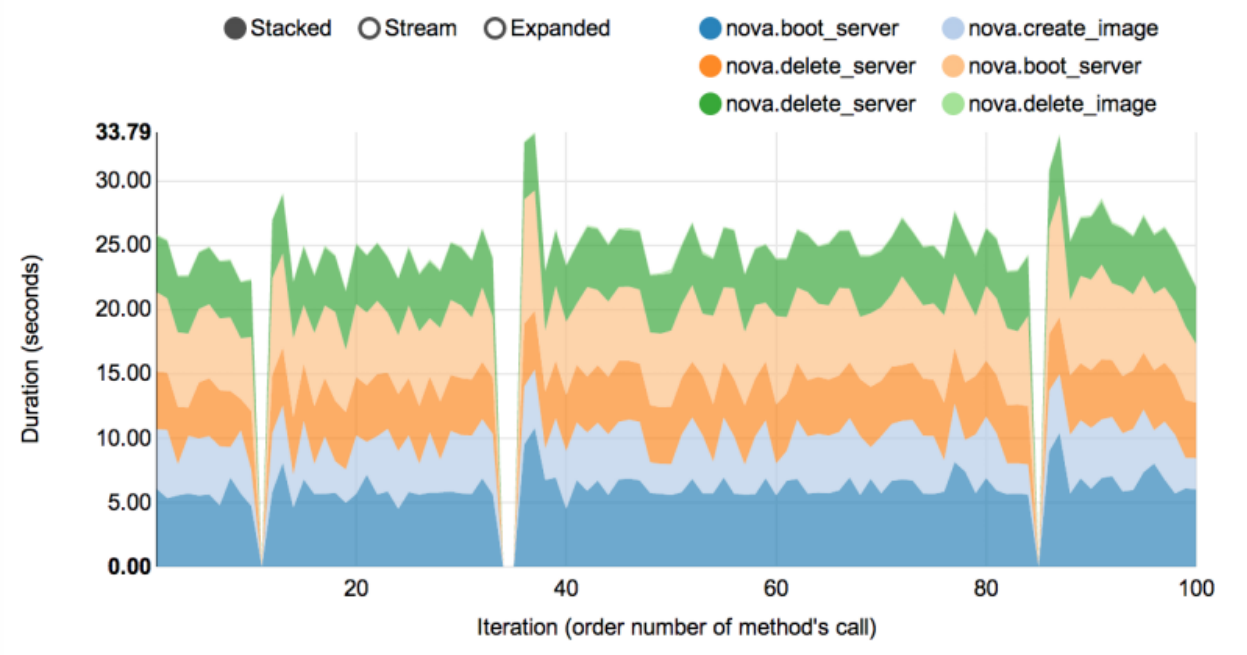


Complex scenarios

In fact, the vast majority of Rally scenarios is expressed as a sequence of **"atomic" actions**. For example, *NovaServers.snapshot* is composed of 6 atomic actions:

1. boot VM
2. snapshot VM
3. delete VM
4. boot VM from snapshot
5. delete VM
6. delete snapshot

Rally measures not only the performance of the benchmark scenario as a whole, but also that of single atomic actions. As a result, Rally also plots the atomic actions performance data for each benchmark iteration in a quite detailed way:

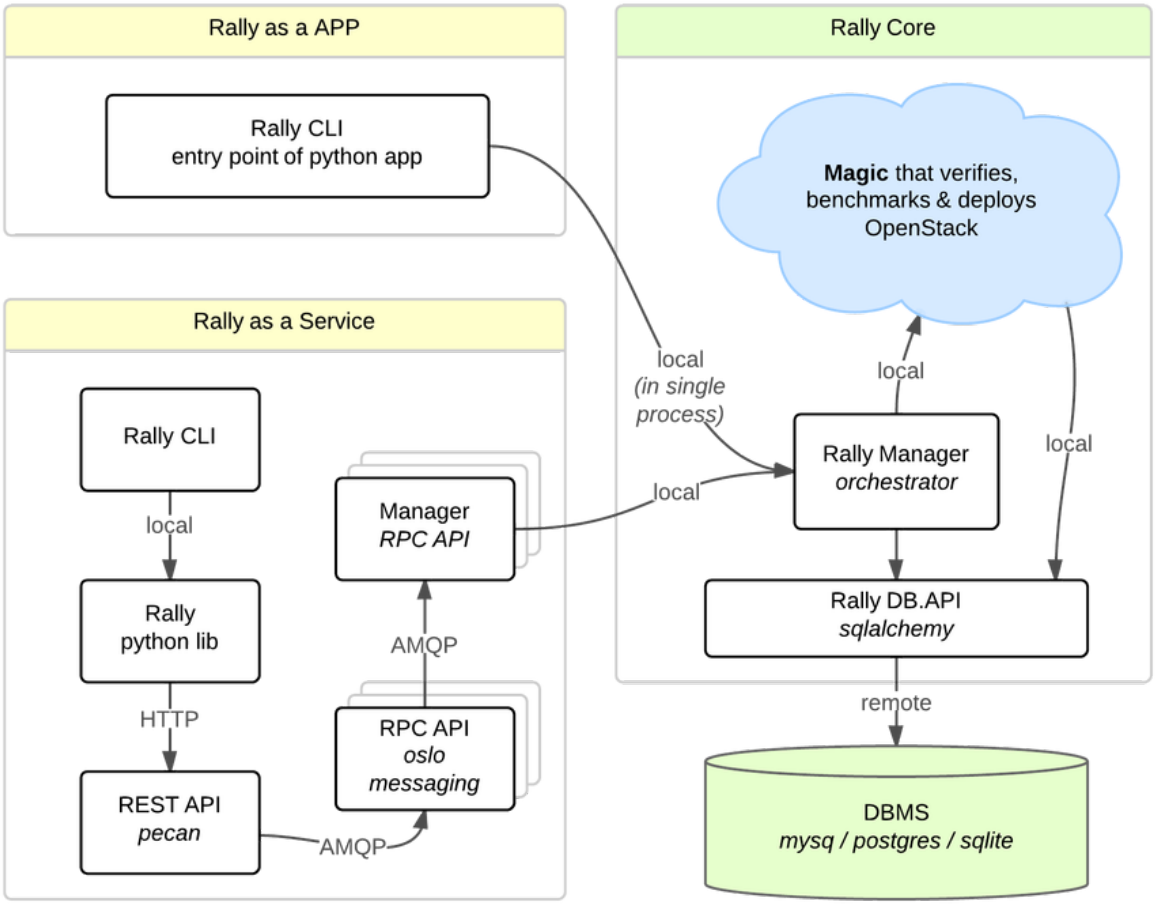


1.1.4 Architecture

Usually OpenStack projects are implemented *"as-a-Service"*, so Rally provides this approach. In addition, it implements a *CLI-driven* approach that does not require a daemon:

1. **Rally as-a-Service**: Run rally as a set of daemons that present Web UI *(work in progress)* so 1 RaaS could be used by a whole team.
2. **Rally as-an-App**: Rally as a just lightweight and portable CLI app (without any daemons) that makes it simple to use & develop.

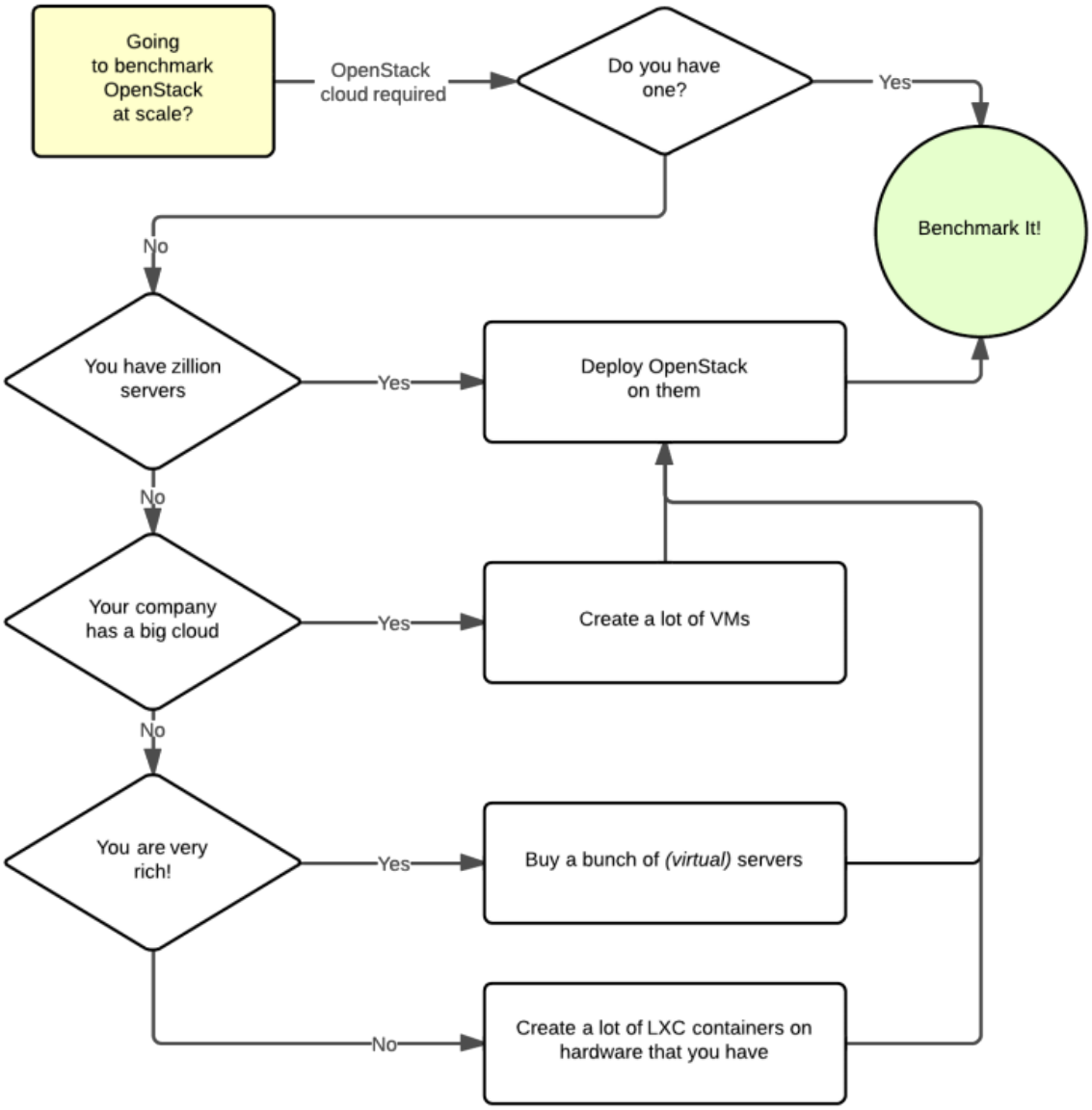
The diagram below shows how this is possible:



The actual **Rally core** consists of 4 main components, listed below in the order they go into action:

1. **Server Providers** - provide a **unified interface** for interaction with different **virtualization technologies** (*LXS*, *Virsh* etc.) and **cloud suppliers** (like *Amazon*): it does so via *ssh* access and in one *L3 network*;
2. **Deploy Engines** - deploy some OpenStack distribution (like *DevStack* or *FUEL*) before any benchmarking procedures take place, using servers retrieved from Server Providers;
3. **Verification** - runs *Tempest* (or another specific set of tests) against the deployed cloud to check that it works correctly, collects results & presents them in human readable form;
4. **Benchmark Engine** - allows to write parameterized benchmark scenarios & run them against the cloud.

It should become fairly obvious why Rally core needs to be split to these parts if you take a look at the following diagram that visualizes a rough **algorithm for starting benchmarking OpenStack at scale**. Keep in mind that there might be lots of different ways to set up virtual servers, as well as to deploy OpenStack to them.



1.2 Installation

1.2.1 Automated installation

The easiest way to install Rally is by executing its installation script

```
wget -q -O- https://raw.githubusercontent.com/openstack/rally/master/install_rally.sh | bash
# or using curl
curl https://raw.githubusercontent.com/openstack/rally/master/install_rally.sh | bash
```

The installation script will also check if all the software required by Rally is already installed in your system; if run as **root** user and some dependency is missing it will ask you if you want to install the required packages.

By default it will install Rally in a virtualenv in `~/rally` when run as standard user, or install system wide when run as root. You can install Rally in a venv by using the option `--target`:

```
./install_rally.sh --target /foo/bar
```

You can also install Rally system wide by running script as root and without `--target` option:

```
sudo ./install_rally.sh
```

Run `./install_rally.sh` with option `--help` to have a list of all available options:

```
 $ ./install_rally.sh --help
 Usage: install_rally.sh [options]

 This script will install rally either in the system (as root) or in a virtual environment.

Options:
  -h, --help             Print this help text
  -v, --verbose          Verbose mode
  -s, --system           Instead of creating a virtualenv, install as
                         system package.
  -d, --target DIRECTORY Install Rally virtual environment into DIRECTORY.
                         (Default: $HOME/rally).
  -f, --overwrite        Remove target directory if it already exists.
  -y, --yes              Do not ask for confirmation: assume a 'yes' reply
                         to every question.
  -D, --dbtype TYPE      Select the database type. TYPE can be one of
                         'sqlite', 'mysql', 'postgres'.
                         Default: sqlite
  --db-user USER         Database user to use. Only used when --dbtype
                         is either 'mysql' or 'postgres'.
  --db-password PASSWORD Password of the database user. Only used when
                         --dbtype is either 'mysql' or 'postgres'.
  --db-host HOST         Database host. Only used when --dbtype is
                         either 'mysql' or 'postgres'
  --db-name NAME         Name of the database. Only used when --dbtype is
                         either 'mysql' or 'postgres'
  -p, --python EXE       The python interpreter to use. Default: /usr/bin/python.
```

Notes: the script will check if all the software required by Rally is already installed in your system. If this is not the case, it will exit, suggesting you the command to issue **as root** in order to install the dependencies.

You also have to set up the **Rally database** after the installation is complete:

```
rally-manage db recreate
```

1.2.2 Rally with DevStack all-in-one installation

It is also possible to install Rally with DevStack. First, clone the corresponding repositories:

```
git clone https://git.openstack.org/openstack-dev/devstack
git clone https://github.com/openstack/rally
```

Then, configure DevStack to run Rally:

```
cd devstack
cp samples/local.conf local.conf
echo "enable_plugin rally https://github.com/openstack/rally master" >> localrc
```

Finally, run DevStack as usually:

```
./stack.sh
```

1.2.3 Rally & Docker

First you need to install Docker; Docker supplies installation instructions for various OSes.

You can either use the official Rally Docker image, or build your own from the Rally source. To do that, change directory to the root directory of the Rally git repository and run:

```
docker build -t myrally .
```

If you build your own Docker image, substitute `myrally` for `rallyforge/rally` in the commands below.

The Rally Docker image is configured to store local settings and the database in the user's home directory. For persistence of these data, you may want to keep this directory outside of the container. This may be done by the following steps:

```
sudo mkdir /var/lib/rally_container
sudo chown 65500 /var/lib/rally_container
docker run -it -v /var/lib/rally_container:/home/rally rallyforge/rally
```

Note: In order for the volume to be accessible by the Rally user (uid: 65500) inside the container, it must be accessible by UID 65500 *outside* the container as well, which is why it is created in `/var/lib/rally`. Creating it in your home directory is only likely to work if your home directory has excessively open permissions (e.g., `0755`), which is not recommended.

You may want to save the last command as an alias:

```
echo 'alias dock_rally="docker run -it -v /var/lib/rally_container:/home/rally rallyforge/rally"' >>
```

After executing `dock_rally`, or `docker run ...`, you will have bash running inside the container with Rally installed. You may do anything with Rally, but you need to create the database first:

```
user@box:~/rally$ dock_rally
rally@1cc98e0b5941:~$ rally-manage db recreate
rally@1cc98e0b5941:~$ rally deployment list
There are no deployments. To create a new deployment, use:
rally deployment create
rally@1cc98e0b5941:~$
```

In case you have SELinux enabled and Rally fails to create the database, try executing the following commands to put SELinux into Permissive Mode on the host machine

```
sed -i 's/SELINUX=enforcing/SELINUX=permissive/' /etc/selinux/config
setenforce permissive
```

Rally currently has no SELinux policy, which is why it must be run in Permissive mode for certain configurations. If you can help create an SELinux policy for Rally, please contribute!

More about docker: https://www.docker.com/

1.3 Rally step-by-step

In the following tutorial, we will guide you step-by-step through different use cases that might occur in Rally, starting with the easy ones and moving towards more complicated cases.

1.3.1 Step 0. Installation

The easiest way to install Rally is by running its installation script:

```
wget -q -O- https://raw.githubusercontent.com/openstack/rally/master/install_rally.sh | bash
# or using curl:
curl https://raw.githubusercontent.com/openstack/rally/master/install_rally.sh | bash
```

If you execute the script as regular user, Rally will create a new virtual environment in `~/rally/` and install in it Rally, and will use *sqlite* as database backend. If you execute the script as root, Rally will be installed system wide. For more installation options, please refer to the *installation* page.

Note: Rally requires Python version 2.6, 2.7 or 3.4.

Now that you have rally installed, you are ready to start *benchmarking OpenStack with it*!

1.3.2 Step 1. Setting up the environment and running a benchmark from samples

- Registering an OpenStack deployment in Rally
- Benchmarking
- Report generation

In this demo, we will show how to perform some basic operations in Rally, such as registering an OpenStack cloud, benchmarking it and generating benchmark reports.

We assume that you have a *Rally installation* and an already existing OpenStack deployment with Keystone available at *<KEYSTONE_AUTH_URL>*.

Registering an OpenStack deployment in Rally

First, you have to provide Rally with an OpenStack deployment it is going to benchmark. This should be done either through OpenRC files or through deployment configuration files. In case you already have an *OpenRC*, it is extremely simple to register a deployment with the *deployment create* command:

```
$ . openrc admin admin
$ rally deployment create --fromenv --name=existing
+--------------------------------------+----------------------------+------------+------------------
| uuid                                 | created_at                 | name       | status
+--------------------------------------+----------------------------+------------+------------------
| 28f90d74-d940-4874-a8ee-04fda59576da | 2015-01-18 00:11:38.059983 | existing   | deploy->finished
+--------------------------------------+----------------------------+------------+------------------
Using deployment : <Deployment UUID>
...
```

Alternatively, you can put the information about your cloud credentials into a JSON configuration file (let's call it existing.json). The *deployment create* command has a slightly different syntax in this case:

```
$ rally deployment create --file=existing.json --name=existing
+--------------------------------------+----------------------------+------------+------------------
| uuid                                 | created_at                 | name       | status
+--------------------------------------+----------------------------+------------+------------------
| 28f90d74-d940-4874-a8ee-04fda59576da | 2015-01-18 00:11:38.059983 | existing   | deploy->finished
+--------------------------------------+----------------------------+------------+------------------
Using deployment : <Deployment UUID>
...
```

Note the last line in the output. It says that the just created deployment is now used by Rally; that means that all the benchmarking operations from now on are going to be performed on this deployment. Later we will show how to switch between different deployments.

Finally, the *deployment check* command enables you to verify that your current deployment is healthy and ready to be benchmarked:

```
$ rally deployment check
keystone endpoints are valid and following services are available:
+----------+----------------+-----------+
| services | type           | status    |
+----------+----------------+-----------+
| cinder   | volume         | Available |
| cinderv2 | volumev2       | Available |
| ec2      | ec2            | Available |
| glance   | image          | Available |
| heat     | orchestration  | Available |
| heat-cfn | cloudformation | Available |
| keystone | identity       | Available |
| nova     | compute        | Available |
| novav21  | computev21     | Available |
| s3       | s3             | Available |
+----------+----------------+-----------+
```

Benchmarking

Now that we have a working and registered deployment, we can start benchmarking it. The sequence of benchmarks to be launched by Rally should be specified in a *benchmark task configuration file* (either in *JSON* or in *YAML* format). Let's try one of the sample benchmark tasks available in samples/tasks/scenarios, say, the one that boots and deletes multiple servers (*samples/tasks/scenarios/nova/boot-and-delete.json*):

```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "^cirros.*uec$"
                },
                "force_delete": false
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 2
            },
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                }
            }
        }
    ]
}
```

To start a benchmark task, run the task start command (you can also add the *-v* option to print more logging information):

```
$ rally task start samples/tasks/scenarios/nova/boot-and-delete.json
--------------------------------------------------------------------------------
 Preparing input task
--------------------------------------------------------------------------------

Input task is:
<Your task config here>

--------------------------------------------------------------------------------
 Task  6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996: started
--------------------------------------------------------------------------------

Benchmarking... This can take a while...

To track task status use:

  rally task status
  or
  rally task detailed

--------------------------------------------------------------------------------
 Task 6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996: finished
--------------------------------------------------------------------------------

test scenario NovaServers.boot_and_delete_server
args position 0
args values:
{u'args': {u'flavor': {u'name': u'm1.tiny'},
           u'force_delete': False,
           u'image': {u'name': u'^cirros.*uec$'}},
 u'context': {u'users': {u'project_domain': u'default',
                         u'resource_management_workers': 30,
                         u'tenants': 3,
                         u'user_domain': u'default',
                         u'users_per_tenant': 2}},
 u'runner': {u'concurrency': 2, u'times': 10, u'type': u'constant'}}
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
| action             | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
| nova.boot_server   | 7.99      | 9.047     | 11.862    | 9.747         | 10.805        | 100.0%  |
| nova.delete_server | 4.427     | 4.574     | 4.772     | 4.677         | 4.725         | 100.0%  |
| total              | 12.556    | 13.621    | 16.37     | 14.252        | 15.311        | 100.0%  |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
Load duration: 70.1310448647
Full duration: 87.545541048


HINTS:
* To plot HTML graphics with this data, run:
    rally task report 6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996 --out output.html

* To get raw JSON output of task results, run:
    rally task results 6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996

Using task: 6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996
```

Note that the Rally input task above uses *regular expressions* to specify the image and flavor name to be used for server creation, since concrete names might differ from installation to installation. If this benchmark task fails, then the reason for that might a non-existing image/flavor specified in the task. To check what images/flavors are available in the deployment you are currently benchmarking, you might use the *rally show* command:

```
$ rally show images
+--------------------------------------+----------------------+------------+
|                 UUID                 |          Name        |  Size (B)  |
+--------------------------------------+----------------------+------------+
| 8dfd6098-0c26-4cb5-8e77-1ecb2db0b8ae |  CentOS 6.5 (x86_64) | 344457216 |
| 2b8d119e-9461-48fc-885b-1477abe2edc5 | CirrOS 0.3.4 (x86_64) |  13287936 |
+--------------------------------------+----------------------+------------+

$ rally show flavors

Flavors for user `admin` in tenant `admin`:
+----+-----------+-------+-----------+-----------+-----------+
| ID | Name      | vCPUs | RAM (MB)  | Swap (MB) | Disk (GB) |
+----+-----------+-------+-----------+-----------+-----------+
| 1  | m1.tiny   | 1     | 512       |           | 1         |
| 2  | m1.small  | 1     | 2048      |           | 20        |
| 3  | m1.medium | 2     | 4096      |           | 40        |
| 4  | m1.large  | 4     | 8192      |           | 80        |
| 5  | m1.xlarge | 8     | 16384     |           | 160       |
+----+-----------+-------+-----------+-----------+-----------+
```

Report generation

One of the most beautiful things in Rally is its task report generation mechanism. It enables you to create illustrative and comprehensive HTML reports based on the benchmarking data. To create and open at once such a report for the last task you have launched, call:

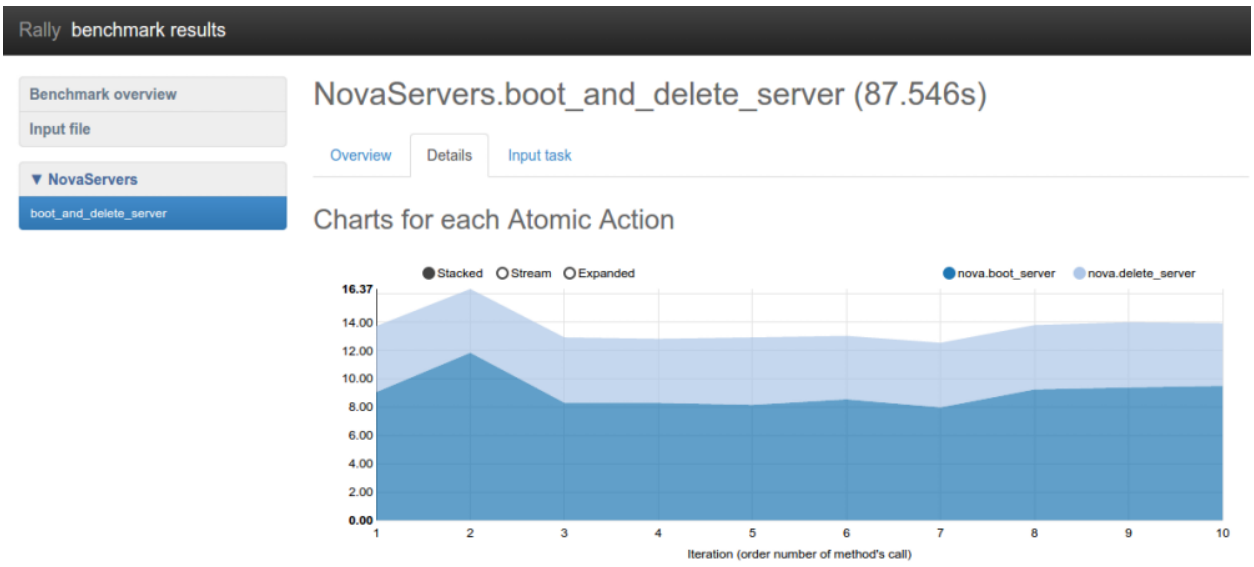
```
$ rally task report --out=report1.html --open
```

This will produce an HTML page with the overview of all the scenarios that you've included into the last benchmark task completed in Rally (in our case, this is just one scenario, and we will cover the topic of multiple scenarios in one task in *the next step of our tutorial*):

This aggregating table shows the duration of the load produced by the corresponding scenario (*"Load duration"*), the overall benchmark scenario execution time, including the duration of environment preparation with contexts (*"Full duration"*), the number of iterations of each scenario (*"Iterations"*), the type of the load used while running the scenario (*"Runner"*), the number of failed iterations (*"Errors"*) and finally whether the scenario has passed certain Success Criteria (*"SLA"*) that were set up by the user in the input configuration file (we will cover these criteria in *one of the next steps*).

By navigating in the left panel, you can switch to the detailed view of the benchmark results for the only scenario we included into our task, namely **NovaServers.boot_and_delete_server**:

This page, along with the description of the success criteria used to check the outcome of this scenario, shows some more detailed information and statistics about the duration of its iterations. Now, the *"Total durations"* table splits the duration of our scenario into the so-called **"atomic actions"**: in our case, the **"boot_and_delete_server"** scenario consists of two actions - **"boot_server"** and **"delete_server"**. You can also see how the scenario duration changed throughout is iterations in the *"Charts for the total duration"* section. Similar charts, but with atomic actions detalization, will arise if you switch to the *"Details"* tab of this page:



Note that all the charts on the report pages are very dynamic: you can change their contents by clicking the switches above the graph and see more information about its single points by hovering the cursor over these points.

Take some time to play around with these graphs and then move on to *the next step of our tutorial*.

1.3.3 Step 2. Rally input task format

- Basic input task syntax
- Multiple benchmarks in a single task
- Multiple configurations of the same scenario

Basic input task syntax

Rally comes with a really great collection of *plugins* and in most real-world cases you will use multiple plugins to test your OpenStack cloud. Rally makes it very easy to run **different test cases defined in a single task**. To do so, use the following syntax:

```
{
    "<ScenarioName1>": [<benchmark_config>, <benchmark_config2>, ...]
    "<ScenarioName2>": [<benchmark_config>, ...]
}
```

where <*benchmark_config*>, as before, is a dictionary:

```
{
    "args": { <scenario-specific arguments> },
    "runner": { <type of the runner and its specific parameters> },
    "context": { <contexts needed for this scenario> },
    "sla": { <different SLA configs> }
}
```

Multiple benchmarks in a single task

As an example, let's edit our configuration file from *step 1* so that it prescribes Rally to launch not only the **NovaServers.boot_and_delete_server** scenario, but also the **KeystoneBasic.create_delete_user** scenario. All we have to do is to append the configuration of the second scenario as yet another top-level key of our json file:

multiple-scenarios.json

```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "^cirros.*uec$"
                },
                "force_delete": false
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 2
            },
```

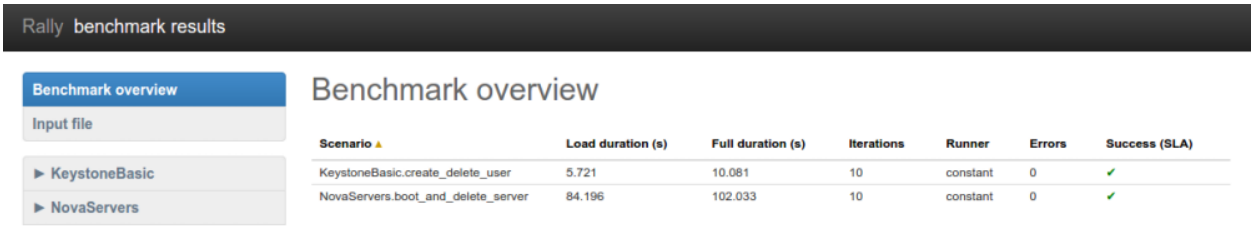
```
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                }
            }
        }
    ],
    "KeystoneBasic.create_delete_user": [
        {
            "args": {
                "name_length": 10
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 3
            }
        }
    ]
}
```

Now you can start this benchmark task as usually:

```
$ rally task start multiple-scenarios.json
...
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
| action             | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
| nova.boot_server   | 8.06      | 11.354    | 18.594    | 18.54         | 18.567        | 100.0%  |
| nova.delete_server | 4.364     | 5.054     | 6.837     | 6.805         | 6.821         | 100.0%  |
| total              | 12.572    | 16.408    | 25.396    | 25.374        | 25.385        | 100.0%  |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+
Load duration: 84.1959171295
Full duration: 102.033041
-------------------------------------------------------------------------------

...

+----------------------+-----------+-----------+-----------+---------------+---------------+---------
| action               | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success
+----------------------+-----------+-----------+-----------+---------------+---------------+---------
| keystone.create_user | 0.676     | 0.875     | 1.03      | 1.02          | 1.025         | 100.0%
| keystone.delete_user | 0.407     | 0.647     | 0.84      | 0.739         | 0.79          | 100.0%
| total                | 1.082     | 1.522     | 1.757     | 1.724         | 1.741         | 100.0%
+----------------------+-----------+-----------+-----------+---------------+---------------+---------
Load duration: 5.72119688988
Full duration: 10.0808410645

...
```

Note that the HTML reports you can generate by typing **rally task report –out=report_name.html** after your benchmark task has completed will get richer as your benchmark task configuration file includes more benchmark scenarios. Let's take a look at the report overview page for a task that covers all the scenarios available in Rally:

```
$ rally task report --out=report_multiple_scenarios.html --open
```



Multiple configurations of the same scenario

Yet another thing you can do in Rally is to launch **the same benchmark scenario multiple times with different configurations**. That's why our configuration file stores a list for the key *"NovaServers.boot_and_delete_server"*: you can just append a different configuration of this benchmark scenario to this list to get it. Let's say, you want to run the **boot_and_delete_server** scenario twice: first using the *"m1.tiny"* flavor and then using the *"m1.tiny"* flavor:

multiple-configurations.json

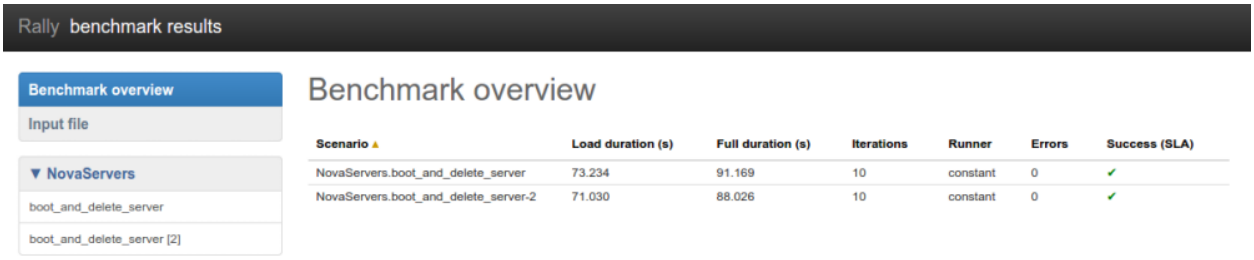
```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "^cirros.*uec$"
                },
                "force_delete": false
            },
            "runner": {...},
            "context": {...}
        },
        {
            "args": {
                "flavor": {
                    "name": "m1.small"
                },
                "image": {
                    "name": "^cirros.*uec$"
                },
                "force_delete": false
            },
            "runner": {...},
            "context": {...}
        }
    ]
}
```

That's it! You will get again the results for each configuration separately:

```
$ rally task start --task=multiple-configurations.json
...
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
| action             | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
| nova.boot_server   | 7.896     | 9.433     | 13.14     | 11.329        | 12.234        | 100.0%  |
| nova.delete_server | 4.435     | 4.898     | 6.975     | 5.144         | 6.059         | 100.0%  |
| total              | 12.404    | 14.331    | 17.979    | 16.72         | 17.349        | 100.0%  |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
Load duration: 73.2339417934
Full duration: 91.1692159176
--------------------------------------------------------------------------------

...

+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
| action             | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
| nova.boot_server   | 8.207     | 8.91      | 9.823     | 9.692         | 9.758         | 100.0%  |
| nova.delete_server | 4.405     | 4.767     | 6.477     | 4.904         | 5.691         | 100.0%  |
| total              | 12.735    | 13.677    | 16.301    | 14.596        | 15.449        | 100.0%  |
+--------------------+-----------+-----------+-----------+---------------+---------------+---------+-
Load duration: 71.029528141
Full duration: 88.0259010792
...
```

The HTML report will also look similar to what we have seen before:

```
$ rally task report --out=report_multiple_configuraions.html --open
```



1.3.4 Step 3. Benchmarking OpenStack with existing users

- Motivation
- Registering existing users in Rally
- Running benchmark scenarios with existing users

Motivation

There are two very important reasons from the production world of why it is preferable to use some already existing users to benchmark your OpenStack cloud:

1. *Read-only Keystone Backends:* creating temporary users for benchmark scenarios in Rally is just impossible in case of r/o Keystone backends like *LDAP* and *AD*.
2. *Safety:* Rally can be run from an isolated group of users, and if something goes wrong, this won't affect the rest of the cloud users.

Registering existing users in Rally

The information about existing users in your OpenStack cloud should be passed to Rally at the *deployment initialization step*. You have to use the **ExistingCloud** deployment plugin that just provides Rally with credentials of an already existing cloud. The difference from the deployment configuration we've seen previously is that you should set up the *"users"* section with the credentials of already existing users. Let's call this deployment configuration file *existing_users.json*:

```
{
    "type": "ExistingCloud",
    "auth_url": "http://example.net:5000/v2.0/",
    "region_name": "RegionOne",
    "endpoint_type": "public",
    "admin": {
        "username": "admin",
        "password": "pa55word",
        "tenant_name": "demo"
    },
    "users": [
        {
            "username": "b1",
            "password": "1234",
            "tenant_name": "testing"
        },
        {
            "username": "b2",
            "password": "1234",
            "tenant_name": "testing"
        }
    ]
}
```

This deployment configuration requires some basic information about the OpenStack cloud like the region name, auth url. admin user credentials, and any amount of users already existing in the system. Rally will use their credentials to generate load in against this deployment as soon as we register it as usual:

```
$ rally deployment create --file existings_users --name our_cloud
+--------------------------------------+----------------------------+------------+------------------+-
| uuid                                 | created_at                 | name       | status           |
+--------------------------------------+----------------------------+------------+------------------+-
| 1849a9bf-4b18-4fd5-89f0-ddcc56eae4c9 | 2015-03-28 02:43:27.759702 | our_cloud | deploy->finished |
+--------------------------------------+----------------------------+------------+------------------+-
Using deployment: 1849a9bf-4b18-4fd5-89f0-ddcc56eae4c9
~/.rally/openrc was updated
```

After that, the **rally show** command lists the resources for each user separately:

```
$ rally show images

Images for user `admin` in tenant `admin`:
+--------------------------------------+--------------------------------+------------+
| UUID                                 | Name                           | Size (B)   |
```

```
+--------------------------------------+--------------------------------+-----------+
| 041cfd70-0e90-4ed6-8c0c-ad9c12a94191 | cirros-0.3.4-x86_64-uec        | 25165824  |
| 87710f09-3625-4496-9d18-e20e34906b72 | Fedora-x86_64-20-20140618-sda  | 209649664 |
| b0f269be-4859-48e0-a0ca-03fb80d14602 | cirros-0.3.4-x86_64-uec-ramdisk | 3740163   |
| d82eaf7a-ff63-4826-9aa7-5fa105610e01 | cirros-0.3.4-x86_64-uec-kernel | 4979632   |
+--------------------------------------+--------------------------------+-----------+

Images for user `b1` in tenant `testing`:
+--------------------------------------+--------------------------------+-----------+
| UUID                                 | Name                           | Size (B)  |
+--------------------------------------+--------------------------------+-----------+
| 041cfd70-0e90-4ed6-8c0c-ad9c12a94191 | cirros-0.3.4-x86_64-uec        | 25165824  |
| 87710f09-3625-4496-9d18-e20e34906b72 | Fedora-x86_64-20-20140618-sda  | 209649664 |
| b0f269be-4859-48e0-a0ca-03fb80d14602 | cirros-0.3.4-x86_64-uec-ramdisk | 3740163   |
| d82eaf7a-ff63-4826-9aa7-5fa105610e01 | cirros-0.3.4-x86_64-uec-kernel | 4979632   |
+--------------------------------------+--------------------------------+-----------+

Images for user `b2` in tenant `testing`:
+--------------------------------------+--------------------------------+-----------+
| UUID                                 | Name                           | Size (B)  |
+--------------------------------------+--------------------------------+-----------+
| 041cfd70-0e90-4ed6-8c0c-ad9c12a94191 | cirros-0.3.4-x86_64-uec        | 25165824  |
| 87710f09-3625-4496-9d18-e20e34906b72 | Fedora-x86_64-20-20140618-sda  | 209649664 |
| b0f269be-4859-48e0-a0ca-03fb80d14602 | cirros-0.3.4-x86_64-uec-ramdisk | 3740163   |
| d82eaf7a-ff63-4826-9aa7-5fa105610e01 | cirros-0.3.4-x86_64-uec-kernel | 4979632   |
+--------------------------------------+--------------------------------+-----------+
```

With this new deployment being active, Rally will use the already existing users *"b1"* and *"b2"* instead of creating the temporary ones when launching benchmark task that do not specify the *"users"* context.

Running benchmark scenarios with existing users

After you have registered a deployment with existing users, don't forget to remove the *"users"* context from your benchmark task configuration if you want to use existing users, like in the following configuration file (*boot-and-delete.json*):

```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor": {
                    "name": "m1.tiny"
                },
                "image": {
                    "name": "^cirros.*uec$"
                },
                "force_delete": false
            },
            "runner": {
                "type": "constant",
                "times": 10,
                "concurrency": 2
            },
            "context": {}
        }
    ]
}
```

When you start this task, it will use the existing users *"b1"* and *"b2"* instead of creating the temporary ones:

```
$ rally task start samples/tasks/scenarios/nova/boot-and-delete.json
...
```

It goes without saying that support of benchmarking with predefined users simplifies the usage of Rally for generating loads against production clouds.

(based on: http://boris-42.me/rally-can-generate-load-with-passed-users-now/)

1.3.5 Step 4. Adding success criteria (SLA) for benchmarks

- SLA - Service-Level Agreement (Success Criteria)
- Checking SLA
- SLA in task report

SLA - Service-Level Agreement (Success Criteria)

Rally allows you to set success criteria (also called *SLA - Service-Level Agreement*) for every benchmark. Rally will automatically check them for you.

To configure the SLA, add the *"sla"* section to the configuration of the corresponding benchmark (the check name is a key associated with its target value). You can combine different success criteria:

```
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                ...
            },
            "runner": {
                ...
            },
            "context": {
                ...
            },
            "sla": {
                "max_seconds_per_iteration": 10,
                "failure_rate": {
                    "max": 25
                }
            }
        }
    ]
}
```

Such configuration will mark the **NovaServers.boot_and_delete_server** benchmark scenario as not successful if either some iteration took more than 10 seconds or more than 25% iterations failed.

Checking SLA

Let us show you how Rally SLA work using a simple example based on **Dummy benchmark scenarios**. These scenarios actually do not perform any OpenStack-related stuff but are very useful for testing the behaviors of Rally. Let us put in a new task, *test-sla.json*, 2 scenarios – one that does nothing and another that just throws an exception:

```
{
    "Dummy.dummy": [
        {
            "args": {},
            "runner": {
                "type": "constant",
                "times": 5,
                "concurrency": 2
            },
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                }
            },
            "sla": {
                "failure_rate": {"max": 0.0}
            }
        }
    ],
    "Dummy.dummy_exception": [
        {
            "args": {},
            "runner": {
                "type": "constant",
                "times": 5,
                "concurrency": 2
            },
            "context": {
                "users": {
                    "tenants": 3,
                    "users_per_tenant": 2
                }
            },
            "sla": {
                "failure_rate": {"max": 0.0}
            }
        }
    ]
}
```

Note that both scenarios in these tasks have the **maximum failure rate of 0%** as their **success criterion**. We expect that the first scenario will pass this criterion while the second will fail it. Let's start the task:

```
$ rally task start test-sla.json
...
```

After the task completes, run *rally task sla_check* to check the results again the success criteria you defined in the task:

```
$ rally task sla_check
+----------------------+-----+--------------+--------+-------------------------------------------
| benchmark            | pos | criterion    | status | detail
+----------------------+-----+--------------+--------+-------------------------------------------
| Dummy.dummy          | 0   | failure_rate | PASS   | Maximum failure rate percent 0.0% failures, r
| Dummy.dummy_exception | 0   | failure_rate | FAIL   | Maximum failure rate percent 0.0% failures, r
+----------------------+-----+--------------+--------+-------------------------------------------
```

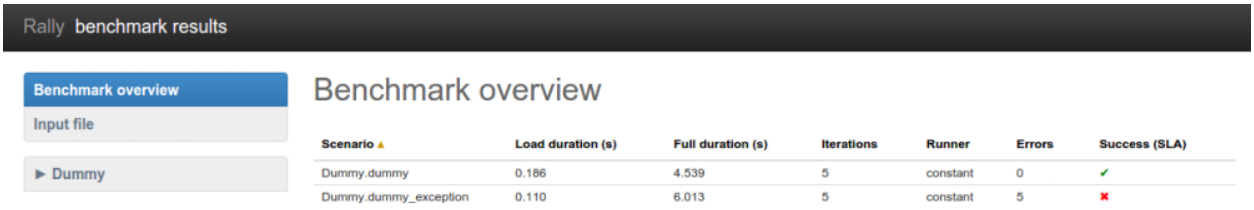
Exactly as expected.

SLA in task report

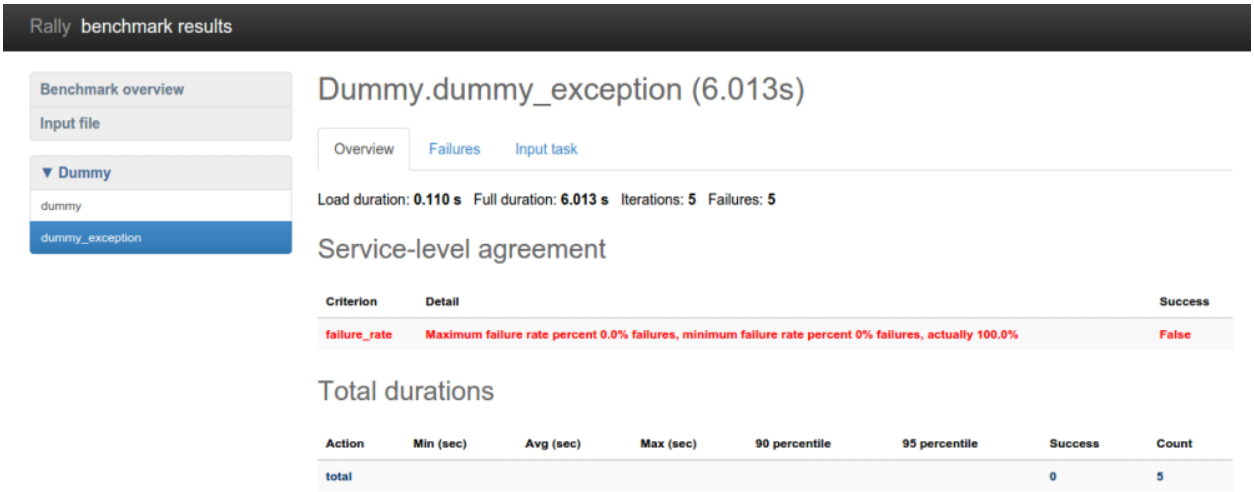
SLA checks are nicely visualized in task reports. Generate one:

```
$ rally task report --out=report_sla.html --open
```

Benchmark scenarios that have passed SLA have a green check on the overview page:



Somewhat more detailed information about SLA is displayed on the scenario pages:



Success criteria present a very useful concept that enables not only to analyze the outcome of your benchmark tasks, but also to control their execution. In *one of the next sections* of our tutorial, we will show how to use SLA to abort the load generation before your OpenStack goes wrong.

1.3.6 Step 5. Rally task templates

- Basic template syntax
- Using the default values
- Advanced templates

Basic template syntax

A nice feature of the input task format used in Rally is that it supports the **template syntax** based on Jinja2. This turns out to be extremely useful when, say, you have a fixed structure of your task but you want to parameterize this task in some way. For example, imagine your input task file (*task.yaml*) runs a set of Nova scenarios:

```
---

  NovaServers.boot_and_delete_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: "^cirros.*uec$"
      runner:
        type: "constant"
        times: 2
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

  NovaServers.resize_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: "^cirros.*uec$"
        to_flavor:
            name: "m1.small"
      runner:
        type: "constant"
        times: 3
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1
```

In all the three scenarios above, the *"^cirros.*uec$"* image is passed to the scenario as an argument (so that these scenarios use an appropriate image while booting servers). Let's say you want to run the same set of scenarios with the same runner/context/sla, but you want to try another image while booting server to compare the performance. The most elegant solution is then to turn the image name into a template variable:

```
---

  NovaServers.boot_and_delete_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: {{image_name}}
      runner:
        type: "constant"
        times: 2
```

```
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

  NovaServers.resize_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: {{image_name}}
        to_flavor:
            name: "m1.small"
      runner:
        type: "constant"
        times: 3
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1
```

and then pass the argument value for **{{image_name}}** when starting a task with this configuration file. Rally provides you with different ways to do that:

1. Pass the argument values directly in the command-line interface (with either a JSON or YAML dictionary):

```
$ rally task start task.yaml --task-args '{"image_name": "^cirros.*uec$"}'
$ rally task start task.yaml --task-args 'image_name: "^cirros.*uec$"'
```

2. Refer to a file that specifies the argument values (JSON/YAML):

```
$ rally task start task.yaml --task-args-file args.json
$ rally task start task.yaml --task-args-file args.yaml
```

where the files containing argument values should look as follows:

args.json:

```
{
    "image_name": "^cirros.*uec$"
}
```

args.yaml:

```
---

  image_name: "^cirros.*uec$"
```

Passed in either way, these parameter values will be substituted by Rally when starting a task:

```
$ rally task start task.yaml --task-args "image_name: "^cirros.*uec$""
--------------------------------------------------------------------------------
 Preparing input task
--------------------------------------------------------------------------------

Input task is:
---
```

```
  NovaServers.boot_and_delete_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: ^cirros.*uec$
      runner:
        type: "constant"
        times: 2
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

  NovaServers.resize_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: ^cirros.*uec$
        to_flavor:
            name: "m1.small"
      runner:
        type: "constant"
        times: 3
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

--------------------------------------------------------------------------------
 Task  cbf7eb97-0f1d-42d3-a1f1-3cc6f45ce23f: started
--------------------------------------------------------------------------------

Benchmarking... This can take a while...
```

Using the default values

Note that the Jinja2 template syntax allows you to set the default values for your parameters. With default values set, your task file will work even if you don't parameterize it explicitly while starting a task. The default values should be set using the *{% set ... %}* clause (*task.yaml*):

```
{% set image_name = image_name or "^cirros.*uec$" %}
---

  NovaServers.boot_and_delete_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: {{image_name}}
      runner:
```

```
        type: "constant"
        times: 2
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

    ...
```

If you don't pass the value for *{{image_name}}* while starting a task, the default one will be used:

```
$ rally task start task.yaml
--------------------------------------------------------------------------------
 Preparing input task
--------------------------------------------------------------------------------

Input task is:
---

  NovaServers.boot_and_delete_server:
    -
      args:
        flavor:
            name: "m1.tiny"
        image:
            name: ^cirros.*uec$
      runner:
        type: "constant"
        times: 2
        concurrency: 1
      context:
        users:
          tenants: 1
          users_per_tenant: 1

    ...
```

Advanced templates

Rally makes it possible to use all the power of Jinja2 template syntax, including the mechanism of **built-in functions**. This enables you to construct elegant task files capable of generating complex load on your cloud.

As an example, let us make up a task file that will create new users with increasing concurrency. The input task file (*task.yaml*) below uses the Jinja2 **for-endfor** construct to accomplish that:

```
---

  KeystoneBasic.create_user:
  {% for i in range(2, 11, 2) %}
    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: {{i}}
      sla:
```

```
        failure_rate:
          max: 0
  {% endfor %}
```

In this case, you don't need to pass any arguments via *–task-args/–task-args-file*, but as soon as you start this task, Rally will automatically unfold the for-loop for you:

```
$ rally task start task.yaml
--------------------------------------------------------------------------------
 Preparing input task
--------------------------------------------------------------------------------

Input task is:
---

  KeystoneBasic.create_user:

    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: 2
      sla:
        failure_rate:
          max: 0

    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: 4
      sla:
        failure_rate:
          max: 0

    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: 6
      sla:
        failure_rate:
          max: 0

    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: 8
      sla:
```

```
        failure_rate:
          max: 0

    -
      args:
        name_length: 10
      runner:
        type: "constant"
        times: 10
        concurrency: 10
      sla:
        failure_rate:
          max: 0


--------------------------------------------------------------------------------
 Task  ea7e97e3-dd98-4a81-868a-5bb5b42b8610: started
--------------------------------------------------------------------------------

Benchmarking... This can take a while...
```

As you can see, the Rally task template syntax is a simple but powerful mechanism that not only enables you to write elegant task configurations, but also makes them more readable for other people. When used appropriately, it can really improve the understanding of your benchmarking procedures in Rally when shared with others.

1.3.7 Step 6. Aborting load generation on success criteria failure

Benchmarking pre-production and production OpenStack clouds is not a trivial task. From the one side it's important to reach the OpenStack cloud's limits, from the other side the cloud shouldn't be damaged. Rally aims to make this task as simple as possible. Since the very beginning Rally was able to generate enough load for any OpenStack cloud. Generating too big a load was the major issue for production clouds, because Rally didn't know how to stop the load until it was too late.

With the **"stop on SLA failure"** feature, however, things are much better.

This feature can be easily tested in real life by running one of the most important and plain benchmark scenario called *"KeystoneBasic.authenticate"*. This scenario just tries to authenticate from users that were pre-created by Rally. Rally input task looks as follows (*auth.yaml*):

```
---
  Authenticate.keystone:
  -
    runner:
      type: "rps"
      times: 6000
      rps: 50
    context:
      users:
        tenants: 5
        users_per_tenant: 10
    sla:
      max_avg_duration: 5
```

In human-readable form this input task means: *Create 5 tenants with 10 users in each, after that try to authenticate to Keystone 6000 times performing 50 authentications per second (running new authentication request every 20ms). Each time we are performing authentication from one of the Rally pre-created user. This task passes only if max average duration of authentication takes less than 5 seconds.*

Note that this test is quite dangerous because it can DDoS Keystone. We are running more and more simultaneously authentication requests and things may go wrong if something is not set properly (like on my DevStack deployment in Small VM on my laptop).

Let's run Rally task with **an argument that prescribes Rally to stop load on SLA failure**:

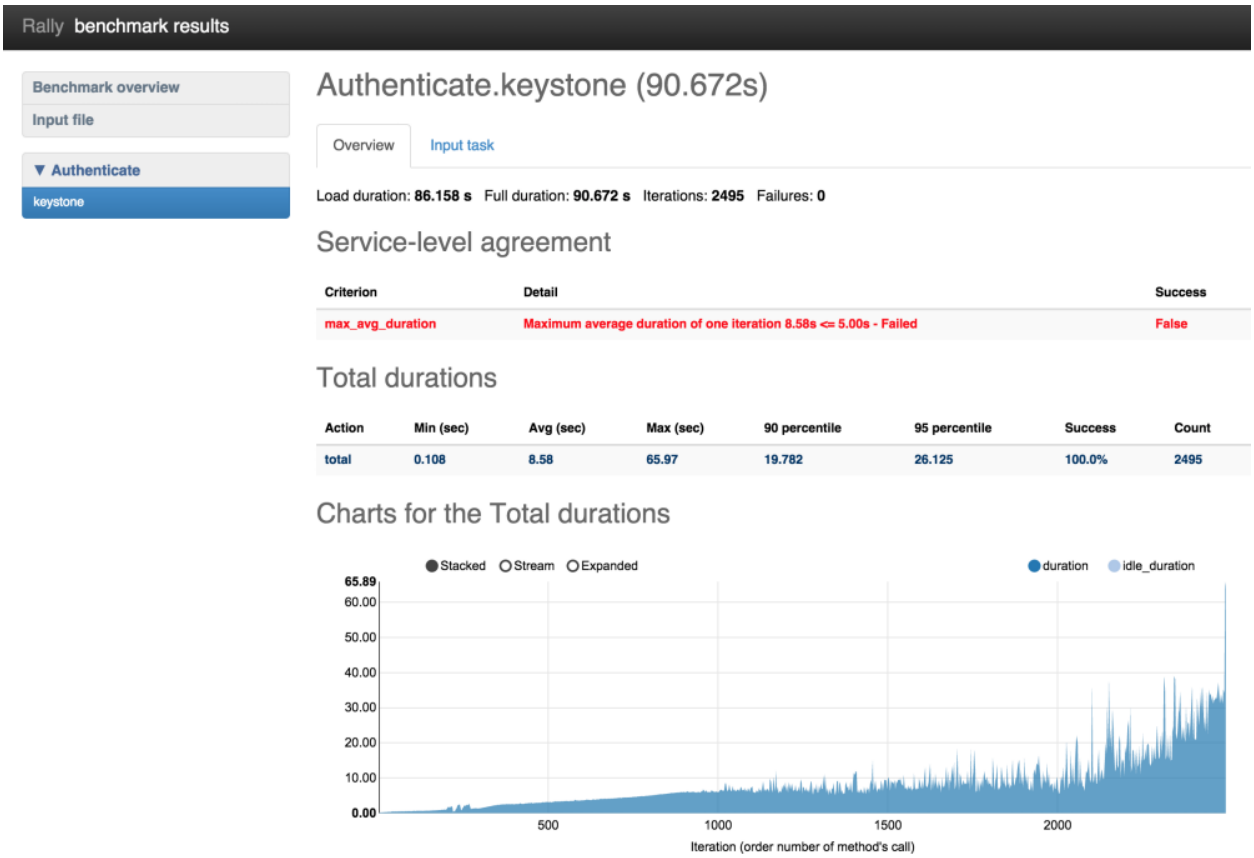
```
$ rally task start --abort-on-sla-failure auth.yaml

....
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
| action | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success | count |
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
| total  | 0.108     | 8.58      | 65.97     | 19.782        | 26.125        | 100.0%  | 2495  |
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
```

On the resulting table there are 2 interesting things:

1. Average duration was 8.58 sec which is more than 5 seconds
2. Rally performed only 2495 (instead of 6000) authentication requests

To understand better what has happened let's generate HTML report:

```
$ rally task report --out auth_report.html
```



On the chart with durations we can observe that the duration of authentication request reaches 65 seconds at the end of the load generation. **Rally stopped load at the very last moment just before the mad things happened. The reason why it runs so many attempts to authenticate is because of not enough good success criteria.** We had to run a lot of iterations to make average duration bigger than 5 seconds. Let's chose better success criteria for this task and run it one more time.

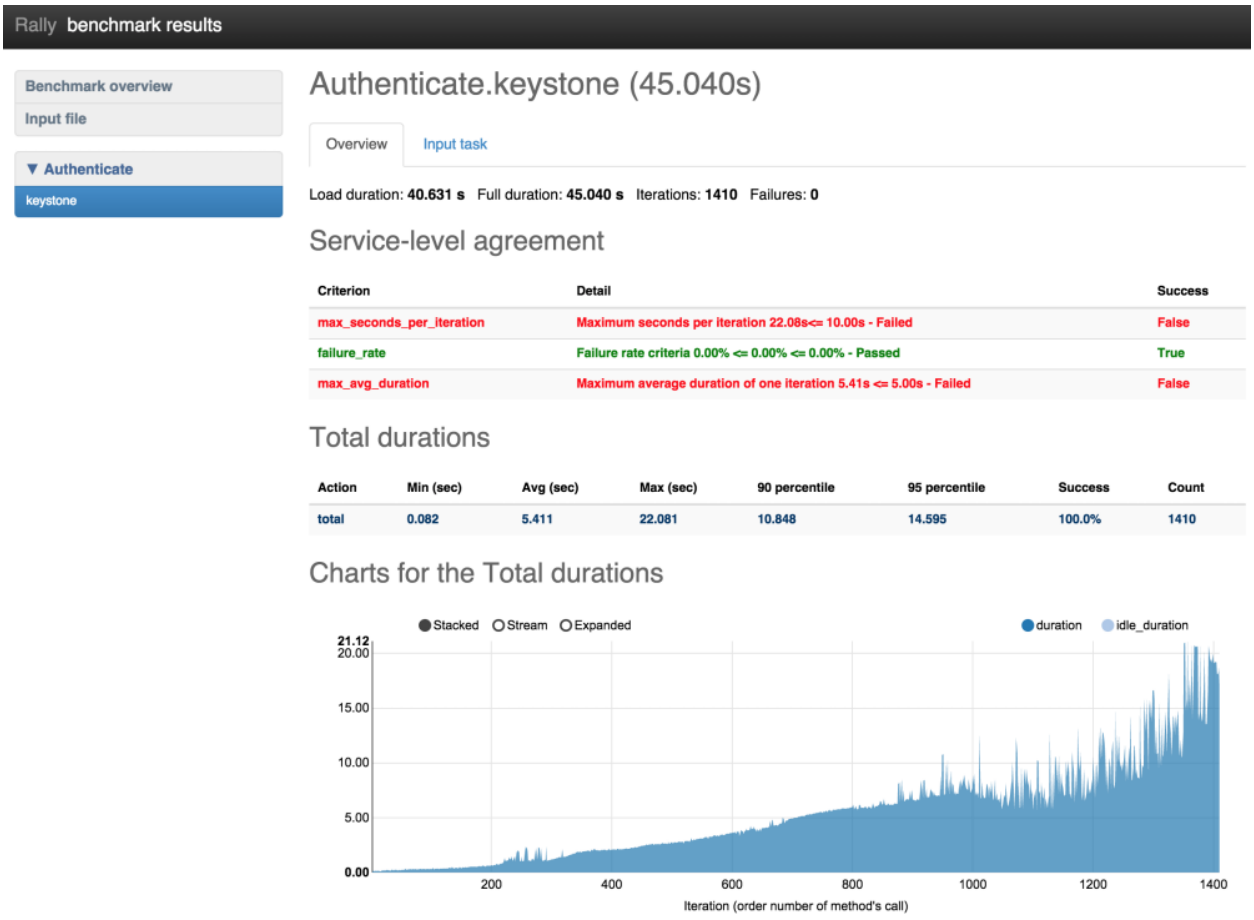
```
---
  Authenticate.keystone:
  -
    runner:
      type: "rps"
      times: 6000
      rps: 50
    context:
      users:
        tenants: 5
        users_per_tenant: 10
    sla:
      max_avg_duration: 5
      max_seconds_per_iteration: 10
      failure_rate:
        max: 0
```

Now our task is going to be successful if the following three conditions hold:

1. maximum average duration of authentication should be less than 5 seconds
2. maximum duration of any authentication should be less than 10 seconds
3. no failed authentication should appear

Let's run it!

```
$ rally task start --abort-on-sla-failure auth.yaml

...
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
| action | min (sec) | avg (sec) | max (sec) | 90 percentile | 95 percentile | success | count |
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
| total  | 0.082     | 5.411     | 22.081    | 10.848        | 14.595        | 100.0%  | 1410  |
+--------+-----------+-----------+-----------+---------------+---------------+---------+-------+
```



This time load stopped after 1410 iterations versus 2495 which is much better. The interesting thing on this chart is that first occurence of "> 10 second" authentication happened on 950 iteration. The reasonable question: "Why Rally run 500 more authentication requests then?". This appears from the math: During the execution of **bad** authentication (10 seconds) Rally performed about 50 request/sec * 10 sec = 500 new requests as a result we run 1400 iterations instead of 950.

(based on: http://boris-42.me/rally-tricks-stop-load-before-your-openstack-goes-wrong/)

1.3.8 Step 7. Working with multiple OpenStack clouds

Rally is an awesome tool that allows you to work with multiple clouds and can itself deploy them. We already know how to work with *a single cloud*. Let us now register 2 clouds in Rally: the one that we have access to and the other that we know is registered with wrong credentials.

```
$ . openrc admin admin  # openrc with correct credentials
$ rally deployment create --fromenv --name=cloud-1
+--------------------------------------+----------------------------+------------+------------------
| uuid                                 | created_at                 | name       | status
+--------------------------------------+----------------------------+------------+------------------
| 4251b491-73b2-422a-aecb-695a94165b5e | 2015-01-18 00:11:14.757203 | cloud-1    | deploy->finished
+--------------------------------------+----------------------------+------------+------------------
Using deployment: 4251b491-73b2-422a-aecb-695a94165b5e
~/.rally/openrc was updated
...
```

```
$ . bad_openrc admin admin  # openrc with wrong credentials
$ rally deployment create --fromenv --name=cloud-2
+--------------------------------------+----------------------------+------------+------------------
| uuid                                 | created_at                 | name       | status
+--------------------------------------+----------------------------+------------+------------------
| 658b9bae-1f9c-4036-9400-9e71e88864fc | 2015-01-18 00:38:26.127171 | cloud-2    | deploy->finished
+--------------------------------------+----------------------------+------------+------------------
Using deployment: 658b9bae-1f9c-4036-9400-9e71e88864fc
~/.rally/openrc was updated
...
```

Let us now list the deployments we have created:

```
$ rally deployment list
+--------------------------------------+----------------------------+------------+------------------
| uuid                                 | created_at                 | name       | status
+--------------------------------------+----------------------------+------------+------------------
| 4251b491-73b2-422a-aecb-695a94165b5e | 2015-01-05 00:11:14.757203 | cloud-1    | deploy->finished
| 658b9bae-1f9c-4036-9400-9e71e88864fc | 2015-01-05 00:40:58.451435 | cloud-2    | deploy->finished
+--------------------------------------+----------------------------+------------+------------------
```

Note that the second is marked as **"active"** because this is the deployment we have created most recently. This means that it will be automatically (unless its UUID or name is passed explicitly via the *–deployment* parameter) used by the commands that need a deployment, like *rally task start ...* or *rally deployment check*:

```
$ rally deployment check
Authentication Issues: wrong keystone credentials specified in your endpoint properties. (HTTP 401).

$ rally deployment check --deployment=cloud-1
keystone endpoints are valid and following services are available:
+----------+----------------+-----------+
| services | type           | status    |
+----------+----------------+-----------+
| cinder   | volume         | Available |
| cinderv2 | volumev2       | Available |
| ec2      | ec2            | Available |
| glance   | image          | Available |
| heat     | orchestration  | Available |
| heat-cfn | cloudformation | Available |
| keystone | identity       | Available |
| nova     | compute        | Available |
| novav21  | computev21     | Available |
| s3       | s3             | Available |
+----------+----------------+-----------+
```

You can also switch the active deployment using the **rally deployment use** command:

```
$ rally deployment use cloud-1
Using deployment: 658b9bae-1f9c-4036-9400-9e71e88864fc
~/.rally/openrc was updated
...

$ rally deployment check
keystone endpoints are valid and following services are available:
+----------+----------------+-----------+
| services | type           | status    |
+----------+----------------+-----------+
| cinder   | volume         | Available |
| cinderv2 | volumev2       | Available |
```

```
| ec2      | ec2            | Available |
| glance   | image          | Available |
| heat     | orchestration  | Available |
| heat-cfn | cloudformation | Available |
| keystone | identity       | Available |
| nova     | compute        | Available |
| novav21  | computev21     | Available |
| s3       | s3             | Available |
+----------+----------------+-----------+
```

Note the first two lines of the CLI output for the *rally deployment use* command. They tell you the UUID of the new active deployment and also say that the *~/.rally/openrc* file was updated – this is the place where the "active" UUID is actually stored by Rally.

One last detail about managing different deployments in Rally is that the *rally task list* command outputs only those tasks that were run against the currently active deployment, and you have to provide the *–all-deployments* parameter to list all the tasks:

```
$ rally task list
+--------------------------------------+-----------------+-----------------------------+--------------
| uuid                                 | deployment_name | created_at                  | duration
+--------------------------------------+-----------------+-----------------------------+--------------
| c21a6ecb-57b2-43d6-bbbb-d7a827f1b420 | cloud-1         | 2015-01-05 01:00:42.099596 | 0:00:13.41922
| f6dad6ab-1a6d-450d-8981-f77062c6ef4f | cloud-1         | 2015-01-05 01:05:57.653253 | 0:00:14.16049
+--------------------------------------+-----------------+-----------------------------+--------------
$ rally task list --all-deployment
+--------------------------------------+-----------------+-----------------------------+--------------
| uuid                                 | deployment_name | created_at                  | duration
+--------------------------------------+-----------------+-----------------------------+--------------
| c21a6ecb-57b2-43d6-bbbb-d7a827f1b420 | cloud-1         | 2015-01-05 01:00:42.099596 | 0:00:13.41922
| f6dad6ab-1a6d-450d-8981-f77062c6ef4f | cloud-1         | 2015-01-05 01:05:57.653253 | 0:00:14.16049
| 6fd9a19f-5cf8-4f76-ab72-2e34bb1d4996 | cloud-2         | 2015-01-05 01:14:51.428958 | 0:00:15.04226
+--------------------------------------+-----------------+-----------------------------+--------------
```

1.3.9 Step 8. Discovering more plugins in Rally

- Plugins in the Rally repository
- CLI: rally plugin show
- CLI: rally plugin list

Plugins in the Rally repository

Rally currently comes with a great collection of plugins that use the API of different OpenStack projects like **Keystone**, **Nova**, **Cinder**, **Glance** and so on. The good news is that you can combine multiple plugins in one task to test your cloud in a comprehensive way.

First, let's see what plugins are available in Rally. One of the ways to discover these plugins is just to inspect their source code. another is to use build-in rally plugin command.

CLI: rally plugin show

Rally plugin CLI command is much more convenient way to learn about different plugins in Rally. This command allows to list plugins and show detailed information about them:

```
$ rally plugin show create_meter_and_get_stats

NAME
    CeilometerStats.create_meter_and_get_stats
NAMESPACE
    default
MODULE
    rally.plugins.openstack.scenarios.ceilometer.stats
DESCRIPTION
    Meter is first created and then statistics is fetched for the same
    using GET /v2/meters/(meter_name)/statistics.
PARAMETERS
+--------+-------------------------------------------------+
| name   | description                                     |
+--------+-------------------------------------------------+
| kwargs | contains optional arguments to create a meter   |
|        |                                                 |
+--------+-------------------------------------------------+
```

In case if multiple found benchmarks found command list all matches elements:

```
$ rally plugin show NovaKeypair

Multiple plugins found:
+-------------------------------------------------+-----------+-----------------------------------
| name                                            | namespace | title
+-------------------------------------------------+-----------+-----------------------------------
| NovaKeypair.boot_and_delete_server_with_keypair | default   | Boot and delete server with keypair.
| NovaKeypair.create_and_delete_keypair           | default   | Create a keypair with random name and
| NovaKeypair.create_and_list_keypairs            | default   | Create a keypair with random name and
+-------------------------------------------------+-----------+-----------------------------------
```

CLI: rally plugin list

This command can be used to list filtered by name list of plugins.

```
rally plugin list --name Keystone

+-------------------------------------------------+-----------+-----------------------------------
| name                                            | namespace | title
+-------------------------------------------------+-----------+-----------------------------------
| Authenticate.keystone                           | default   | Check Keystone Client.
| KeystoneBasic.add_and_remove_user_role          | default   | Create a user role add to a user and
| KeystoneBasic.create_add_and_list_user_roles    | default   | Create user role, add it and list us
| KeystoneBasic.create_and_delete_ec2credential   | default   | Create and delete keystone ec2-crede
| KeystoneBasic.create_and_delete_role            | default   | Create a user role and delete it.
| KeystoneBasic.create_and_delete_service         | default   | Create and delete service.
| KeystoneBasic.create_and_list_ec2credentials    | default   | Create and List all keystone ec2-cre
| KeystoneBasic.create_and_list_services          | default   | Create and list services.
| KeystoneBasic.create_and_list_tenants           | default   | Create a keystone tenant with randor
| KeystoneBasic.create_and_list_users             | default   | Create a keystone user with random n
| KeystoneBasic.create_delete_user                | default   | Create a keystone user with random n
| KeystoneBasic.create_tenant                     | default   | Create a keystone tenant with randor
| KeystoneBasic.create_tenant_with_users          | default   | Create a keystone tenant and several
| KeystoneBasic.create_update_and_delete_tenant   | default   | Create, update and delete tenant.
| KeystoneBasic.create_user                       | default   | Create a keystone user with random n
| KeystoneBasic.create_user_set_enabled_and_delete | default   | Create a keystone user, enable or di
| KeystoneBasic.create_user_update_password       | default   | Create user and update password for
```

```
| KeystoneBasic.get_entities                          | default   | Get instance of a tenant, user, role
+-----------------------------------------------------+-----------+-------------------------------------
```

1.3.10 Step 9. Deploying OpenStack from Rally

Along with supporting already existing OpenStack deployments, Rally itself can **deploy OpenStack automatically** by using one of its *deployment engines*. Take a look at other deployment configuration file samples. For example, *devstack-in-existing-servers.json* is a deployment configuration file that tells Rally to deploy OpenStack with **Devstack** on the existing servers with given credentials:

```
{
    "type": "DevstackEngine",
    "provider": {
        "type": "ExistingServers",
        "credentials": [{"user": "root", "host": "10.2.0.8"}]
    }
}
```

You can try to deploy OpenStack in your Virtual Machine using this script. Edit the configuration file with your IP address/user name and run, as usual:

```
$ rally deployment create --file=samples/deployments/for_deploying_openstack_with_rally/devstack-in-e
+--------------------------+--------------------------+--------------+-------------------+
|           uuid           |         created_at       |     name     |      status       |
+--------------------------+--------------------------+--------------+-------------------+
|     <Deployment UUID>    | 2015-01-10 22:00:28.270941 | new-devstack | deploy->finished |
+--------------------------+--------------------------+--------------+-------------------+
Using deployment : <Deployment UUID>
```

1.4 User stories

Many users of Rally were able to make interesting discoveries concerning their OpenStack clouds using our benchmarking tool. Numerous user stories presented below show how Rally has made it possible to find performance bugs and validate improvements for different OpenStack installations.

1.4.1 4x performance increase in Keystone inside Apache using the token creation benchmark

(Contributed by Neependra Khare, Red Hat)

Below we describe how we were able to get and verify a 4x better performance of Keystone inside Apache. To do that, we ran a Keystone token creation benchmark with Rally under different load (this benchmark scenario essentially just authenticate users with keystone to get tokens).

Goal

- Get the data about performance of token creation under different load.
- Ensure that keystone with increased public_workers/admin_workers values and under Apache works better than the default setup.

Summary

- As the concurrency increases, time to authenticate the user gets up.
- **Keystone is CPU bound process and by default only one thread of keystone-all process get started. We can increase the pa**

    1. increasing public_workers/admin_workers values in keystone.conf file
    2. running keystone inside Apache

- We configured Keystone with 4 public_workers and ran Keystone inside Apache. In both cases we got upto 4x better performance as compared to default keystone configuration.

Setup

Server : Dell PowerEdge R610

CPU make and model : Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

CPU count: 24

RAM : 48 GB

Devstack - Commit#d65f7a2858fb047b20470e8fa62ddaede2787a85

Keystone - Commit#455d50e8ae360c2a7598a61d87d9d341e5d9d3ed

Keystone API - 2

To increase public_workers - Uncomment line with public_workers and set public_workers to 4. Then restart keystone service.

To run keystone inside Apache - Added *APACHE_ENABLED_SERVICES=key* in localrc file while setting up OpenStack environment with devstack.

Results

1. Concurrency = 4

```
{'context': {'users': {'concurrent': 30,
                       'tenants': 12,
                       'users_per_tenant': 512}},
                       'runner': {'concurrency': 4, 'times': 10000, 'type': 'constant'}}
```

action	min (sec)	avg (sec)	max (sec)	90 percentile	95 percentile	success	count	apache enabled keystone	public_workers
total	0.537	0.998	4.553	1.233	1.391	100.0%	10000	N	1
total	0.189	0.296	5.099	0.417	0.474	100.0%	10000	N	4
total	0.208	0.299	3.228	0.437	0.485	100.0%	10000	Y	NA

2. Concurrency = 16

```
{'context': {'users': {'concurrent': 30,
                       'tenants': 12,
                       'users_per_tenant': 512}},
                       'runner': {'concurrency': 16, 'times': 10000, 'type': 'constant'}}
```

action	min (sec)	avg (sec)	max (sec)	90 percentile	95 percentile	success	count	apache enabled keystone	public_workers
total	1.036	3.905	11.254	5.258	5.700	100.0%	10000	N	1
total	0.187	1.012	5.894	1.61	1.856	100.0%	10000	N	4
total	0.515	0.970	2.076	1.113	1.192	100.0%	10000	Y	NA

3. Concurrency = 32

```
{'context': {'users': {'concurrent': 30,
                       'tenants': 12,
                       'users_per_tenant': 512}},
                       'runner': {'concurrency': 32, 'times': 10000, 'type': 'constant'}}
```

action	min (sec)	avg (sec)	max (sec)	90 percentile	95 percentile	success	count	apache enabled keystone	public_workers
total	1.493	7.752	16.007	10.428	11.183	100.0%	10000	N	1
total	0.198	1.967	8.54	3.223	3.701	100.0%	10000	N	4
total	1.115	1.986	6.224	2.133	2.244	100.0%	10000	Y	NA

1.4.2 Finding a Keystone bug while benchmarking 20 node HA cloud performance at creating 400 VMs

(Contributed by Alexander Maretskiy, Mirantis)

Below we describe how we found a bug in keystone and achieved 2x average performance increase at booting Nova servers after fixing that bug. Our initial goal was to benchmark the booting of a significant amount of servers on a cluster (running on a custom build of Mirantis OpenStack v5.1) and to ensure that this operation has reasonable performance and completes with no errors.

Goal

- Get data on how a cluster behaves when a huge amount of servers is started
- Get data on how good the neutron component is good in this case

Summary

- Creating 400 servers with configured networking
- Servers are being created simultaneously - 5 servers at the same time

Hardware

Having a real hardware lab with 20 nodes:

Vendor	SUPERMICRO SUPERSERVER
CPU	12 cores, Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz
RAM	32GB (4 x Samsung DDRIII 8GB)
HDD	1TB

Cluster

This cluster was created via Fuel Dashboard interface.

Deployment	Custom build of Mirantis OpenStack v5.1
OpenStack release	Icehouse
Operating System	Ubuntu 12.04.4
Mode	High availability
Hypervisor	KVM
Networking	Neutron with GRE segmentation
Controller nodes	3
Compute nodes	17

Rally

Version

For this benchmark, we use custom rally with the following patch:

https://review.openstack.org/#/c/96300/

Deployment

Rally was deployed for cluster using ExistingCloud type of deployment.

Server flavor

```
$ nova flavor-show ram64
+----------------------------+--------------------------------------+
| Property                   | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 0                                    |
| extra_specs                | {}                                   |
| id                         | 2e46aba0-9e7f-4572-8b0a-b12cfe7e06a1 |
| name                       | ram64                                |
| os-flavor-access:is_public | True                                 |
| ram                        | 64                                   |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 1                                    |
+----------------------------+--------------------------------------+
```

Server image

```
$ nova image-show TestVM
+----------------------------+-------------------------------------------------+
| Property                   | Value                                           |
+----------------------------+-------------------------------------------------+
| OS-EXT-IMG-SIZE:size       | 13167616                                        |
| created                    | 2014-08-21T11:18:49Z                            |
| id                         | 7a0d90cb-4372-40ef-b711-8f63b0ea9678            |
| metadata murano_image_info | {"title": "Murano Demo", "type": "cirros.demo"} |
| minDisk                    | 0                                               |
| minRam                     | 64                                              |
| name                       | TestVM                                          |
| progress                   | 100                                             |
| status                     | ACTIVE                                          |
```

```
| updated                      | 2014-08-21T11:18:50Z                                 |
+------------------------------+------------------------------------------------------+
```

Task configuration file (in JSON format):

```
{
   "NovaServers.boot_server": [
       {
           "args": {
               "flavor": {
                   "name": "ram64"
               },
               "image": {
                   "name": "TestVM"
               }
           },
           "runner": {
               "type": "constant",
               "concurrency": 5,
               "times": 400
           },
           "context": {
               "neutron_network": {
                   "network_ip_version": 4
               },
               "users": {
                   "concurrent": 30,
                   "users_per_tenant": 5,
                   "tenants": 5
               },
               "quotas": {
                   "neutron": {
                       "subnet": -1,
                       "port": -1,
                       "network": -1,
                       "router": -1
                   }
               }
           }
       }
   ]
}
```

The only difference between first and second run is that runner.times for first time was set to 500

Results

First time - a bug was found:

Starting from 142 server, we have error from novaclient: Error <class 'novaclient.exceptions.Unauthorized'>: Unauthorized (HTTP 401).

That is how a bug in keystone was found.

action	min (sec)	avg (sec)	max (sec)	90 percentile	95 percentile	success	count
nova.boot_server	6.507	17.402	100.303	39.222	50.134	26.8%	500
total	6.507	17.402	100.303	39.222	50.134	26.8%	500

Second run, with bugfix:

After a patch was applied (using RPC instead of neutron client in metadata agent), we got **100% success and 2x improved average perfomance**:

action	min (sec)	avg (sec)	max (sec)	90 percentile	95 percentile	success	count
nova.boot_server	5.031	8.008	14.093	9.616	9.716	100.0%	400
total	5.031	8.008	14.093	9.616	9.716	100.0%	400

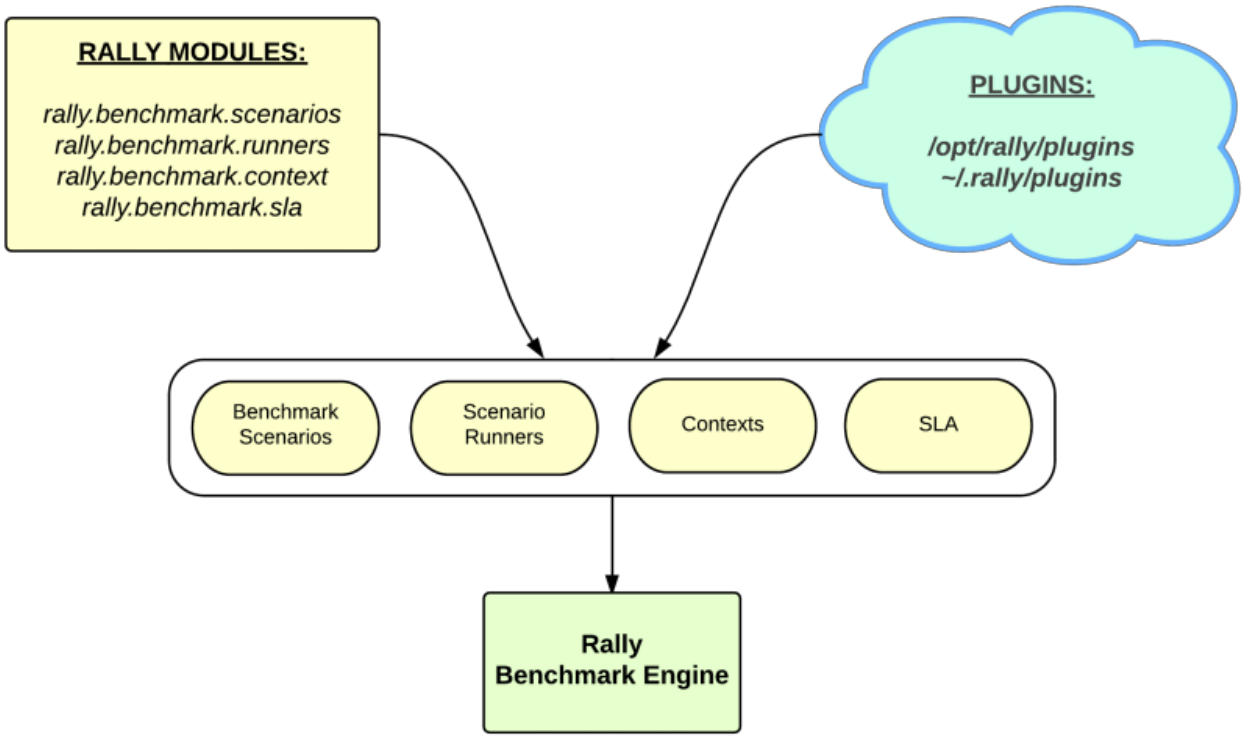
1.5 Rally Plugins

1.5.1 Rally Plugin Reference

Rally has a plugin oriented architecture - in other words Rally team is trying to make all places of code pluggable. Such architecture leds to the big amount of plugins. *Rally Plugins Reference page* contains a full list with detailed descriptions of all official Rally plugins.

1.5.2 How plugins work

Rally provides an opportunity to create and use a **custom benchmark scenario, runner or context** as a **plugin**:



Plugins can be quickly written and used, with no need to contribute them to the actual Rally code. Just place a python module with your plugin class into the **/opt/rally/plugins** or **~/.rally/plugins** directory (or it's subdirectories), and it will be autoloaded.

1.5.3 Example: Benchmark scenario as a plugin

Let's create as a plugin a simple scenario which list flavors.

Creation

Inherit a class for your plugin from the base *Scenario* class and implement a scenario method inside it as usual. In our scenario, let us first list flavors as an ordinary user, and then repeat the same using admin clients:

```
from rally.task import atomic
from rally.task import scenario


class ScenarioPlugin(scenario.Scenario):
    """Sample plugin which lists flavors."""

    @atomic.action_timer("list_flavors")
    def _list_flavors(self):
        """Sample of usage clients - list flavors

        You can use self.context, self.admin_clients and self.clients which are
        initialized on scenario instance creation"""
        self.clients("nova").flavors.list()

    @atomic.action_timer("list_flavors_as_admin")
    def _list_flavors_as_admin(self):
        """The same with admin clients"""
        self.admin_clients("nova").flavors.list()

    @scenario.configure()
    def list_flavors(self):
        """List flavors."""
        self._list_flavors()
        self._list_flavors_as_admin()
```

Placement

Put the python module with your plugin class into the **/opt/rally/plugins** or **~/.rally/plugins** directory or it's subdirectories and it will be autoloaded. You can also use a script **unpack_plugins_samples.sh** from **samples/plugins** which will automatically create the **~/.rally/plugins** directory.

Usage

You can refer to your plugin scenario in the benchmark task configuration files just in the same way as to any other scenarios:

```
{
    "ScenarioPlugin.list_flavors": [
        {
            "runner": {
                "type": "serial",
                "times": 5,
            },
            "context": {
                "create_flavor": {
```

```
                    "ram": 512,
                }
            }
        }
    ]
}
```

This configuration file uses the *"create_flavor"* context which we'll create as a plugin below.

1.5.4 Example: Context as a plugin

Let's create as a plugin a simple context which adds a flavor to the environment before the benchmark task starts and deletes it after it finishes.

Creation

Inherit a class for your plugin from the base *Context* class. Then, implement the Context API: the *setup()* method that creates a flavor and the *cleanup()* method that deletes it.

```
from rally.task import context
from rally.common import log as logging
from rally import consts
from rally import osclients

LOG = logging.getLogger(__name__)


@context.configure(name="create_flavor", order=1000)
class CreateFlavorContext(context.Context):
    """This sample create flavor with specified options before task starts and
    delete it after task completion.

    To create your own context plugin, inherit it from
    rally.task.context.Context
    """

    CONFIG_SCHEMA = {
        "type": "object",
        "$schema": consts.JSON_SCHEMA,
        "additionalProperties": False,
        "properties": {
            "flavor_name": {
                "type": "string",
            },
            "ram": {
                "type": "integer",
                "minimum": 1
            },
            "vcpus": {
                "type": "integer",
                "minimum": 1
            },
            "disk": {
                "type": "integer",
                "minimum": 1
            }
```

```
        }
    }

    def setup(self):
        """This method is called before the task start"""
        try:
            # use rally.osclients to get necessary client instance
            nova = osclients.Clients(self.context["admin"]["endpoint"]).nova()
            # and than do what you need with this client
            self.context["flavor"] = nova.flavors.create(
                # context settings are stored in self.config
                name=self.config.get("flavor_name", "rally_test_flavor"),
                ram=self.config.get("ram", 1),
                vcpus=self.config.get("vcpus", 1),
                disk=self.config.get("disk", 1)).to_dict()
            LOG.debug("Flavor with id '%s'" % self.context["flavor"]["id"])
        except Exception as e:
            msg = "Can't create flavor: %s" % e.message
            if logging.is_debug():
                LOG.exception(msg)
            else:
                LOG.warning(msg)

    def cleanup(self):
        """This method is called after the task finish"""
        try:
            nova = osclients.Clients(self.context["admin"]["endpoint"]).nova()
            nova.flavors.delete(self.context["flavor"]["id"])
            LOG.debug("Flavor '%s' deleted" % self.context["flavor"]["id"])
        except Exception as e:
            msg = "Can't delete flavor: %s" % e.message
            if logging.is_debug():
                LOG.exception(msg)
            else:
                LOG.warning(msg)
```

Placement

Put the python module with your plugin class into the **/opt/rally/plugins** or **~/.rally/plugins** directory or it's subdirectories and it will be autoloaded. You can also use a script **unpack_plugins_samples.sh** from **samples/plugins** which will automatically create the **~/.rally/plugins** directory.

Usage

You can refer to your plugin context in the benchmark task configuration files just in the same way as to any other contexts:

```
{
    "Dummy.dummy": [
        {
            "args": {
                "sleep": 0.01
            },
            "runner": {
                "type": "constant",
                "times": 5,
```

```
                "concurrency": 1
            },
            "context": {
                "users": {
                    "tenants": 1,
                    "users_per_tenant": 1
                },
                 "create_flavor": {
                    "ram": 1024
                }
            }
        }
    ]
}
```

1.5.5 Example: SLA as a plugin

Let's create as a plugin an SLA (success criterion) which checks whether the range of the observed performance measurements does not exceed the allowed maximum value.

Creation

Inherit a class for your plugin from the base *SLA* class and implement its API (the *add_iteration(iteration)*, the *details()* method):

```
from rally.task import sla
from rally.common.i18n import _

@sla.configure(name="max_duration_range")
class MaxDurationRange(sla.SLA):
    """Maximum allowed duration range in seconds."""

    CONFIG_SCHEMA = {
        "type": "number",
        "minimum": 0.0,
    }

    def __init__(self, criterion_value):
        super(MaxDurationRange, self).__init__(criterion_value)
        self._min = 0
        self._max = 0

    def add_iteration(self, iteration):
      # Skipping failed iterations (that raised exceptions)
        if iteration.get("error"):
            return self.success   # This field is defined in base class

        # Updating _min and _max values
        self._max = max(self._max, iteration["duration"])
        self._min = min(self._min, iteration["duration"])

        # Updating successfulness based on new max and min values
        self.success = self._max - self._min <= self.criterion_value
        return self.success

    def details(self):
```

```
        return (_("%s - Maximum allowed duration range: %.2f%% <= %.2f%%") %
                (self.status(), self._max - self._min, self.criterion_value))
```

Placement

Put the python module with your plugin class into the **/opt/rally/plugins** or **~/.rally/plugins** directory or it's subdirectories and it will be autoloaded. You can also use a script **unpack_plugins_samples.sh** from **samples/plugins** which will automatically create the **~/.rally/plugins** directory.

Usage

You can refer to your SLA in the benchmark task configuration files just in the same way as to any other SLA:

```
{
    "Dummy.dummy": [
        {
            "args": {
                "sleep": 0.01
            },
            "runner": {
                "type": "constant",
                "times": 5,
                "concurrency": 1
            },
            "context": {
                "users": {
                    "tenants": 1,
                    "users_per_tenant": 1
                }
            },
            "sla": {
                "max_duration_range": 2.5
            }
        }
    ]
}
```

1.5.6 Example: Scenario runner as a plugin

Let's create as a plugin a scenario runner which runs a given benchmark scenario for a random number of times (chosen at random from a given range).

Creation

Inherit a class for your plugin from the base *ScenarioRunner* class and implement its API (the *_run_scenario()* method):

```
import random

from rally.task import runner
from rally import consts
```

```
@runner.configure(name="random_times")
class RandomTimesScenarioRunner(runner.ScenarioRunner):
    """Sample of scenario runner plugin.

    Run scenario random number of times, which is chosen between min_times and
    max_times.
    """

    CONFIG_SCHEMA = {
        "type": "object",
        "$schema": consts.JSON_SCHEMA,
        "properties": {
            "type": {
                "type": "string"
            },
            "min_times": {
                "type": "integer",
                "minimum": 1
            },
            "max_times": {
                "type": "integer",
                "minimum": 1
            }
        },
        "additionalProperties": True
    }

    def _run_scenario(self, cls, method_name, context, args):
        # runners settings are stored in self.config
        min_times = self.config.get('min_times', 1)
        max_times = self.config.get('max_times', 1)

        for i in range(random.randrange(min_times, max_times)):
            run_args = (i, cls, method_name,
                        runner._get_scenario_context(context), args)
            result = runner._run_scenario_once(run_args)
            # use self.send_result for result of each iteration
            self._send_result(result)
```

Placement

Put the python module with your plugin class into the **/opt/rally/plugins** or **~/.rally/plugins** directory or it's subdirectories and it will be autoloaded. You can also use a script **unpack_plugins_samples.sh** from **samples/plugins** which will automatically create the **~/.rally/plugins** directory.

Usage

You can refer to your scenario runner in the benchmark task configuration files just in the same way as to any other runners. Don't forget to put you runner-specific parameters to the configuration as well (*"min_times"* and *"max_times"* in our example):

```
{
    "Dummy.dummy": [
        {
            "runner": {
                "type": "random_times",
```

```
                "min_times": 10,
                "max_times": 20,
            },
            "context": {
                "users": {
                    "tenants": 1,
                    "users_per_tenant": 1
                }
            }
        }
    ]
}
```

Different plugin samples are available here.

1.6 Rally Plugins Reference

Scenario Runners [task]

constant [scenario runner]

```
Creates constant load executing a scenario a specified number of times.

This runner will place a constant load on the cloud under test by
executing each scenario iteration without pausing between iterations
up to the number of times specified in the scenario config.

The concurrency parameter of the scenario config controls the
number of concurrent scenarios which execute during a single
iteration in order to simulate the activities of multiple users
placing load on the cloud under test.

MODULE:
rally.plugins.common.runners.constant
```

constant_for_duration [scenario runner]

```
Creates constant load executing a scenario for an interval of time.

This runner will place a constant load on the cloud under test by
executing each scenario iteration without pausing between iterations
until a specified interval of time has elapsed.

The concurrency parameter of the scenario config controls the
number of concurrent scenarios which execute during a single
iteration in order to simulate the activities of multiple users
placing load on the cloud under test.

MODULE:
rally.plugins.common.runners.constant
```

rps [scenario runner]

```
Scenario runner that does the job with specified frequency.

Every single benchmark scenario iteration is executed with specified
frequency (runs per second) in a pool of processes. The scenario will be
launched for a fixed number of times in total (specified in the config).

An example of a rps scenario is booting 1 VM per second. This
execution type is thus very helpful in understanding the maximal load that
a certain cloud can handle.

MODULE:
rally.plugins.common.runners.rps
```

serial [scenario runner]

```
Scenario runner that executes benchmark scenarios serially.

Unlike scenario runners that execute in parallel, the serial scenario
runner executes scenarios one-by-one in the same python interpreter process
as Rally. This allows you to benchmark your scenario without introducing
any concurrent operations as well as interactively debug the scenario
from the same command that you use to start Rally.

MODULE:
rally.plugins.common.runners.serial
```

SLAs [task]

outliers [SLA]

```
Limit the number of outliers (iterations that take too much time).

The outliers are detected automatically using the computation of the mean
and standard deviation (std) of the data.

MODULE:
rally.plugins.common.sla.outliers
```

max_avg_duration [SLA]

```
Maximum average duration of one iteration in seconds.

MODULE:
rally.plugins.common.sla.max_average_duration
```

failure_rate [SLA]

```
Failure rate minimum and maximum in percents.
```

```
MODULE:
rally.plugins.common.sla.failure_rate
```

max_seconds_per_iteration [SLA]

```
Maximum time for one iteration in seconds.

MODULE:
rally.plugins.common.sla.iteraion_time
```

Contexts [task]

flavors [context]

```
Context creates a list of flavors.

MODULE:
rally.plugins.openstack.context.nova.flavors
```

roles [context]

```
Context class for adding temporary roles for benchmarks.

MODULE:
rally.plugins.openstack.context.keystone.roles
```

users [context]

```
Context class for generating temporary users/tenants for benchmarks.

MODULE:
rally.plugins.openstack.context.keystone.users
```

existing_users [context]

```
This context supports using existing users in Rally.

It uses information about deployment to properly
initialize context[''users''] and context[''tenants'']

So there won't be big difference between usage of ``users'' and
``existing_users'' context.

MODULE:
rally.plugins.openstack.context.keystone.existing_users
```

sahara_cluster [context]

Context class for setting up the Cluster an EDP job.

MODULE:
rally.plugins.openstack.context.sahara.sahara_cluster

sahara_image [context]

Context class for adding and tagging Sahara images.

MODULE:
rally.plugins.openstack.context.sahara.sahara_image

sahara_edp [context]

Context class for setting up the environment for an EDP job.

MODULE:
rally.plugins.openstack.context.sahara.sahara_edp

admin_cleanup [context]

Context class for admin resources cleanup.

MODULE:
rally.plugins.openstack.context.cleanup.context

cleanup [context]

Context class for user resources cleanup.

MODULE:
rally.plugins.openstack.context.cleanup.context

murano_packages [context]

Context class for uploading applications for murano.

MODULE:
rally.plugins.openstack.context.murano.murano_packages

ceilometer [context]

Context for creating samples and collecting resources for benchmarks.

MODULE:
rally.plugins.openstack.context.ceilometer.samples

lbaas [context]

MODULE:
rally.plugins.openstack.context.neutron.lbaas

manila_share_networks [context]

This context creates resources specific for Manila project.

MODULE:
rally.plugins.openstack.context.manila.manila_share_networks

ec2_servers [context]

Context class for adding temporary servers for benchmarks.

Servers are added for each tenant.

MODULE:
rally.plugins.openstack.context.ec2.servers

keypair [context]

MODULE:
rally.plugins.openstack.context.nova.keypairs

servers [context]

Context class for adding temporary servers for benchmarks.

Servers are added for each tenant.

MODULE:
rally.plugins.openstack.context.nova.servers

quotas [context]

Context class for updating benchmarks' tenants quotas.

MODULE:
rally.plugins.openstack.context.quotas.quotas

tempest [context]

MODULE:
rally.plugins.openstack.context.not_for_production.tempest

images [context]

Context class for adding images to each user for benchmarks.

MODULE:
rally.plugins.openstack.context.glance.images

existing_network [context]

This context supports using existing networks in Rally.

This context should be used on a deployment with existing users.

MODULE:
rally.plugins.openstack.context.network.existing_network

network [context]

MODULE:
rally.plugins.openstack.context.network.networks

allow_ssh [context]

Sets up security groups for all users to access VM via SSH.

MODULE:
rally.plugins.openstack.context.network.allow_ssh

stacks [context]

Context class for create temporary stacks with resources.

Stack generator allows to generate arbitrary number of stacks for
each tenant before test scenarios. In addition, it allows to define
number of resources (namely OS::Heat::RandomString) that will be created
inside each stack. After test execution the stacks will be
automatically removed from heat.

MODULE:
rally.plugins.openstack.context.heat.stacks

dummy_context [context]

```
Dummy context.

MODULE:
rally.plugins.common.context.dummy
```

custom_image [context]

```
Base class for the contexts providing customized image with.

Every context class for the specific customization must implement
the method `_customize_image` that is able to connect to the server
using SSH and e.g. install applications inside it.

This is used e.g. to install the benchmark application using SSH
access.

This base context class provides a way to prepare an image with
custom preinstalled applications. Basically, this code boots a VM, calls
the `_customize_image` and then snapshots the VM disk, removing the VM
afterwards. The image UUID is stored in the user[''custom_image''][''id'']
and can be used afterwards by scenario.

MODULE:
rally.plugins.openstack.context.vm.custom_image
```

image_command_customizer [context]

```
Context class for generating image customized by a command execution.

Run a command specified by configuration to prepare image.

Use this script e.g. to download and install something.

MODULE:
rally.plugins.openstack.context.vm.image_command_customizer
```

volumes [context]

```
Context class for adding volumes to each user for benchmarks.

MODULE:
rally.plugins.openstack.context.cinder.volumes
```

Scenarios [task]

KeystoneBasic.create_user [scenario]

```
Create a keystone user with random name.
```

```
PARAMETERS:
* name_length: length of the random part of user name
* kwargs: Other optional parameters to create users like
``tenant_id'', ``enabled''.


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_delete_user [scenario]

```
Create a keystone user with random name and then delete it.

PARAMETERS:
* name_length: length of the random part of user name
* kwargs: Other optional parameters to create users like
``tenant_id'', ``enabled''.


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_user_set_enabled_and_delete [scenario]

```
Create a keystone user, enable or disable it, and delete it.

PARAMETERS:
* enabled: Initial state of user `enabled' flag. The user
will be created with `enabled' set to this
value, and then it will be toggled.
* kwargs: Other optional parameters to create user.


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_tenant [scenario]

```
Create a keystone tenant with random name.

PARAMETERS:
* name_length: length of the random part of tenant name
* kwargs: Other optional parameters


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_tenant_with_users [scenario]

```
Create a keystone tenant and several users belonging to it.

PARAMETERS:
* name_length: length of the random part of tenant/user name
* users_per_tenant: number of users to create for the tenant
* kwargs: Other optional parameters for tenant creation


RETURNS:
keystone tenant instance

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_list_users [scenario]

```
Create a keystone user with random name and list all users.

PARAMETERS:
* name_length: length of the random part of user name
* kwargs: Other optional parameters to create users like
``tenant_id'', ``enabled''.


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_list_tenants [scenario]

```
Create a keystone tenant with random name and list all tenants.

PARAMETERS:
* name_length: length of the random part of tenant name
* kwargs: Other optional parameters


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.add_and_remove_user_role [scenario]

```
Create a user role add to a user and disassociate.

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_delete_role [scenario]

```
Create a user role and delete it.

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_add_and_list_user_roles [scenario]

```
Create user role, add it and list user roles for given user.

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.get_entities [scenario]

```
Get instance of a tenant, user, role and service by id's.

An ephemeral tenant, user, and role are each created. By
default, fetches the `keystone' service. This can be
overridden (for instance, to get the `Identity Service'
service on older OpenStack), or None can be passed explicitly
to service_name to create a new service and then query it by
ID.

PARAMETERS:
* service_name: The name of the service to get by ID; or
None, to create an ephemeral service and
get it by ID.


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_delete_service [scenario]

```
Create and delete service.

PARAMETERS:
* service_type: type of the service
* description: description of the service


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_update_and_delete_tenant [scenario]

```
Create, update and delete tenant.
```

```
PARAMETERS:
* name_length: length of the random part of tenant name
* kwargs: Other optional parameters for tenant creation


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_user_update_password [scenario]

```
Create user and update password for that user.

PARAMETERS:
* name_length: length of the user name
* password_length: length of the password


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_list_services [scenario]

```
Create and list services.

PARAMETERS:
* service_type: type of the service
* description: description of the service


MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_list_ec2credentials [scenario]

```
Create and List all keystone ec2-credentials.

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

KeystoneBasic.create_and_delete_ec2credential [scenario]

```
Create and delete keystone ec2-credential.

MODULE:
rally.plugins.openstack.scenarios.keystone.basic
```

Authenticate.keystone [scenario]

```
Check Keystone Client.
```

```
MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

Authenticate.validate_glance [scenario]

```
Check Glance Client to ensure validation of token.

Creation of the client does not ensure validation of the token.
We have to do some minimal operation to make sure token gets validated.
In following we are checking for non-existent image.

PARAMETERS:
* repetitions: number of times to validate


MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

Authenticate.validate_nova [scenario]

```
Check Nova Client to ensure validation of token.

Creation of the client does not ensure validation of the token.
We have to do some minimal operation to make sure token gets validated.

PARAMETERS:
* repetitions: number of times to validate


MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

Authenticate.validate_cinder [scenario]

```
Check Cinder Client to ensure validation of token.

Creation of the client does not ensure validation of the token.
We have to do some minimal operation to make sure token gets validated.

PARAMETERS:
* repetitions: number of times to validate


MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

Authenticate.validate_neutron [scenario]

```
Check Neutron Client to ensure validation of token.
```

```
Creation of the client does not ensure validation of the token.
We have to do some minimal operation to make sure token gets validated.

PARAMETERS:
* repetitions: number of times to validate


MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

Authenticate.validate_heat [scenario]

```
Check Heat Client to ensure validation of token.

Creation of the client does not ensure validation of the token.
We have to do some minimal operation to make sure token gets validated.

PARAMETERS:
* repetitions: number of times to validate


MODULE:
rally.plugins.openstack.scenarios.authenticate.authenticate
```

SaharaClusters.create_and_delete_cluster [scenario]

```
Launch and delete a Sahara Cluster.

This scenario launches a Hadoop cluster, waits until it becomes
`Active' and deletes it.

PARAMETERS:
* flavor: Nova flavor that will be for nodes in the
created node groups
* workers_count: number of worker instances in a cluster
* plugin_name: name of a provisioning plugin
* hadoop_version: version of Hadoop distribution supported by
the specified plugin.
* floating_ip_pool: floating ip pool name from which Floating
IPs will be allocated. Sahara will determine
automatically how to treat this depending on
it's own configurations. Defaults to None
because in some cases Sahara may work w/o
Floating IPs.
* volumes_per_node: number of Cinder volumes that will be
attached to every cluster node
* volumes_size: size of each Cinder volume in GB
* auto_security_group: boolean value. If set to True Sahara will
create a Security Group for each Node Group
in the Cluster automatically.
* security_groups: list of security groups that will be used
```

```
while creating VMs. If auto_security_group
is set to True, this list can be left empty.
* node_configs: config dict that will be passed to each Node
Group
* cluster_configs: config dict that will be passed to the
Cluster
* enable_anti_affinity: If set to true the vms will be scheduled
one per compute node.


MODULE:
rally.plugins.openstack.scenarios.sahara.clusters
```

SaharaClusters.create_scale_delete_cluster [scenario]

```
Launch, scale and delete a Sahara Cluster.

This scenario launches a Hadoop cluster, waits until it becomes
`Active'. Then a series of scale operations is applied. The scaling
happens according to numbers listed in

PARAMETERS:
* flavor: Nova flavor that will be for nodes in the
created node groups
* workers_count: number of worker instances in a cluster
* plugin_name: name of a provisioning plugin
* hadoop_version: version of Hadoop distribution supported by
the specified plugin.
* deltas: list of integers which will be used to add or
remove worker nodes from the cluster
* floating_ip_pool: floating ip pool name from which Floating
IPs will be allocated. Sahara will determine
automatically how to treat this depending on
it's own configurations. Defaults to None
because in some cases Sahara may work w/o
Floating IPs.
* neutron_net_id: id of a Neutron network that will be used
for fixed IPs. This parameter is ignored when
Nova Network is set up.
* volumes_per_node: number of Cinder volumes that will be
attached to every cluster node
* volumes_size: size of each Cinder volume in GB
* auto_security_group: boolean value. If set to True Sahara will
create a Security Group for each Node Group
in the Cluster automatically.
* security_groups: list of security groups that will be used
while creating VMs. If auto_security_group
is set to True this list can be left empty.
* node_configs: configs dict that will be passed to each Node
Group
* cluster_configs: configs dict that will be passed to the
Cluster
* enable_anti_affinity: If set to true the vms will be scheduled
```

```
one per compute node.


MODULE:
rally.plugins.openstack.scenarios.sahara.clusters
```

SaharaJob.create_launch_job [scenario]

```
Create and execute a Sahara EDP Job.

This scenario Creates a Job entity and launches an execution on a
Cluster.

PARAMETERS:
* job_type: type of the Data Processing Job
* configs: config dict that will be passed to a Job Execution
* job_idx: index of a job in a sequence. This index will be
used to create different atomic actions for each job
in a sequence


MODULE:
rally.plugins.openstack.scenarios.sahara.jobs
```

SaharaJob.create_launch_job_sequence [scenario]

```
Create and execute a sequence of the Sahara EDP Jobs.

This scenario Creates a Job entity and launches an execution on a
Cluster for every job object provided.

PARAMETERS:
* jobs: list of jobs that should be executed in one context


MODULE:
rally.plugins.openstack.scenarios.sahara.jobs
```

SaharaJob.create_launch_job_sequence_with_scaling [scenario]

```
Create and execute Sahara EDP Jobs on a scaling Cluster.

This scenario Creates a Job entity and launches an execution on a
Cluster for every job object provided. The Cluster is scaled according
to the deltas values and the sequence is launched again.

PARAMETERS:
* jobs: list of jobs that should be executed in one context
* deltas: list of integers which will be used to add or
remove worker nodes from the cluster
```

```
MODULE:
rally.plugins.openstack.scenarios.sahara.jobs
```

SaharaNodeGroupTemplates.create_and_list_node_group_templates [scenario]

```
Create and list Sahara Node Group Templates.

This scenario creates two Node Group Templates with different set of
node processes. The master Node Group Template contains Hadoop's
management processes. The worker Node Group Template contains
Hadoop's worker processes.

By default the templates are created for the vanilla Hadoop
provisioning plugin using the version 1.2.1

After the templates are created the list operation is called.

PARAMETERS:
* flavor: Nova flavor that will be for nodes in the
created node groups
* plugin_name: name of a provisioning plugin
* hadoop_version: version of Hadoop distribution supported by
the specified plugin.


MODULE:
rally.plugins.openstack.scenarios.sahara.node_group_templates
```

SaharaNodeGroupTemplates.create_delete_node_group_templates [scenario]

```
Create and delete Sahara Node Group Templates.

This scenario creates and deletes two most common types of
Node Group Templates.

By default the templates are created for the vanilla Hadoop
provisioning plugin using the version 1.2.1

PARAMETERS:
* flavor: Nova flavor that will be for nodes in the
created node groups
* plugin_name: name of a provisioning plugin
* hadoop_version: version of Hadoop distribution supported by
the specified plugin.


MODULE:
rally.plugins.openstack.scenarios.sahara.node_group_templates
```

MistralWorkbooks.list_workbooks [scenario]

```
Scenario test mistral workbook-list command.

This simple scenario tests the Mistral workbook-list
command by listing all the workbooks.

MODULE:
rally.plugins.openstack.scenarios.mistral.workbooks
```

MistralWorkbooks.create_workbook [scenario]

```
Scenario tests workbook creation and deletion.

This scenario is a very useful tool to measure the
``mistral workbook-create'' and ``mistral workbook-delete''
commands performance.

PARAMETERS:
* definition: string (yaml string) representation of given
file content (Mistral workbook definition)
* do_delete: if False than it allows to check performance
in ``create only'' mode.


MODULE:
rally.plugins.openstack.scenarios.mistral.workbooks
```

TempestScenario.single_test [scenario]

```
Launch a single Tempest test by its name.

PARAMETERS:
* test_name: name of tempest scenario for launching
* log_file: name of file for junitxml results
* tempest_conf: User specified tempest.conf location


MODULE:
rally.plugins.openstack.scenarios.tempest.tempest
```

TempestScenario.all [scenario]

```
Launch all discovered Tempest tests by their names.

PARAMETERS:
* log_file: name of file for junitxml results
* tempest_conf: User specified tempest.conf location


MODULE:
```

```
rally.plugins.openstack.scenarios.tempest.tempest
```

TempestScenario.set [scenario]

```
Launch all Tempest tests from a given set.

PARAMETERS:
* set_name: set name of tempest scenarios for launching
* log_file: name of file for junitxml results
* tempest_conf: User specified tempest.conf location


MODULE:
rally.plugins.openstack.scenarios.tempest.tempest
```

TempestScenario.list_of_tests [scenario]

```
Launch all Tempest tests from a given list of their names.

PARAMETERS:
* test_names: list of tempest scenarios for launching
* log_file: name of file for junitxml results
* tempest_conf: User specified tempest.conf location


MODULE:
rally.plugins.openstack.scenarios.tempest.tempest
```

TempestScenario.specific_regex [scenario]

```
Launch Tempest tests whose names match a given regular expression.

PARAMETERS:
* regex: regexp to match Tempest test names against
* log_file: name of file for junitxml results
* tempest_conf: User specified tempest.conf location


MODULE:
rally.plugins.openstack.scenarios.tempest.tempest
```

SwiftObjects.create_container_and_object_then_list_objects [scenario]

```
Create container and objects then list all objects.

PARAMETERS:
* objects_per_container: int, number of objects to upload
* object_size: int, temporary local object size
* kwargs: dict, optional parameters to create container
```

```
MODULE:
rally.plugins.openstack.scenarios.swift.objects
```

SwiftObjects.create_container_and_object_then_delete_all [scenario]

```
Create container and objects then delete everything created.

PARAMETERS:
* objects_per_container: int, number of objects to upload
* object_size: int, temporary local object size
* kwargs: dict, optional parameters to create container


MODULE:
rally.plugins.openstack.scenarios.swift.objects
```

SwiftObjects.create_container_and_object_then_download_object [scenario]

```
Create container and objects then download all objects.

PARAMETERS:
* objects_per_container: int, number of objects to upload
* object_size: int, temporary local object size
* kwargs: dict, optional parameters to create container


MODULE:
rally.plugins.openstack.scenarios.swift.objects
```

MuranoEnvironments.list_environments [scenario]

```
List the murano environments.

Run murano environment-list for listing all environments.

MODULE:
rally.plugins.openstack.scenarios.murano.environments
```

MuranoEnvironments.create_and_delete_environment [scenario]

```
Create environment, session and delete environment.

MODULE:
rally.plugins.openstack.scenarios.murano.environments
```

MuranoEnvironments.create_and_deploy_environment [scenario]

```
Create environment, session and deploy environment.
```

```
Create environment, create session, add app to environment
packages_per_env times, send environment to deploy.

PARAMETERS:
* packages_per_env: number of packages per environment


MODULE:
rally.plugins.openstack.scenarios.murano.environments
```

ZaqarBasic.create_queue [scenario]

```
Create a Zaqar queue with a random name.

PARAMETERS:
* name_length: length of generated (random) part of name
* kwargs: other optional parameters to create queues like
``metadata''


MODULE:
rally.plugins.openstack.scenarios.zaqar.basic
```

ZaqarBasic.producer_consumer [scenario]

```
Serial message producer/consumer.

Creates a Zaqar queue with random name, sends a set of messages
and then retrieves an iterator containing those.

PARAMETERS:
* name_length: length of generated (random) part of name
* min_msg_count: min number of messages to be posted
* max_msg_count: max number of messages to be posted
* kwargs: other optional parameters to create queues like
``metadata''


MODULE:
rally.plugins.openstack.scenarios.zaqar.basic
```

DesignateBasic.create_and_list_domains [scenario]

```
Create a domain and list all domains.

Measure the ``designate domain-list'' command performance.

If you have only 1 user in your context, you will
add 1 domain on every iteration. So you will have more
and more domain and will be able to measure the
```

```
performance of the ``designate domain-list'' command depending on
the number of domains owned by users.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.list_domains [scenario]

```
List Designate domains.

This simple scenario tests the designate domain-list command by listing
all the domains.

Suppose if we have 2 users in context and each has 2 domains
uploaded for them we will be able to test the performance of
designate domain-list command in this case.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.create_and_delete_domain [scenario]

```
Create and then delete a domain.

Measure the performance of creating and deleting domains
with different level of load.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.create_and_delete_records [scenario]

```
Create and then delete records.

Measure the performance of creating and deleting records
with different level of load.

PARAMETERS:
* records_per_domain: Records to create pr domain.


MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.list_records [scenario]

```
List Designate records.

This simple scenario tests the designate record-list command by listing
all the records in a domain.
```

```
Suppose if we have 2 users in context and each has 2 domains
uploaded for them we will be able to test the performance of
designate record-list command in this case.

PARAMETERS:
* domain_id: Domain ID


MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.create_and_list_records [scenario]

```
Create and then list records.

If you have only 1 user in your context, you will
add 1 record on every iteration. So you will have more
and more records and will be able to measure the
performance of the ``designate record-list'' command depending on
the number of domains/records owned by users.

PARAMETERS:
* records_per_domain: Records to create pr domain.


MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.create_and_list_servers [scenario]

```
Create a Designate server and list all servers.

If you have only 1 user in your context, you will
add 1 server on every iteration. So you will have more
and more server and will be able to measure the
performance of the ``designate server-list'' command depending on
the number of servers owned by users.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.create_and_delete_server [scenario]

```
Create and then delete a server.

Measure the performance of creating and deleting servers
with different level of load.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

DesignateBasic.list_servers [scenario]

```
List Designate servers.

This simple scenario tests the designate server-list command by listing
all the servers.

MODULE:
rally.plugins.openstack.scenarios.designate.basic
```

CeilometerMeters.list_meters [scenario]

```
Fetch user's meters.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.meters
```

CeilometerSamples.list_samples [scenario]

```
Fetch all samples.

This scenario fetches list of all samples.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.samples
```

CeilometerTraits.create_user_and_list_traits [scenario]

```
Create user and fetch all event traits.

This scenario creates user to store new event and
fetches list of all traits for certain event type and
trait name using GET /v2/event_types/<event_type>/traits/<trait_name>.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.traits
```

CeilometerTraits.create_user_and_list_trait_descriptions [scenario]

```
Create user and fetch all trait descriptions.

This scenario creates user to store new event and
fetches list of all traits for certain event type using
GET /v2/event_types/<event_type>/traits.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.traits
```

CeilometerEvents.create_user_and_list_events [scenario]

```
Create user and fetch all events.

This scenario creates user to store new event and
fetches list of all events using GET /v2/events.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.events
```

CeilometerEvents.create_user_and_list_event_types [scenario]

```
Create user and fetch all event types.

This scenario creates user to store new event and
fetches list of all events types using GET /v2/event_types.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.events
```

CeilometerEvents.create_user_and_get_event [scenario]

```
Create user and gets event.

This scenario creates user to store new event and
fetches one event using GET /v2/events/<message_id>.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.events
```

CeilometerResource.list_resources [scenario]

```
Fetch all resources.

This scenario fetches list of all resources using GET /v2/resources.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.resources
```

CeilometerResource.get_tenant_resources [scenario]

```
Get all tenant resources.

This scenario retrieves information about tenant resources using
GET /v2/resources/(resource_id)

MODULE:
rally.plugins.openstack.scenarios.ceilometer.resources
```

CeilometerStats.create_meter_and_get_stats [scenario]

```
Create a meter and fetch its statistics.

Meter is first created and then statistics is fetched for the same
using GET /v2/meters/(meter_name)/statistics.

PARAMETERS:
* kwargs: contains optional arguments to create a meter


MODULE:
rally.plugins.openstack.scenarios.ceilometer.stats
```

CeilometerAlarms.create_alarm [scenario]

```
Create an alarm.

This scenarios test POST /v2/alarms.
meter_name and threshold are required parameters for alarm creation.
kwargs stores other optional parameters like `ok_actions',
`project_id' etc that may be passed while creating an alarm.

PARAMETERS:
* meter_name: specifies meter name of the alarm
* threshold: specifies alarm threshold
* kwargs: specifies optional arguments for alarm creation.


MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerAlarms.list_alarms [scenario]

```
Fetch all alarms.

This scenario fetches list of all alarms using GET /v2/alarms.

MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerAlarms.create_and_list_alarm [scenario]

```
Create and get the newly created alarm.

This scenarios test GET /v2/alarms/(alarm_id)
Initially alarm is created and then the created alarm is fetched using
its alarm_id. meter_name and threshold are required parameters
for alarm creation. kwargs stores other optional parameters like
`ok_actions', `project_id' etc. that may be passed while creating
an alarm.
```

```
PARAMETERS:
* meter_name: specifies meter name of the alarm
* threshold: specifies alarm threshold
* kwargs: specifies optional arguments for alarm creation.


MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerAlarms.create_and_update_alarm [scenario]

```
Create and update the newly created alarm.

This scenarios test PUT /v2/alarms/(alarm_id)
Initially alarm is created and then the created alarm is updated using
its alarm_id. meter_name and threshold are required parameters
for alarm creation. kwargs stores other optional parameters like
`ok_actions', `project_id' etc that may be passed while alarm creation.

PARAMETERS:
* meter_name: specifies meter name of the alarm
* threshold: specifies alarm threshold
* kwargs: specifies optional arguments for alarm creation.


MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerAlarms.create_and_delete_alarm [scenario]

```
Create and delete the newly created alarm.

This scenarios test DELETE /v2/alarms/(alarm_id)
Initially alarm is created and then the created alarm is deleted using
its alarm_id. meter_name and threshold are required parameters
for alarm creation. kwargs stores other optional parameters like
`ok_actions', `project_id' etc that may be passed while alarm creation.

PARAMETERS:
* meter_name: specifies meter name of the alarm
* threshold: specifies alarm threshold
* kwargs: specifies optional arguments for alarm creation.


MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerAlarms.create_alarm_and_get_history [scenario]

```
Create an alarm, get and set the state and get the alarm history.
```

```
This scenario makes following queries:
GET /v2/alarms/{alarm_id}/history
GET /v2/alarms/{alarm_id}/state
PUT /v2/alarms/{alarm_id}/state
Initially alarm is created and then get the state of the created alarm
using its alarm_id. Then get the history of the alarm. And finally the
state of the alarm is updated using given state. meter_name and
threshold are required parameters for alarm creation. kwargs stores
other optional parameters like `ok_actions', `project_id' etc that may
be passed while alarm creation.

PARAMETERS:
* meter_name: specifies meter name of the alarm
* threshold: specifies alarm threshold
* state: an alarm state to be set
* timeout: The number of seconds for which to attempt a
successful check of the alarm state
* kwargs: specifies optional arguments for alarm creation.


MODULE:
rally.plugins.openstack.scenarios.ceilometer.alarms
```

CeilometerQueries.create_and_query_alarms [scenario]

```
Create an alarm and then query it with specific parameters.

This scenario tests POST /v2/query/alarms
An alarm is first created and then fetched using the input query.

PARAMETERS:
* meter_name: specifies meter name of alarm
* threshold: specifies alarm threshold
* filter: optional filter query dictionary
* orderby: optional param for specifying ordering of results
* limit: optional param for maximum number of results returned
* kwargs: optional parameters for alarm creation


MODULE:
rally.plugins.openstack.scenarios.ceilometer.queries
```

CeilometerQueries.create_and_query_alarm_history [scenario]

```
Create an alarm and then query for its history.

This scenario tests POST /v2/query/alarms/history
An alarm is first created and then its alarm_id is used to fetch the
history of that specific alarm.

PARAMETERS:
```

```
* meter_name: specifies meter name of alarm
* threshold: specifies alarm threshold
* orderby: optional param for specifying ordering of results
* limit: optional param for maximum number of results returned
* kwargs: optional parameters for alarm creation


MODULE:
rally.plugins.openstack.scenarios.ceilometer.queries
```

CeilometerQueries.create_and_query_samples [scenario]

```
Create a sample and then query it with specific parameters.

This scenario tests POST /v2/query/samples
A sample is first created and then fetched using the input query.

PARAMETERS:
* counter_name: specifies name of the counter
* counter_type: specifies type of the counter
* counter_unit: specifies unit of the counter
* counter_volume: specifies volume of the counter
* resource_id: specifies resource id for the sample created
* filter: optional filter query dictionary
* orderby: optional param for specifying ordering of results
* limit: optional param for maximum number of results returned
* kwargs: parameters for sample creation


MODULE:
rally.plugins.openstack.scenarios.ceilometer.queries
```

NeutronLoadbalancerV1.create_and_list_pools [scenario]

```
Create a pool(v1) and then list pools(v1).

Measure the ``neutron lb-pool-list'' command performance.
The scenario creates a pool for every subnet and then lists pools.

PARAMETERS:
* pool_create_args: dict, POST /lb/pools request options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronLoadbalancerV1.create_and_delete_pools [scenario]

```
Create pools(v1) and delete pools(v1).

Measure the ``neutron lb-pool-create'' and ``neutron lb-pool-delete''
```

```
command performance. The scenario creates a pool for every subnet
and then deletes those pools.

PARAMETERS:
* pool_create_args: dict, POST /lb/pools request options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronLoadbalancerV1.create_and_update_pools [scenario]

```
Create pools(v1) and update pools(v1).

Measure the ``neutron lb-pool-create'' and ``neutron lb-pool-update''
command performance. The scenario creates a pool for every subnet
and then update those pools.

PARAMETERS:
* pool_create_args: dict, POST /lb/pools request options
* pool_update_args: dict, POST /lb/pools update options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronLoadbalancerV1.create_and_list_vips [scenario]

```
Create a vip(v1) and then list vips(v1).

Measure the ``neutron lb-vip-create'' and ``neutron lb-vip-list'' command
performance. The scenario creates a vip for every pool created and
then lists vips.

PARAMETERS:
* vip_create_args: dict, POST /lb/vips request options
* pool_create_args: dict, POST /lb/pools request options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronLoadbalancerV1.create_and_delete_vips [scenario]

```
Create a vip(v1) and then delete vips(v1).

Measure the ``neutron lb-vip-create'' and ``neutron lb-vip-delete''
command performance. The scenario creates a vip for pool and
then deletes those vips.

PARAMETERS:
```

```
* vip_create_args: dict, POST /lb/vips request options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronLoadbalancerV1.create_and_update_vips [scenario]

```
Create vips(v1) and update vips(v1).

Measure the ``neutron lb-vip-create'' and ``neutron lb-vip-update''
command performance. The scenario creates a pool for every subnet
and then update those pools.

PARAMETERS:
* pool_create_args: dict, POST /lb/pools request options
* vip_create_args: dict, POST /lb/vips request options
* vip_update_args: dict, POST /lb/vips update options


MODULE:
rally.plugins.openstack.scenarios.neutron.loadbalancer_v1
```

NeutronNetworks.create_and_list_networks [scenario]

```
Create a network and then list all networks.

Measure the ``neutron net-list'' command performance.

If you have only 1 user in your context, you will
add 1 network on every iteration. So you will have more
and more networks and will be able to measure the
performance of the ``neutron net-list'' command depending on
the number of networks owned by users.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_update_networks [scenario]

```
Create and update a network.

Measure the ``neutron net-create and net-update'' command performance.

PARAMETERS:
* network_update_args: dict, PUT /v2.0/networks update request
* network_create_args: dict, POST /v2.0/networks request options
```

```
MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_delete_networks [scenario]

```
Create and delete a network.

Measure the ``neutron net-create'' and ``net-delete'' command performance.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_list_subnets [scenario]

```
Create and a given number of subnets and list all subnets.

The scenario creates a network, a given number of subnets and then
lists subnets.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_update_subnets [scenario]

```
Create and update a subnet.

The scenario creates a network, a given number of subnets
and then updates the subnet. This scenario measures the
``neutron subnet-update'' command performance.

PARAMETERS:
* subnet_update_args: dict, PUT /v2.0/subnets update options
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network
```

```
MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_delete_subnets [scenario]

```
Create and delete a given number of subnets.

The scenario creates a network, a given number of subnets and then
deletes subnets.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_list_routers [scenario]

```
Create and a given number of routers and list all routers.

Create a network, a given number of subnets and routers
and then list all routers.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network
* router_create_args: dict, POST /v2.0/routers request options


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_update_routers [scenario]

```
Create and update a given number of routers.

Create a network, a given number of subnets and routers
and then updating all routers.

PARAMETERS:
* router_update_args: dict, PUT /v2.0/routers update options
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network
```

```
* router_create_args: dict, POST /v2.0/routers request options


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_delete_routers [scenario]

```
Create and delete a given number of routers.

Create a network, a given number of subnets and routers
and then delete all routers.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* subnet_create_args: dict, POST /v2.0/subnets request options
* subnet_cidr_start: str, start value for subnets CIDR
* subnets_per_network: int, number of subnets for one network
* router_create_args: dict, POST /v2.0/routers request options


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_list_ports [scenario]

```
Create and a given number of ports and list all ports.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* port_create_args: dict, POST /v2.0/ports request options
* ports_per_network: int, number of ports for one network


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_update_ports [scenario]

```
Create and update a given number of ports.

Measure the ``neutron port-create'' and ``neutron port-update'' commands
performance.

PARAMETERS:
* port_update_args: dict, PUT /v2.0/ports update request options
* network_create_args: dict, POST /v2.0/networks request options
* port_create_args: dict, POST /v2.0/ports request options
* ports_per_network: int, number of ports for one network
```

```
MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

NeutronNetworks.create_and_delete_ports [scenario]

```
Create and delete a port.

Measure the ``neutron port-create'' and ``neutron port-delete'' commands
performance.

PARAMETERS:
* network_create_args: dict, POST /v2.0/networks request options
* port_create_args: dict, POST /v2.0/ports request options
* ports_per_network: int, number of ports for one network


MODULE:
rally.plugins.openstack.scenarios.neutron.network
```

FuelEnvironments.create_and_delete_environment [scenario]

```
Create and delete Fuel environments.

PARAMETERS:
* release_id: release id (default 1)
* network_provider: network provider (default `neutron')
* deployment_mode: deployment mode (default `ha_compact')
* net_segment_type: net segment type (default `vlan')
* delete_retries: retries count on delete oprations (default 5)


MODULE:
rally.plugins.openstack.scenarios.fuel.environments
```

FuelEnvironments.create_and_list_environments [scenario]

```
Create and list Fuel environments

PARAMETERS:
* release_id: release id (default 1)
* network_provider: network provider (default `neutron')
* deployment_mode: deployment mode (default `ha_compact')
* net_segment_type: net segment type (default `vlan')


MODULE:
rally.plugins.openstack.scenarios.fuel.environments
```

ManilaShares.create_and_delete_share [scenario]

```
Create and delete a share.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between share creation and deletion
(of random duration from [min_sleep, max_sleep]).

PARAMETERS:
* share_proto: share protocol, valid values are NFS, CIFS,
GlusterFS and HDFS
* size: share size in GB, should be greater than 0
* min_sleep: minimum sleep time in seconds (non-negative)
* max_sleep: maximum sleep time in seconds (non-negative)
* kwargs: optional args to create a share


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.list_shares [scenario]

```
Basic scenario for `share list' operation.

PARAMETERS:
* detailed: defines either to return detailed list of
objects or not.
* search_opts: container of search opts such as
``name'', ``host'', ``share_type'', etc.


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.create_share_network_and_delete [scenario]

```
Creates share network and then deletes.

PARAMETERS:
* neutron_net_id: ID of Neutron network
* neutron_subnet_id: ID of Neutron subnet
* nova_net_id: ID of Nova network
* name: share network name
* description: share network description


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.create_share_network_and_list [scenario]

```
Creates share network and then lists it.

PARAMETERS:
* neutron_net_id: ID of Neutron network
* neutron_subnet_id: ID of Neutron subnet
* nova_net_id: ID of Nova network
* name: share network name
* description: share network description
* detailed: defines either to return detailed list of
objects or not.
* search_opts: container of search opts such as
``name'', ``nova_net_id'', ``neutron_net_id'', etc.


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.list_share_servers [scenario]

```
Lists share servers.

Requires admin creds.

PARAMETERS:
* search_opts: container of following search opts:
``host'', ``status'', ``share_network'' and ``project_id''.


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.create_security_service_and_delete [scenario]

```
Creates security service and then deletes.

PARAMETERS:
* security_service_type: security service type, permitted values
are `ldap', `kerberos' or `active_directory'.
* dns_ip: dns ip address used inside tenant's network
* server: security service server ip address or hostname
* domain: security service domain
* user: security identifier used by tenant
* password: password used by user
* name: security service name
* description: security service description


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

ManilaShares.attach_security_service_to_share_network [scenario]

```
Attaches security service to share network.

PARAMETERS:
* security_service_type: type of security service to use.
Should be one of following: `ldap', `kerberos' or
`active_directory'.


MODULE:
rally.plugins.openstack.scenarios.manila.shares
```

IronicNodes.create_and_list_node [scenario]

```
Create and list nodes.

PARAMETERS:
* associated: Optional. Either a Boolean or a string
representation of a Boolean that indicates whether
to return a list of associated (True or ``True'') or
unassociated (False or ``False'') nodes.
* maintenance: Optional. Either a Boolean or a string
representation of a Boolean that indicates whether
to return nodes in maintenance mode (True or
``True''), or not in maintenance mode (False or
``False'').
* marker: Optional, the UUID of a node, eg the last
node from a previous result set. Return
the next result set.
* limit: The maximum number of results to return per
request, if:
1) limit > 0, the maximum number of nodes to return.
2) limit == 0, return the entire list of nodes.
3) limit param is NOT specified (None), the number of items
returned respect the maximum imposed by the Ironic API
(see Ironic's api.max_limit option).
* detail: Optional, boolean whether to return detailed
information about nodes.
* sort_key: Optional, field used for sorting.
* sort_dir: Optional, direction of sorting, either `asc' (the
default) or `desc'.
* kwargs: Optional additional arguments for node creation


MODULE:
rally.plugins.openstack.scenarios.ironic.nodes
```

IronicNodes.create_and_delete_node [scenario]

```
Create and delete node.
```

```
PARAMETERS:
* kwargs: Optional additional arguments for node creation


MODULE:
rally.plugins.openstack.scenarios.ironic.nodes
```

EC2Servers.list_servers [scenario]

```
List all servers.

This simple scenario tests the EC2 API list function by listing
all the servers.

MODULE:
rally.plugins.openstack.scenarios.ec2.servers
```

EC2Servers.boot_server [scenario]

```
Boot a server.

Assumes that cleanup is done elsewhere.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* kwargs: optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.ec2.servers
```

NovaHypervisors.list_hypervisors [scenario]

```
List hypervisors.

Measure the ``nova hypervisor-list'' command performance.

PARAMETERS:
* detailed: True if the hypervisor listing should contain
detailed information about all of them


MODULE:
rally.plugins.openstack.scenarios.nova.hypervisors
```

NovaKeypair.create_and_list_keypairs [scenario]

```
Create a keypair with random name and list keypairs.
```

```
This scenario creates a keypair and then lists all keypairs.

PARAMETERS:
* kwargs: Optional additional arguments for keypair creation


MODULE:
rally.plugins.openstack.scenarios.nova.keypairs
```

NovaKeypair.create_and_delete_keypair [scenario]

```
Create a keypair with random name and delete keypair.

This scenario creates a keypair and then delete that keypair.

PARAMETERS:
* kwargs: Optional additional arguments for keypair creation


MODULE:
rally.plugins.openstack.scenarios.nova.keypairs
```

NovaKeypair.boot_and_delete_server_with_keypair [scenario]

```
Boot and delete server with keypair.

Plan of this scenario:
- create a keypair
- boot a VM with created keypair
- delete server
- delete keypair

PARAMETERS:
* image: ID of the image to be used for server creation
* flavor: ID of the flavor to be used for server creation
* kwargs: Optional additional arguments for keypair creation


MODULE:
rally.plugins.openstack.scenarios.nova.keypairs
```

NovaFloatingIpsBulk.create_and_list_floating_ips_bulk [scenario]

```
Create nova floating IP by range and list it.

This scenario creates a floating IP by range and then lists all.

PARAMETERS:
* start_cidr: Floating IP range
* kwargs: Optional additional arguments for range IP creation
```

```
MODULE:
rally.plugins.openstack.scenarios.nova.floating_ips_bulk
```

NovaFloatingIpsBulk.create_and_delete_floating_ips_bulk [scenario]

```
Create nova floating IP by range and delete it.

This scenario creates a floating IP by range and then delete it.

PARAMETERS:
* start_cidr: Floating IP range
* kwargs: Optional additional arguments for range IP creation


MODULE:
rally.plugins.openstack.scenarios.nova.floating_ips_bulk
```

NovaServers.boot_and_list_server [scenario]

```
Boot a server from an image and then list all servers.

Measure the ``nova list'' command performance.

If you have only 1 user in your context, you will
add 1 server on every iteration. So you will have more
and more servers and will be able to measure the
performance of the ``nova list'' command depending on
the number of servers owned by users.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* detailed: True if the server listing should contain
detailed information about all of them
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.list_servers [scenario]

```
List all servers.

This simple scenario test the nova list command by listing
all the servers.

PARAMETERS:
* detailed: True if detailed information about servers
should be listed
```

```
MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_delete_server [scenario]

```
Boot and delete a server.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between volume creation and deletion
(of random duration from [min_sleep, max_sleep]).

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_delete_multiple_servers [scenario]

```
Boot multiple servers in a single request and delete them.

Deletion is done in parallel with one request per server, not
with a single request for all servers.

PARAMETERS:
* image: The image to boot from
* flavor: Flavor used to boot instance
* count: Number of instances to boot
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for instance creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_server_from_volume_and_delete [scenario]

```
Boot a server from volume and then delete it.

The scenario first creates a volume and then a server.
Optional `min_sleep' and `max_sleep' parameters allow the scenario
```

```
to simulate a pause between volume creation and deletion
(of random duration from [min_sleep, max_sleep]).

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* volume_size: volume size (in GB)
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_bounce_server [scenario]

```
Boot a server and run specified actions against it.

Actions should be passed into the actions parameter. Available actions
are `hard_reboot', `soft_reboot', `stop_start' and `rescue_unrescue'.
Delete server after all actions were completed.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* force_delete: True if force_delete should be used
* actions: list of action dictionaries, where each action
dictionary speicifes an action to be performed
in the following format:
{``action_name'': <no_of_iterations>}
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_lock_unlock_and_delete [scenario]

```
Boot a server, lock it, then unlock and delete it.

Optional `min_sleep' and `max_sleep' parameters allow the
scenario to simulate a pause between locking and unlocking the
server (of random duration from min_sleep to max_sleep).

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* min_sleep: Minimum sleep time between locking and unlocking
in seconds
* max_sleep: Maximum sleep time between locking and unlocking
```

```
in seconds
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.snapshot_server [scenario]

```
Boot a server, make its snapshot and delete both.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_server [scenario]

```
Boot a server.

Assumes that cleanup is done elsewhere.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* auto_assign_nic: True if NICs should be assigned
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_server_from_volume [scenario]

```
Boot a server from volume.

The scenario first creates a volume and then a server.
Assumes that cleanup is done elsewhere.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* volume_size: volume size (in GB)
* auto_assign_nic: True if NICs should be assigned
* kwargs: Optional additional arguments for server creation
```

```
MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.resize_server [scenario]

```
Boot a server, then resize and delete it.

This test will confirm the resize by default,
or revert the resize if confirm is set to false.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* to_flavor: flavor to be used to resize the booted instance
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.suspend_and_resume_server [scenario]

```
Create a server, suspend, resume and then delete it

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.pause_and_unpause_server [scenario]

```
Create a server, pause, unpause and then delete it

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.shelve_and_unshelve_server [scenario]

```
Create a server, shelve, unshelve and then delete it

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* force_delete: True if force_delete should be used
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_live_migrate_server [scenario]

```
Live Migrate a server.

This scenario launches a VM on a compute node available in
the availability zone and then migrates the VM to another
compute node on the same availability zone.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between VM booting and running live migration
(of random duration from range [min_sleep, max_sleep]).

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* block_migration: Specifies the migration type
* disk_over_commit: Specifies whether to allow overcommit
on migrated instance or not
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_server_from_volume_and_live_migrate [scenario]

```
Boot a server from volume and then migrate it.

The scenario first creates a volume and a server booted from
the volume on a compute node available in the availability zone and
then migrates the VM to another compute node on the same availability
zone.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between VM booting and running live migration
(of random duration from range [min_sleep, max_sleep]).
```

```
PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* volume_size: volume size (in GB)
* block_migration: Specifies the migration type
* disk_over_commit: Specifies whether to allow overcommit
on migrated instance or not
* force_delete: True if force_delete should be used
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_server_attach_created_volume_and_live_migrate [scenario]

```
Create a VM, attach a volume to it and live migrate.

Simple test to create a VM and attach a volume, then migrate the VM,
detach the volume and delete volume/VM.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between attaching a volume and running live
migration (of random duration from range [min_sleep, max_sleep]).

PARAMETERS:
* image: Glance image name to use for the VM
* flavor: VM flavor name
* size: volume size (in GB)
* block_migration: Specifies the migration type
* disk_over_commit: Specifies whether to allow overcommit
on migrated instance or not
* boot_server_kwargs: optional arguments for VM creation
* create_volume_kwargs: optional arguments for volume creation
* min_sleep: Minimum sleep time in seconds (non-negative)
* max_sleep: Maximum sleep time in seconds (non-negative)


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_migrate_server [scenario]

```
Migrate a server.

This scenario launches a VM on a compute node available in
the availability zone and stops the VM, and then migrates the VM
to another compute node on the same availability zone.
```

```
PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_rebuild_server [scenario]

```
Rebuild a server.

This scenario launches a VM, then rebuilds that VM with a
different image.

PARAMETERS:
* from_image: image to be used to boot an instance
* to_image: image to be used to rebuild the instance
* flavor: flavor to be used to boot an instance
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaServers.boot_and_associate_floating_ip [scenario]

```
Boot a server and associate a floating IP to it.

PARAMETERS:
* image: image to be used to boot an instance
* flavor: flavor to be used to boot an instance
* kwargs: Optional additional arguments for server creation


MODULE:
rally.plugins.openstack.scenarios.nova.servers
```

NovaSecGroup.create_and_delete_secgroups [scenario]

```
Create and delete security groups.

This scenario creates N security groups with M rules per group and then
deletes them.

PARAMETERS:
* security_group_count: Number of security groups
* rules_per_security_group: Number of rules per security group
```

```
MODULE:
rally.plugins.openstack.scenarios.nova.security_group
```

NovaSecGroup.create_and_list_secgroups [scenario]

```
Create and list security groups.

This scenario creates N security groups with M rules per group and then
lists them.

PARAMETERS:
* security_group_count: Number of security groups
* rules_per_security_group: Number of rules per security group


MODULE:
rally.plugins.openstack.scenarios.nova.security_group
```

NovaSecGroup.create_and_update_secgroups [scenario]

```
Create and update security groups.

This scenario creates `security_group_count' security groups
then updates their name and description.

PARAMETERS:
* security_group_count: Number of security groups


MODULE:
rally.plugins.openstack.scenarios.nova.security_group
```

NovaSecGroup.boot_and_delete_server_with_secgroups [scenario]

```
Boot and delete server with security groups attached.

Plan of this scenario:
- create N security groups with M rules per group
vm with security groups)
- boot a VM with created security groups
- get list of attached security groups to server
- delete server
- delete all security groups
- check that all groups were attached to server

PARAMETERS:
* image: ID of the image to be used for server creation
* flavor: ID of the flavor to be used for server creation
* security_group_count: Number of security groups
* rules_per_security_group: Number of rules per security group
* **kwargs: Optional arguments for booting the instance
```

```
MODULE:
rally.plugins.openstack.scenarios.nova.security_group
```

NovaNetworks.create_and_list_networks [scenario]

```
Create nova network and list all networks.

PARAMETERS:
* start_cidr: IP range
* kwargs: Optional additional arguments for network creation


MODULE:
rally.plugins.openstack.scenarios.nova.networks
```

NovaNetworks.create_and_delete_network [scenario]

```
Create nova network and delete it.

PARAMETERS:
* start_cidr: IP range
* kwargs: Optional additional arguments for network creation


MODULE:
rally.plugins.openstack.scenarios.nova.networks
```

Quotas.nova_update [scenario]

```
Update quotas for Nova.

PARAMETERS:
* max_quota: Max value to be updated for quota.


MODULE:
rally.plugins.openstack.scenarios.quotas.quotas
```

Quotas.nova_update_and_delete [scenario]

```
Update and delete quotas for Nova.

PARAMETERS:
* max_quota: Max value to be updated for quota.


MODULE:
rally.plugins.openstack.scenarios.quotas.quotas
```

Quotas.cinder_update [scenario]

```
Update quotas for Cinder.

PARAMETERS:
* max_quota: Max value to be updated for quota.


MODULE:
rally.plugins.openstack.scenarios.quotas.quotas
```

Quotas.cinder_update_and_delete [scenario]

```
Update and Delete quotas for Cinder.

PARAMETERS:
* max_quota: Max value to be updated for quota.


MODULE:
rally.plugins.openstack.scenarios.quotas.quotas
```

Quotas.neutron_update [scenario]

```
Update quotas for neutron.

PARAMETERS:
* max_quota: Max value to be updated for quota.


MODULE:
rally.plugins.openstack.scenarios.quotas.quotas
```

GlanceImages.create_and_list_image [scenario]

```
Create an image and then list all images.

Measure the ``glance image-list'' command performance.

If you have only 1 user in your context, you will
add 1 image on every iteration. So you will have more
and more images and will be able to measure the
performance of the ``glance image-list'' command depending on
the number of images owned by users.

PARAMETERS:
* container_format: container format of image. Acceptable
formats: ami, ari, aki, bare, and ovf
* image_location: image file location
* disk_format: disk format of image. Acceptable formats:
ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, and iso
```

```
* kwargs: optional parameters to create image


MODULE:
rally.plugins.openstack.scenarios.glance.images
```

GlanceImages.list_images [scenario]

```
List all images.

This simple scenario tests the glance image-list command by listing
all the images.

Suppose if we have 2 users in context and each has 2 images
uploaded for them we will be able to test the performance of
glance image-list command in this case.

MODULE:
rally.plugins.openstack.scenarios.glance.images
```

GlanceImages.create_and_delete_image [scenario]

```
Create and then delete an image.

PARAMETERS:
* container_format: container format of image. Acceptable
formats: ami, ari, aki, bare, and ovf
* image_location: image file location
* disk_format: disk format of image. Acceptable formats:
ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, and iso
* kwargs: optional parameters to create image


MODULE:
rally.plugins.openstack.scenarios.glance.images
```

GlanceImages.create_image_and_boot_instances [scenario]

```
Create an image and boot several instances from it.

PARAMETERS:
* container_format: container format of image. Acceptable
formats: ami, ari, aki, bare, and ovf
* image_location: image file location
* disk_format: disk format of image. Acceptable formats:
ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, and iso
* flavor: Nova flavor to be used to launch an instance
* number_instances: number of Nova servers to boot
* kwargs: optional parameters to create server
```

```
MODULE:
rally.plugins.openstack.scenarios.glance.images
```

HeatStacks.create_and_list_stack [scenario]

```
Create a stack and then list all stacks.

Measure the ``heat stack-create'' and ``heat stack-list'' commands
performance.

PARAMETERS:
* template_path: path to stack template file
* parameters: parameters to use in heat template
* files: files used in template
* environment: stack environment definition


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.list_stacks_and_resources [scenario]

```
List all resources from tenant stacks.

MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_and_delete_stack [scenario]

```
Create and then delete a stack.

Measure the ``heat stack-create'' and ``heat stack-delete'' commands
performance.

PARAMETERS:
* template_path: path to stack template file
* parameters: parameters to use in heat template
* files: files used in template
* environment: stack environment definition


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_check_delete_stack [scenario]

```
Create, check and delete a stack.

Measure the performance of the following commands:
- heat stack-create
```

```
- heat action-check
- heat stack-delete

PARAMETERS:
* template_path: path to stack template file
* parameters: parameters to use in heat template
* files: files used in template
* environment: stack environment definition


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_update_delete_stack [scenario]

```
Create, update and then delete a stack.

Measure the ``heat stack-create'', ``heat stack-update''
and ``heat stack-delete'' commands performance.

PARAMETERS:
* template_path: path to stack template file
* updated_template_path: path to updated stack template file
* parameters: parameters to use in heat template
* updated_parameters: parameters to use in updated heat template
If not specified then parameters will be
used instead
* files: files used in template
* updated_files: files used in updated template. If not specified
files value will be used instead
* environment: stack environment definition
* updated_environment: environment definition for updated stack


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_stack_and_scale [scenario]

```
Create an autoscaling stack and invoke a scaling policy.

Measure the performance of autoscaling webhooks.

PARAMETERS:
* template_path: path to template file that includes an
OS::Heat::AutoScalingGroup resource
* output_key: the stack output key that corresponds to
the scaling webhook
* delta: the number of instances the stack is expected to
change by.
* parameters: parameters to use in heat template
* files: files used in template (dict of file name to
```

```
file path)
* environment: stack environment definition (dict)


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_suspend_resume_delete_stack [scenario]

```
Create, suspend-resume and then delete a stack.

Measure performance of the following commands:
heat stack-create
heat action-suspend
heat action-resume
heat stack-delete

PARAMETERS:
* template_path: path to stack template file
* parameters: parameters to use in heat template
* files: files used in template
* environment: stack environment definition


MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.list_stacks_and_events [scenario]

```
List events from tenant stacks.

MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

HeatStacks.create_snapshot_restore_delete_stack [scenario]

```
Create, snapshot-restore and then delete a stack.

Measure performance of the following commands:
heat stack-create
heat stack-snapshot
heat stack-restore
heat stack-delete

PARAMETERS:
* template_path: path to stack template file
* parameters: parameters to use in heat template
* files: files used in template
* environment: stack environment definition
```

```
MODULE:
rally.plugins.openstack.scenarios.heat.stacks
```

VMTasks.boot_runcommand_delete [scenario]

```
Boot a server, run a script that outputs JSON, delete the server.

Example Script in samples/tasks/support/instance_dd_test.sh

PARAMETERS:
* image: glance image name to use for the vm
* flavor: VM flavor name
* username: ssh username on server, str
* password: Password on SSH authentication
* script: DEPRECATED. Use `command' instead. Script to run on
server, must output JSON mapping metric names to values (see the
sample script below)
* interpreter: DEPRECATED. Use `command' instead. server's
interpreter to run the script
* command: Command-specifying dictionary that either specifies
remote command path via `remote_path' (can be uploaded from a
local file specified by `local_path`), an inline script via
`script_inline' or a local script file path using `script_file'.
Both `script_file' and `local_path' are checked to be accessible
by the `file_exists' validator code.

The `script_inline' and `script_file' both require an `interpreter'
value to specify the interpreter script should be run with.

Note that any of `interpreter' and `remote_path' can be an array
prefixed with environment variables and suffixed with args for
the `interpreter' command. `remote_path's last component must be
a path to a command to execute (also upload destination if a
`local_path' is given). Uploading an interpreter is possible
but requires that `remote_path' and `interpreter' path do match.


Examples::

# Run a `local_script.pl' file sending it to a remote
# Perl interpreter
command = {
``script_file'': ``local_script.pl'',
``interpreter'': ``/usr/bin/perl''
}

# Run an inline script sending it to a remote interpreter
command = {
``script_inline'': ``echo `Hello, World!'''',
``interpreter'': ``/bin/sh''
}

# Run a remote command
```

```
command = {
``remote_path'': ``/bin/false''
}

# Copy a local command and run it
command = {
``remote_path'': ``/usr/local/bin/fio'',
``local_path'': ``/home/foobar/myfiodir/bin/fio''
}

# Copy a local command and run it with environment variable
command = {
``remote_path'': [''HOME=/root'', ``/usr/local/bin/fio''],
``local_path'': ``/home/foobar/myfiodir/bin/fio''
}

# Run an inline script sending it to a remote interpreter
command = {
``script_inline'': ``echo ``Hello, ${NAME:-World}'''',
``interpreter'': [''NAME=Earth'', ``/bin/sh'']
}

# Run an inline script sending it to a uploaded remote
# interpreter
command = {
``script_inline'': ``echo ``Hello, ${NAME:-World}'''',
``interpreter'': [''NAME=Earth'', ``/tmp/sh''],
``remote_path'': ``/tmp/sh'',
``local_path'': ``/home/user/work/cve/sh-1.0/bin/sh''
}
* volume_args: volume args for booting server from volume
* floating_network: external network name, for floating ip
* port: ssh port for SSH connection
* use_floating_ip: bool, floating or fixed IP for SSH connection
* force_delete: whether to use force_delete for servers
* wait_for_ping: whether to check connectivity on server creation
* **kwargs: extra arguments for booting the server


RETURNS:
dictionary with keys `data' and `errors':
data: dict, JSON output from the script
errors: str, raw data from the script's stderr stream

MODULE:
rally.plugins.openstack.scenarios.vm.vmtasks
```

VMTasks.boot_runcommand_delete_custom_image [scenario]

```
Boot a server from a custom image, run a command that outputs JSON.

Example Script in rally-jobs/extra/script_benchmark.sh
```

```
MODULE:
rally.plugins.openstack.scenarios.vm.vmtasks
```

CinderVolumes.create_and_list_volume [scenario]

```
Create a volume and list all volumes.

Measure the ``cinder volume-list'' command performance.

If you have only 1 user in your context, you will
add 1 volume on every iteration. So you will have more
and more volumes and will be able to measure the
performance of the ``cinder volume-list'' command depending on
the number of images owned by users.

PARAMETERS:
* size: volume size (integer, in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* detailed: determines whether the volume listing should contain
detailed information about all of them
* image: image to be used to create volume
* kwargs: optional args to create a volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.list_volumes [scenario]

```
List all volumes.

This simple scenario tests the cinder list command by listing
all the volumes.

PARAMETERS:
* detailed: True if detailed information about volumes
should be listed


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_update_volume [scenario]

```
Create a volume and update its name and description.

PARAMETERS:
* size: volume size (integer, in GB)
* image: image to be used to create volume
```

```
* create_volume_kwargs: dict, to be used to create volume
* update_volume_kwargs: dict, to be used to update volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_delete_volume [scenario]

```
Create and then delete a volume.

Good for testing a maximal bandwidth of cloud. Optional `min_sleep'
and `max_sleep' parameters allow the scenario to simulate a pause
between volume creation and deletion (of random duration from
[min_sleep, max_sleep]).

PARAMETERS:
* size: volume size (integer, in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* image: image to be used to create volume
* min_sleep: minimum sleep time between volume creation and
deletion (in seconds)
* max_sleep: maximum sleep time between volume creation and
deletion (in seconds)
* kwargs: optional args to create a volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_volume [scenario]

```
Create a volume.

Good test to check how influence amount of active volumes on
performance of creating new.

PARAMETERS:
* size: volume size (integer, in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* image: image to be used to create volume
* kwargs: optional args to create a volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.modify_volume_metadata [scenario]

```
Modify a volume's metadata.

This requires a volume to be created with the volumes
context. Additionally, ``sets * set_size`` must be greater
than or equal to ``deletes * delete_size``.

PARAMETERS:
* sets: how many set_metadata operations to perform
* set_size: number of metadata keys to set in each
set_metadata operation
* deletes: how many delete_metadata operations to perform
* delete_size: number of metadata keys to delete in each
delete_metadata operation


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_extend_volume [scenario]

```
Create and extend a volume and then delete it.

PARAMETERS:
* size: volume size (in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* new_size: volume new size (in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
to extend.
Notice: should be bigger volume size
* min_sleep: minimum sleep time between volume extension and
deletion (in seconds)
* max_sleep: maximum sleep time between volume extension and
deletion (in seconds)
* kwargs: optional args to extend the volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_from_volume_and_delete_volume [scenario]

```
Create volume from volume and then delete it.

Scenario for testing volume clone.Optional `min_sleep' and `max_sleep'
parameters allow the scenario to simulate a pause between volume
creation and deletion (of random duration from [min_sleep, max_sleep]).
```

```
PARAMETERS:
* size: volume size (in GB), or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
Should be equal or bigger source volume size
* min_sleep: minimum sleep time between volume creation and
deletion (in seconds)
* max_sleep: maximum sleep time between volume creation and
deletion (in seconds)
* kwargs: optional args to create a volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_delete_snapshot [scenario]

```
Create and then delete a volume-snapshot.

Optional `min_sleep' and `max_sleep' parameters allow the scenario
to simulate a pause between snapshot creation and deletion
(of random duration from [min_sleep, max_sleep]).

PARAMETERS:
* force: when set to True, allows snapshot of a volume when
the volume is attached to an instance
* min_sleep: minimum sleep time between snapshot creation and
deletion (in seconds)
* max_sleep: maximum sleep time between snapshot creation and
deletion (in seconds)
* kwargs: optional args to create a snapshot


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_attach_volume [scenario]

```
Create a VM and attach a volume to it.

Simple test to create a VM and attach a volume, then
detach the volume and delete volume/VM.

PARAMETERS:
* size: volume size (integer, in GB) or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* image: Glance image name to use for the VM
* flavor: VM flavor name
```

```
* kwargs: optional arguments for VM creation


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_snapshot_and_attach_volume [scenario]

```
Create volume, snapshot and attach/detach volume.

This scenario is based off of the standalone qaStressTest.py
(https://github.com/WaltHP/cinder-stress).

PARAMETERS:
* volume_type: Whether or not to specify volume type when creating
volumes.
* size: Volume size - dictionary, contains two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
default values: {``min'': 1, ``max'': 5}
* kwargs: Optional parameters used during volume
snapshot creation.


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_nested_snapshots_and_attach_volume [scenario]

```
Create a volume from snapshot and attach/detach the volume

This scenario create volume, create it's snapshot, attach volume,
then create new volume from existing snapshot and so on,
with defined nested level, after all detach and delete them.
volume->snapshot->volume->snapshot->volume ...

PARAMETERS:
* size: Volume size - dictionary, contains two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
default values: {``min'': 1, ``max'': 5}
* nested_level: Nested level - dictionary, contains two values:
min - minimum number of volumes will be created
from snapshot;
max - maximum number of volumes will be created
from snapshot.
default values: {``min'': 5, ``max'': 10}
* kwargs: Optional parameters used during volume
snapshot creation.


MODULE:
```

```
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_list_snapshots [scenario]

```
Create and then list a volume-snapshot.

PARAMETERS:
* force: when set to True, allows snapshot of a volume when
the volume is attached to an instance
* detailed: True if detailed information about snapshots
should be listed
* kwargs: optional args to create a snapshot


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_upload_volume_to_image [scenario]

```
Create and upload a volume to image.

PARAMETERS:
* size: volume size (integers, in GB), or
dictionary, must contain two values:
min - minimum size volumes will be created as;
max - maximum size volumes will be created as.
* force: when set to True volume that is attached to an instance
could be uploaded to image
* container_format: image container format
* disk_format: disk format for image
* do_delete: deletes image and volume after uploading if True
* kwargs: optional args to create a volume


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_volume_backup [scenario]

```
Create a volume backup.

PARAMETERS:
* size: volume size in GB
* do_delete: if True, a volume and a volume backup will
be deleted after creation.
* create_volume_kwargs: optional args to create a volume
* create_backup_kwargs: optional args to create a volume backup


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_restore_volume_backup [scenario]

```
Restore volume backup.

PARAMETERS:
* size: volume size in GB
* do_delete: if True, the volume and the volume backup will
be deleted after creation.
* create_volume_kwargs: optional args to create a volume
* create_backup_kwargs: optional args to create a volume backup


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

CinderVolumes.create_and_list_volume_backups [scenario]

```
Create and then list a volume backup.

PARAMETERS:
* size: volume size in GB
* detailed: True if detailed information about backup
should be listed
* do_delete: if True, a volume backup will be deleted
* create_volume_kwargs: optional args to create a volume
* create_backup_kwargs: optional args to create a volume backup


MODULE:
rally.plugins.openstack.scenarios.cinder.volumes
```

Dummy.dummy [scenario]

```
Do nothing and sleep for the given number of seconds (0 by default).

Dummy.dummy can be used for testing performance of different
ScenarioRunners and of the ability of rally to store a large
amount of results.

PARAMETERS:
* sleep: idle time of method (in seconds).


MODULE:
rally.plugins.common.scenarios.dummy.dummy
```

Dummy.dummy_exception [scenario]

```
Throw an exception.

Dummy.dummy_exception can be used for test if exceptions are processed
```

```
properly by ScenarioRunners and benchmark and analyze rally
results storing process.

PARAMETERS:
* size_of_message: int size of the exception message
* sleep: idle time of method (in seconds).


MODULE:
rally.plugins.common.scenarios.dummy.dummy
```

Dummy.dummy_exception_probability [scenario]

```
Throw an exception with given probability.

Dummy.dummy_exception_probability can be used to test if exceptions
are processed properly by ScenarioRunners. This scenario will throw
an exception sometimes, depending on the given exception probability.

PARAMETERS:
* exception_probability: Sets how likely it is that an exception
will be thrown. Float between 0 and 1
0=never 1=always.


MODULE:
rally.plugins.common.scenarios.dummy.dummy
```

Dummy.dummy_with_scenario_output [scenario]

```
Return a dummy scenario output.

Dummy.dummy_with_scenario_output can be used to test the scenario
output processing.

MODULE:
rally.plugins.common.scenarios.dummy.dummy
```

Dummy.dummy_random_fail_in_atomic [scenario]

```
Randomly throw exceptions in atomic actions.

Dummy.dummy_random_fail_in_atomic can be used to test atomic actions
failures processing.

PARAMETERS:
* exception_probability: Probability with which atomic actions
fail in this dummy scenario (0 <= p <= 1)


MODULE:
```

```
rally.plugins.common.scenarios.dummy.dummy
```

HttpRequests.check_request [scenario]

```
Standard way to benchmark web services.

This benchmark is used to make request and check it with expected
Response.

PARAMETERS:
* url: url for the Request object
* method: method for the Request object
* status_code: expected response code
* kwargs: optional additional request parameters


MODULE:
rally.plugins.common.scenarios.requests.http_requests
```

HttpRequests.check_random_request [scenario]

```
Benchmark the list of requests

This scenario takes random url from list of requests, and raises
exception if the response is not the expected response.

PARAMETERS:
* requests: List of request dicts
* status_code: Expected Response Code it will
be used only if we doesn't specified it in request proper


MODULE:
rally.plugins.common.scenarios.requests.http_requests
```

Engines [deployment]

ExistingCloud [engine]

```
Just use an existing OpenStack deployment without deploying anything.

To use ExistingCloud, you should put endpoint information to the config:

{
``type'': ``ExistingCloud'',
``auth_url'': ``http://localhost:5000/v2.0/'',
``region_name'': ``RegionOne'',
``endpoint_type'': ``public'',
``admin'': {
``username'': ``admin'',
``password'': ``password'',
``tenant_name'': ``demo''
```

```
},
``https_insecure'': False,
``https_cacert'': ``'',
}

Or, using keystone v3 API endpoint:

{
``type'': ``ExistingCloud'',
``auth_url'': ``http://localhost:5000/v3/'',
``region_name'': ``RegionOne'',
``endpoint_type'': ``public'',
``admin'': {
``username'': ``admin'',
``password'': ``admin'',
``user_domain_name'': ``admin'',
``project_name'': ``admin'',
``project_domain_name'': ``admin'',
},
``https_insecure'': False,
``https_cacert'': ``'',
}

MODULE:
rally.deployment.engines.existing
```

FuelEngine [engine]

```
Deploy with FuelWeb.

Sample configuration:

{
``type'': ``FuelEngine'',
``deploy_name'': ``Rally multinode 01'',
``release'': ``Havana on CentOS 6.4'',
``api_url'': ``http://10.20.0.2:8000/api/v1/'',
``mode'': ``multinode'',
``nodes'': {
``controller'': {``amount'': 1, ``filters'': [''storage>80G'']},
``compute'': {``amount'': 1, ``filters'': [''storage>80G'']}
},
``net_provider'': ``nova_network'',
``dns_nameservers'': [''172.18.208.44'', ``8.8.8.8''],
``networks'': {

``public'': {
``cidr'': ``10.3.3.0/24'',
``gateway'': ``10.3.3.1'',
``ip_ranges'': [[''10.3.3.5'', ``10.3.3.254'']],
``vlan_start'': 14
},
```

```
``floating'': {
``cidr'': ``10.3.4.0/24'',
``ip_ranges'': [[''10.3.4.5'', ``10.3.4.254'']],
``vlan_start'': 14
}
}
}

MODULE:
rally.deployment.engines.fuel
```

LxcEngine [engine]

```
Deploy with other engines in lxc containers.

Sample configuration:

{
``type'': ``LxcEngine'',
``provider'': {
``type'': ``DummyProvider'',
``credentials'': [{``user'': ``root'', ``host'': ``example.net''}]
},
``distribution'': ``ubuntu'',
``release'': ``raring'',
``tunnel_to'': [''10.10.10.10'', ``10.10.10.11''],
``start_lxc_network'': ``10.1.1.0/24'',
``container_name_prefix'': ``devstack-node'',
``containers_per_host'': 16,
``start_script'': ``~/start.sh'',
``engine'': { ... }
}

MODULE:
rally.deployment.engines.lxc
```

DevstackEngine [engine]

```
Deploy Devstack cloud.

Sample configuration:

{
``type'': ``DevstackEngine'',
``devstack_repo'': ``https://example.com/devstack/'',
``localrc'': {
``ADMIN_PASSWORD'': ``secret''
},
``provider'': {
``type'': ``ExistingServers'',
``credentials'': [{``user'': ``root'', ``host'': ``10.2.0.8''}]
}
```

```
}

MODULE:
rally.deployment.engines.devstack
```

MultihostEngine [engine]

```
Deploy multihost cloud with existing engines.

Sample configuration:

{
``type'': ``MultihostEngine'',
``controller'': {
``type'': ``DevstackEngine'',
``provider'': {
``type'': ``DummyProvider''
}
},
``nodes'': [
{``type'': ``Engine1'', ``config'': ``Config1''},
{``type'': ``Engine2'', ``config'': ``Config2''},
{``type'': ``Engine3'', ``config'': ``Config3''},
]
}

If {controller_ip} is specified in configuration values, it will be
replaced with controller address taken from endpoint returned by
controller engine:

...
``nodes'': [
{
``type'': ``DevstackEngine'',
``localrc'': {
``GLANCE_HOSTPORT'': ``{controller_ip}:9292'',
...

MODULE:
rally.deployment.engines.multihost
```

Server Providers [deployment]

LxcProvider [server provider]

```
Provide lxc container(s) on given host.

Sample configuration:
{
``type'': ``LxcProvider'',
``distribution'': ``ubuntu'',
``start_lxc_network'': ``10.1.1.0/24'',
``containers_per_host'': 32,
```

```
``tunnel_to'': [''10.10.10.10''],
``forward_ssh'': false,
``container_name_prefix'': ``rally-multinode-02'',
``host_provider'': {
``type'': ``ExistingServers'',
``credentials'': [{``user'': ``root'', ``host'': ``host.net''}]
}
}

MODULE:
rally.deployment.serverprovider.providers.lxc
```

ExistingServers [server provider]

```
Just return endpoints from its own configuration.

Sample configuration:

{
``type'': ``ExistingServers'',
``credentials'': [{``user'': ``root'', ``host'': ``localhost''}]
}

MODULE:
rally.deployment.serverprovider.providers.existing
```

VirshProvider [server provider]

```
Create VMs from prebuilt templates.

Sample configuration:

{
``type'': ``VirshProvider'',
``connection'': ``alex@performance-01'',  # ssh connection to vms host
``template_name'': ``stack-01-devstack-template'',  # vm image template
``template_user'': ``ubuntu'',  # vm user to launch devstack
``template_password'': ``password'' # vm password to launch devstack
}

MODULE:
rally.deployment.serverprovider.providers.virsh
```

OpenStackProvider [server provider]

```
Provide VMs using an existing OpenStack cloud.

Sample configuration:

{
``type'': ``OpenStackProvider'',
```

```
``amount'': 42
``user'': ``admin'',
``tenant'': ``admin'',
``password'': ``secret'',
``auth_url'': ``http://example.com/'',
``flavor_id'': 2,
``image'': {
``checksum'': ``75846dd06e9fcfd2b184aba7fa2b2a8d'',
``url'': ``http://example.com/disk1.img'',
``name'': ``Ubuntu Precise(added by rally)'',
``format'': ``qcow2'',
``userdata'': ``#cloud-config
disable_root: false''
}
}

MODULE:
rally.deployment.serverprovider.providers.openstack
```

1.7 Contribute to Rally

1.7.1 Where to begin

Please take a look our Roadmap to get information about our current work directions.

In case you have questions or want to share your ideas, be sure to contact us at the `#openstack-rally` IRC channel on **irc.freenode.net**.

If you are going to contribute to Rally, you will probably need to grasp a better understanding of several main design concepts used throughout our project (such as **benchmark scenarios**, **contexts** etc.). To do so, please read *this article*.

1.7.2 How to contribute

1. You need a Launchpad account and need to be joined to the Openstack team. You can also join the Rally team if you want to. Make sure Launchpad has your SSH key, Gerrit (the code review system) uses this.
2. Sign the CLA as outlined in the account setup section of the developer guide.
3. Tell git your details:

```
git config --global user.name "Firstname Lastname"
git config --global user.email "your_email@youremail.com"
```

4. Install git-review. This tool takes a lot of the pain out of remembering commands to push code up to Gerrit for review and to pull it back down to edit it. It is installed using:

```
pip install git-review
```

Several Linux distributions (notably Fedora 16 and Ubuntu 12.04) are also starting to include git-review in their repositories so it can also be installed using the standard package manager.

5. Grab the Rally repository:

```
git clone git@github.com:openstack/rally.git
```

6. Checkout a new branch to hack on:

```
git checkout -b TOPIC-BRANCH
```

7. Start coding
8. Run the test suite locally to make sure nothing broke, e.g. (this will run py26/py27/pep8 tests):

```
tox
```

(NOTE: you should have installed tox<=1.6.1)

If you extend Rally with new functionality, make sure you have also provided unit and/or functional tests for it.

9. Commit your work using:

```
git commit -a
```

Make sure you have supplied your commit with a neat commit message, containing a link to the corresponding blueprint / bug, if appropriate.

10. Push the commit up for code review using:

```
git review -R
```

That is the awesome tool we installed earlier that does a lot of hard work for you.

11. Watch your email or review site, it will automatically send your code for a battery of tests on our Jenkins setup and the core team for the project will review your code. If there are any changes that should be made they will let you know.
12. When all is good the review site will automatically merge your code.

(This tutorial is based on: http://www.linuxjedi.co.uk/2012/03/real-way-to-start-hacking-on-openstack.html)

1.7.3 Testing

Please, don't hesitate to write tests ;)

Unit tests

*Files: /tests/unit/**

The goal of unit tests is to ensure that internal parts of the code work properly. All internal methods should be fully covered by unit tests with a reasonable mocks usage.

About Rally unit tests:

- All unit tests are located inside /tests/unit/*
- Tests are written on top of: *testtools*, *fixtures* and *mock* libs
- Tox is used to run unit tests

To run unit tests locally:

```
$ pip install tox
$ tox
```

To run py26, py27 or pep8 only:

```
$ tox -e <name>

#NOTE: <name> is one of py26, py27 or pep8
```

To get test coverage:

```
$ tox -e cover

#NOTE: Results will be in /cover/index.html
```

To generate docs:

```
$ tox -e docs

#NOTE: Documentation will be in doc/source/_build/html/index.html
```

Functional tests

*Files: /tests/functional/**

The goal of functional tests is to check that everything works well together. Functional tests use Rally API only and check responses without touching internal parts.

To run functional tests locally:

```
$ source openrc
$ rally deployment create --fromenv --name testing
$ tox -e cli

#NOTE: openrc file with OpenStack admin credentials
```

Output of every Rally execution will be collected under some reports root in directory structure like: reports_root/ClassName/MethodName_suffix.extension This functionality implemented in tests.functional.utils.Rally.__call__ method. Use 'gen_report_path' method of 'Rally' class to get automatically generated file path and name if you need. You can use it to publish html reports, generated during tests. Reports root can be passed throw environment variable 'REPORTS_ROOT'. Default is 'rally-cli-output-files'.

Rally CI scripts

*Files: /tests/ci/**

This directory contains scripts and files related to the Rally CI system.

Rally Style Commandments

Files: /tests/hacking/

This module contains Rally specific hacking rules for checking commandments.

For more information about Style Commandments, read the OpenStack Style Commandments manual.

1.8 Rally OS Gates

1.8.1 Gate jobs

The **Openstack CI system** uses the so-called **"Gate jobs"** to control merges of patched submitted for review on Gerrit. These **Gate jobs** usually just launch a set of tests – unit, functional, integration, style – that check that the proposed patch does not break the software and can be merged into the target branch, thus providing additional guarantees for the stability of the software.

1.8.2 Create a custom Rally Gate job

You can create a **Rally Gate job** for your project to run Rally benchmarks against the patchsets proposed to be merged into your project.

To create a rally-gate job, you should create a **rally-jobs/** directory at the root of your project.

As a rule, this directory contains only **{projectname}.yaml**, but more scenarios and jobs can be added as well. This yaml file is in fact an input Rally task file specifying benchmark scenarios that should be run in your gate job.

To make *{projectname}.yaml* run in gates, you need to add *"rally-jobs"* to the "jobs" section of *projects.yaml* in *openstack-infra/project-config*.

1.8.3 Example: Rally Gate job for Glance

Let's take a look at an example for the Glance project:

Edit *jenkins/jobs/projects.yaml:*

```
- project:
    name: glance
    node: `bare-precise || bare-trusty'
    tarball-site: tarballs.openstack.org
    doc-publisher-site: docs.openstack.org

    jobs:
      - python-jobs
      - python-icehouse-bitrot-jobs
      - python-juno-bitrot-jobs
      - openstack-publish-jobs
      - translation-jobs
      - rally-jobs
```

Also add *gate-rally-dsvm-{projectname}* to *zuul/layout.yaml*:

```
- name: openstack/glance
  template:
    - name: merge-check
    - name: python26-jobs
    - name: python-jobs
    - name: openstack-server-publish-jobs
    - name: openstack-server-release-jobs
    - name: periodic-icehouse
    - name: periodic-juno
    - name: check-requirements
    - name: integrated-gate
    - name: translation-jobs
    - name: large-ops
    - name: experimental-tripleo-jobs
  check:
    - check-devstack-dsvm-cells
    - gate-rally-dsvm-glance
  gate:
    - gate-devstack-dsvm-cells
  experimental:
    - gate-grenade-dsvm-forward
```

To add one more scenario and job, you need to add *{scenarioname}.yaml* file here, and *gate-rally-dsvm-{scenarioname}* to *projects.yaml*.

For example, you can add *myscenario.yaml* to *rally-jobs* directory in your project and then edit *jenkins/jobs/projects.yaml* in this way:

```
- project:
    name: glance
    github-org: openstack
    node: bare-precise
    tarball-site: tarballs.openstack.org
    doc-publisher-site: docs.openstack.org

    jobs:
      - python-jobs
      - python-havana-bitrot-jobs
      - openstack-publish-jobs
      - translation-jobs
      - rally-jobs
      - `gate-rally-dsvm-{name}':
        name: myscenario
```

Finally, add *gate-rally-dsvm-myscenario* to *zuul/layout.yaml*:

```
- name: openstack/glance
  template:
    - name: python-jobs
    - name: openstack-server-publish-jobs
    - name: periodic-havana
    - name: check-requirements
    - name: integrated-gate
  check:
    - check-devstack-dsvm-cells
    - check-tempest-dsvm-postgres-full
    - gate-tempest-dsvm-large-ops
    - gate-tempest-dsvm-neutron-large-ops
    - gate-rally-dsvm-myscenario
```

It is also possible to arrange your input task files as templates based on jinja2. Say, you want to set the image names used throughout the *myscenario.yaml* task file as a variable parameter. Then, replace concrete image names in this file with a variable:

```
...

NovaServers.boot_and_delete_server:
  -
    args:
      image:
          name: {{image_name}}
    ...

NovaServers.boot_and_list_server:
  -
    args:
      image:
          name: {{image_name}}
    ...
```

and create a file named *myscenario_args.yaml* that will define the parameter values:

```
---

  image_name: "^cirros.*uec$"
```

this file will be automatically used by Rally to substitute the variables in *myscenario.yaml*.

1.8.4 Plugins & Extras in Rally Gate jobs

Along with scenario configs in yaml, the **rally-jobs** directory can also contain two subdirectories:

- **plugins**: *Plugins* needed for your gate job;
- **extra**: auxiliary files like bash scripts or images.

Both subdirectories will be copied to *~/.rally/* before the job gets started.

1.9 Request New Features

To request a new feature, you should create a document similar to other feature requests and then contribute it to the **doc/feature_request** directory of the Rally repository (see the *How-to-contribute tutorial*).

If you don't have time to contribute your feature request via gerrit, please contact Boris Pavlovic (boris@pavlovic.me)

Active feature requests:

1.9.1 Capture Logs from services

Use case

A developer is executing various task and would like to capture logs as well as test results.

Problem description

In case of errors it is quite hard to debug what happend.

Possible solution

- Add special context that can capture the logs from tested services.

1.9.2 Check queue perfdata

Use case ———— Sometimes OpenStack services use common messaging system very prodigally. For example neutron metering agent sending all database table data on new object creation i.e https://review.openstack.org/#/c/143672/. It cause to neutron degradation and other obvious problems. It will be nice to have a way to track messages count and messages size in queue during tests/benchmarks.

Problem description ——————— Heavy usage of queue isn't checked.

Possible solution ——————— * Before running tests/benchmarks start process which will connect to queue topics and measure messages count, size and other data which we need.

1.9.3 Ability to compare results between task

Use case

During the work on performance it's essential to be able to compare results of similar task before and after change in system.

Problem description

There is no command to compare two or more tasks and get tables and graphs.

Possible solution

- Add command that accepts 2 tasks UUID and prints graphs that compares result

1.9.4 Distributed load generation

Use Case

Some OpenStack projects (Marconi, MagnetoDB) require a real huge load, like 10-100k request per second for benchmarking.

To generate such huge load Rally have to create load from different servers.

Problem Description

- Rally can't generate load from different servers
- Result processing can't handle big amount of data
- There is no support for chunking results

1.9.5 Explicitly specify existing users for scenarios

Use Case

Rally allows to reuse existing users for scenario runs. And we should be able to use only specified set of existing users for specific scenarios.

Problem Description

For the moment if used *deployment* with existing users then Rally chooses user for each scenario run randomly. But there are cases when we may want to use one scenario with one user and another with different one specific user. Main reason for it is in different set of resources that each user has and those resources may be required for scenarios. Without this feature Rally user is forced to make all existing users similar and have all required resources set up for all scenarios he uses. But it is redundant.

Possible solution

- Make it possible to use explicitly existing_users context

1.9.6 Historical performance data

Use case

OpenStack is really rapidly developed. Hundreds of patches are merged daily and it's really hard to track how performance is changed during time. It will be nice to have a way to track performance of major functionality of OpenStack running periodically rally task and building graphs that represent how performance of specific method is changed during the time.

Problem description

There is no way to bind tasks

Possible solution

- Add grouping for tasks
- Add command that creates historical graphs

1.9.7 Enhancements to installation script: `--version` and `--uninstall`

Use case

User might wish to control which rally version is installed or even purge rally from the machine completely.

Problem description

1. Installation script doesn't allow to choose version.
2. No un-install support.

Possible solution

1. Add `--version` option to installation script.
2. Add `--uninstall` option to installation script or create an un-installation script

1.9.8 Installation script: `--pypi-mirror`, `--package-mirror` and `--venv-mirror`

Use case

Installation is pretty easy when there is an Internet connection available. And there is surely a number of Openstack uses when whole environment is isolated. In this case, we need somehow specify where installation script should take required libs and packages.

Problem description

1. Installation script can't work without direct Internet connection

Possible solution #1

1. Add `--pypi-mirror` option to installation script.
2. Add `--package-mirror` option to installation script.
3. Add `--venv-mirror` option to installation script.

1.9.9 Launch Specific Benchmark(s)

Use case

A developer is working on a feature that is covered by one or more specific benchmarks/scenarios. He/she would like to execute a rally task with an existing task template file (yaml or json) indicating exactly which benchmark(s) will be executed.

Problem description

When executing a task with a template file in Rally, all benchmarks are executed without the ability to specify one or a set of benchmarks the user would like to execute.

Possible solution

- Add optional flag to rally task start command to specify one or more

benchmarks to execute as part of that test run.

1.9.10 Using multi scenarios to generate load

Use Case

Rally should be able to generate real life load. Simultaneously create load on different components of OpenStack, e.g. simultaneously booting VM, uploading image and listing users.

Problem Description

At the moment Rally is able to run only 1 scenario per benchmark. Scenario are quite specific (e.g. boot and delete VM for example) and can't actually generate real life load.

Writing a lot of specific benchmark scenarios that will produce more real life load will produce mess and a lot of duplication of code.

Possible solution

- Extend Rally task benchmark configuration in such way to support passing multiple benchmark scenarios in single benchmark context
- Extend Rally task output format to support results of multiple scenarios in single benchmark separately.
- Extend rally task plot2html and rally task detailed to show results separately for every scenario.

1.9.11 Add support of persistence benchmark environment

Use Case

To benchmark many of operations like show, list, detailed you need to have already these resource in cloud. So it will be nice to be able to create benchmark environment once before benchmarking. So run some amount of benchmarks that are using it and at the end just delete all created resources by benchmark environment.

Problem Description

Fortunately Rally has already a mechanism for creating benchmark environment, that is used to create load. Unfortunately it's atomic operation: (create environment, make load, delete environment). This should be split to 3 separated steps.

Possible solution

- Add new CLI operations to work with benchmark environment: (show, create, delete, list)
- Allow task to start against benchmark environment (instead of deployment)

1.9.12 Production read cleanups

Use Case

Rally should delete in any case all resources that it created during benchmark.

Problem Description

- (implemented) Deletion rate limit

  You can kill cloud by deleting too many objects simultaneously, so deletion rate limit is required

- (implemented) Retry on failures

  There should be few attempts to delete resource in case of failures

- (implemented) Log resources that failed to be deleted

  We should log warnings about all non deleted resources. This information should include UUID of resource, it's type and project.

- (implemented) Pluggable

  It should be simple to add new cleanups adding just plugins somewhere.

- Disaster recovery

  Rally should use special name patterns, to be able to delete resources in such case if something went wrong with server that is running rally. And you have just new instance (without old rally db) of rally on new server.

1.10 Project Info

1.10.1 Maintainers

Project Team Lead (PTL)

If you would like to refactor whole Rally or have UX/community/other issues please contact me.

Project Core maintainers

Contact	Area of interest
Andrey Kurilin andreykurilin (irc) akurilin@mirantis.com	• Rally-Tempest Integration • Rally verify
Sergey Skripnick redixin (irc) sskripnick@mirantis.com	• Rally CI/CD • Rally deploy • Automation of everything
Mikhail Dubov msdubov (irc) mdubov@mirantis.com	• Rally docs • Rally info
Yair Fried yfried (irc) yfried@redhat.com	• Rally-Tempest integration • Rally task & benchmark
Chris St. Pierre stpierre (irc) cstpierr@cisco.com	• Rally task & benchmark • Bash guru ;)
Li Yingjun liyingjun (irc) yingjun.li@kylin-cloud.com	• Rally task & benchmark

All cores from this list are reviewing all changes that are proposed to Rally. To avoid duplication of efforts, please contact them before starting work on your code.

Plugin Core reviewers

Contact	Area of interest
Ivan Kolodyazhny e0ne (irc) e0ne@e0ne.info	• Cinder plugins
Nikita Konovalov NikitaKonovalov (irc) nkonovalov@mirantis.com	• Sahara plugins

All cores from this list are responsible for their component plugins. To avoid duplication of efforts, please contact them before starting working on your own plugins.

1.10.2 Useful links

- Source code
- Rally road map
- Project space
- Bugs
- Patches on review
- Meeting logs (server: **irc.freenode.net**, channel: **#openstack-meeting**)
- Release meeting logs (server: **irc.freenode.net**, channel: **#openstack-rally**)
- IRC logs (server: **irc.freenode.net**, channel: **#openstack-rally**)

1.10.3 Where can I discuss and propose changes?

- Our IRC channel: **#openstack-rally** on **irc.freenode.net**;
- Weekly Rally team meeting (in IRC): **#openstack-meeting** on **irc.freenode.net**, held on Mondays at 14:00 UTC;
- Weekly release meeting (in IRC): **#openstack-rally** on **irc.freenode.net**, held on Mondays at 13:00 UTC;
- Openstack mailing list: **openstack-dev@lists.openstack.org** (see subscription and usage instructions);
- Rally team on Launchpad: Answers/Bugs/Blueprints.

1.11 Release Notes

1.11.1 All release notes

Rally v0.0.1

Information

Commits	**1039**
Bug fixes	**0**
Dev cycle	**547 days**
Release date	**26/Jan/2015**

Details

Rally is awesome tool for testing verifying and benchmarking OpenStack clouds.

A lot of people started using Rally in their CI/CD so Rally team should provide more stable product with clear strategy of deprecation and upgrades.

Rally v0.0.2

Information

Commits	**100**
Bug fixes	**18**
Dev cycle	**45 days**
Release date	**12/Mar/2015**

Details

This release contains new features, new benchmark plugins, bug fixes, various code and API improvements.

New Features

- rally task start **–abort-on-sla-failure**

  Stopping load before things go wrong. Load generation will be interrupted if SLA criteria stop passing.

- Rally verify command supports multiple Tempest sources now.
- python34 support
- postgres DB backend support

API changes

- [new] **rally [deployment | verify | task] use** subcommand

  It should be used instead of root command **rally use**

- [new] Rally as a Lib API

  To avoid code duplication between Rally as CLI tool and Rally as a Service we decide to make Rally as a Lib as a common part between these 2 modes.

  Rally as a Service will be a daemon that just maps HTTP request to Rally as a Lib API.

- [deprecated] **rally use** CLI command
- [deprecated] Old Rally as a Lib API

  Old Rally API was quite mixed up so we decide to deprecate it

Plugins

- **Benchmark Scenario Runners**:

  [improved] Improved algorithm of generation load in **constant runner**

  Before we used processes to generate load, now it creates pool of processes (amount of processes is equal to CPU count) after that in each process use threads to generate load. So now you can easily generate load of 1k concurrent scenarios.

  [improved] Unify code of **constant** and **rps** runners

  [interface] Added **abort()** to runner's plugin interface

  New method **abort()** is used to immediately interrupt execution.

- **Benchmark Scenarios**:

  [new] DesignateBasic.create_and_delete_server

  [new] DesignateBasic.create_and_list_servers

  [new] DesignateBasic.list_servers

  [new] MistralWorkbooks.list_workbooks

  [new] MistralWorkbooks.create_workbook

  [new] Quotas.neutron_update

  [new] HeatStacks.create_update_delete_stack

  [new] HeatStacks.list_stacks_and_resources

  [new] HeatStacks.create_suspend_resume_delete_stac

  [new] HeatStacks.create_check_delete_stack

  [new] NeutronNetworks.create_and_delete_routers

  [new] NovaKeypair.create_and_delete_keypair

  [new] NovaKeypair.create_and_list_keypairs

  [new] NovaKeypair.boot_and_delete_server_with_keypair

  [new] NovaServers.boot_server_from_volume_and_live_migrate

  [new] NovaServers.boot_server_attach_created_volume_and_live_migrate

  [new] CinderVolumes.create_and_upload_volume_to_image

  [fix] CinderVolumes.create_and_attach_volume

  Pass optional **kwargs only to create server command

[fix] GlanceImages.create_image_and_boot_instances

Pass optional **kwargs only to create server command

[fix] TempestScenario.* removed stress cleanup.

Major issue is that tempest stress cleanup cleans whole OpenStack. This is very dangerous, so it's better to remove it and leave some extra resources.

[improved] NovaSecGroup.boot_and_delete_server_with_secgroups

Add optional **kwargs that are passed to boot server comment

- **Benchmark Context**:

[new] **stacks**

Generates passed amount of heat stacks for all tenants.

[new] **custom_image**

Prepares images for benchmarks in VMs.

To Support generating workloads in VMs by existing tools like: IPerf, Blogbench, HPCC and others we have to have prepared images, with already installed and configured tools.

Rally team decide to generate such images on fly from passed to avoid requirements of having big repository with a lot of images.

This context is abstract context that allows to automate next steps:

1. runs VM with passed image (with floating ip and other stuff)
2. execute abstract method that has access to VM
3. snapshot this image

In future we are going to use this as a base for making context that prepares images.

[improved] **allow_ssh**

Automatically disable it if security group are disabled in neutron.

[improved] **keypair**

Key pairs are stored in "users" space it means that accessing keypair from scenario is simpler now:

self.context["user"]["keypair"]["private"]

[fix] **users**

Pass proper EndpointType for newly created users

[fix] **sahara_edp**

The Job Binaries data should be treated as a binary content

- **Benchmark SLA**:

[interface] SLA calculations is done in additive way now

Resolves scale issues, because now we don't need to have whole array of iterations in memory to process SLA.

This is required to implement **–abort-on-sla-failure** feature

[all] SLA plugins were rewritten to implement new interface

Bug fixes **18 bugs were fixed, the most critical are**:

- Fix **rally task detailed –iterations-data**

  It didn't work in case of missing atomic actions. Such situation can occur if scenario method raises exceptions

- Add user-friendly message if the task cannot be deleted

  In case of trying to delete task that is not in "finished" status users get traces instead of user-friendly message try to run it with –force key.

- Network context cleanups networks properly now

Documentation

- Image sizes are fixed
- New tutorial in "Step by Step" relate to **–abort-on-sla-failure**
- Various fixes

Rally v0.0.3

Information

Commits	**53**
Bug fixes	**14**
Dev cycle	**33 days**
Release date	**14/Apr/2015**

Details

This release contains new features, new benchmark plugins, bug fixes, various code and API improvements.

New Features & API changes

- Add the ability to specify versions for clients in benchmark scenarios

  You can call self.clients("glance", "2") and get any client for specific version.

- Add API for tempest uninstall

  $ rally-manage tempest uninstall # removes fully tempest for active deployment

- Add a –uuids-only option to rally task list

  $ rally task list –uuids-only # returns list with only task uuids

- Adds endpoint to –fromenv deployment creation

  $ rally deployment create –fromenv # recognizes standard OS_ENDPOINT environment variable

- Configure SSL per deployment

  Now SSL information is deployment specific not Rally specific and rally.conf option is deprecated

  Like in this sample https://github.com/openstack/rally/blob/14d0b5ba0c75ececfdb6a6c121d9cf2810571f77/samples/deployn
  L12

Specs

- [spec] Proposal for new task input file format

  This spec describes new task input format that will allow us to generate multi scenario load which is crucial for HA and more real life testing:

  https://github.com/openstack/rally/blob/master/doc/specs/in-progress/new_rally_input_task_format.rst

Plugins

- **Benchmark Scenario Runners**:
  - Add a maximum concurrency option to rps runner

    To avoid running to heavy load you can set 'concurrency' to configuration and in case if cloud is not able to process all requests it won't start more parallel requests then 'concurrency' value.

- **Benchmark Scenarios**:

  [new] CeilometerAlarms.create_alarm_and_get_history

  [new] KeystoneBasic.get_entities

  [new] EC2Servers.boot_server

  [new] KeystoneBasic.create_and_delete_service

  [new] MuranoEnvironments.list_environments

  [new] MuranoEnvironments.create_and_delete_environment

  [new] NovaServers.suspend_and_resume_server

  [new] NovaServers.pause_and_unpause_server

  [new] NovaServers.boot_and_rebuild_server

  [new] KeystoneBasic.create_and_list_services

  [new] HeatStacks.list_stacks_and_events

  [improved] VMTask.boot_runcommand_delete

  restore ability to use fixed IP and floating IP to connect to VM via ssh

  [fix] NovaServers.boot_server_attach_created_volume_and_live_migrate

  Kwargs in nova scenario were wrongly passed

- **Benchmark SLA**:
  - [new] aborted_on_sla

  This is internal SLA criteria, that is added if task was aborted

  - [new] something_went_wrong

  This is internal SLA criteria, that is added if something went wrong, context failed to create or runner raised some exceptions

Bug fixes 14 bugs were fixed, the most critical are:

- Set default task uuid to running task. Before it was set only after task was fully finished.
- The "rally task results" command showed a disorienting "task not found" message for a task that is currently running.

- Rally didn't know how to reconnect to OpenStack in case if token expired.

Documentation

- New tutorial **task templates**

https://rally.readthedocs.org/en/latest/tutorial/step_5_task_templates.html

- Various fixes

Rally v0.0.4

Information

Commits	**87**
Bug fixes	**21**
Dev cycle	**30 days**
Release date	**14/May/2015**

Details

This release contains new features, new benchmark plugins, bug fixes, various code and API improvements.

New Features & API changes

- Rally now can generate load with users that already exist

  Now one can use Rally for benchmarking OpenStack clouds that are using LDAP, AD or any other read-only keystone backend where it is not possible to create any users. To do this, one should set up the "users" section of the deployment configuration of the ExistingCloud type. This feature also makes it safer to run Rally against production clouds: when run from an isolated group of users, Rally won't affect rest of the cloud users if something goes wrong.

- New decorator *@osclients.Clients.register* can add new OpenStack clients at runtime

  It is now possible to add a new OpenStack client dynamically at runtime. The added client will be available from osclients.Clients at the module level and cached. Example:

```
>>> from rally import osclients
>>> @osclients.Clients.register("supernova")
... def another_nova_client(self):
...   from novaclient import client as nova
...   return nova.Client("2", auth_token=self.keystone().auth_token,
...                      **self._get_auth_info(password_key="key"))
...
>>> clients = osclients.Clients.create_from_env()
>>> clients.supernova().services.list()[:2]
[<Service: nova-conductor>, <Service: nova-cert>]
```

- Assert methods now available for scenarios and contexts

  There is now a new *FunctionalMixin* class that implements basic unittest assert methods. The *base.Context* and *base.Scenario* classes inherit from this mixin, so now it is possible to use *base.assertX()* methods in scenarios and contexts.

- Improved installation script

  The installation script has been almost completely rewritten. After this change, it can be run from an unprivileged user, supports different database types, allows to specify a custom python binary, always asks confirmation before doing potentially dangerous actions, automatically install needed software if run as root, and also automatically cleans up the virtualenv and/or the downloaded repository if interrupted.

Specs & Feature requests

- [Spec] Reorder plugins

  The spec describes how to split Rally framework and plugins codebase to make it simpler for newbies to understand how Rally code is organized and how it works.

- [Feature request] Specify what benchmarks to execute in task

  This feature request proposes to add the ability to specify benchmark(s) to be executed when the user runs the *rally task start* command. A possible solution would be to add a special flag to the *rally task start* command.

Plugins

- **Benchmark Scenario Runners**:
  - Add limits for maximum Core usage to constant and rps runners

    The new 'max_cpu_usage' parameter can be used to avoid possible 100% usage of all available CPU cores by reducing the number of CPU cores available for processes started by the corresponding runner.

- **Benchmark Scenarios**:
  - [new] KeystoneBasic.create_update_and_delete_tenant
  - [new] KeystoneBasic.create_user_update_password
  - [new] NovaServers.shelve_and_unshelve_server
  - [new] NovaServers.boot_and_associate_floating_ip
  - [new] NovaServers.boot_lock_unlock_and_delete
  - [new] NovaHypervisors.list_hypervisors
  - [new] CeilometerSamples.list_samples
  - [new] CeilometerResource.get_resources_on_tenant
  - [new] SwiftObjects.create_container_and_object_then_delete_all
  - [new] SwiftObjects.create_container_and_object_then_download_object
  - [new] SwiftObjects.create_container_and_object_then_list_objects
  - [new] MuranoEnvironments.create_and_deploy_environment
  - [new] HttpRequests.check_random_request
  - [new] HttpRequests.check_request
  - [improved] NovaServers live migrate benchmarks

    add 'min_sleep' and 'max_sleep' parameters to simulate a pause between VM booting and running live migration
  - [improved] NovaServers.boot_and_live_migrate_server

    add a usage sample to samples/tasks

  - [improved] CinderVolumes benchmarks

    support size range to be passed to the 'size' argument as a dictionary *{"min": <minimum_size>, "max": <maximum_size>}*

- **Benchmark Contexts**:
  - [new] MuranoPackage

    This new context can upload a package to Murano from some specified path.

  - [new] CeilometerSampleGenerator

    Context that can be used for creating samples and collecting resources for benchmarks in a list.

- **Benchmark SLA**:
  - [new] outliers

    This new SLA checks that the number of outliers (calculated from the mean and standard deviation of the iteration durations) does not exceed some maximum value. The SLA is highly configurable: the parameters used for outliers threshold calculation can be set by the user.

Bug fixes 21 bugs were fixed, the most critical are:

- Make it possible to use relative imports for plugins that are outside of rally package.
- Fix heat stacks cleanup by deleting them only 1 time per tenant (get rid of "stack not found" errors in logs).
- Fix the wrong behavior of 'rally task detailed –iterations-data' (it lacked the iteration info before).
- Fix security groups cleanup: a security group called "default", created automatically by Neutron, did not get deleted for each tenant.

Other changes

- Streaming algorithms that scale

  This release introduces the common/streaming_algorithms.py module. This module is going to contain implementations of benchmark data processing algorithms that scale: these algorithms do not store exhaustive information about every single benchmark iteration duration processed. For now, the module contains implementations of algorithms for computation of mean & standard deviation.

- Coverage job to check that new patches come with unit tests

  Rally now has a coverage job that checks that every patch submitted for review does not decrease the number of lines covered by unit tests (at least too much). This job allows to mark most patches with no unit tests with '-1'.

- Splitting the plugins code (Runners & SLA) into common/openstack plugins

  According to the spec "Reorder plugins" (see above), the plugins code for runners and SLA has been moved to the *plugins/common/* directory. Only base classes now remain in the *benchmark/* directory.

Documentation

- Various fixes
  - Remove obsolete *.rst* files (*deploy_engines.rst* / *server_providers.rst* / ...)
  - Restructure the docs files to make them easier to navigate through
  - Move the chapter on task templates to the 4th step in the tutorial
  - Update the information about meetings (new release meeting & time changes)

Rally v0.1.0

Information

Commits	**355**
Bug fixes	**90**
Dev cycle	**132 days**
Release date	**25/September/2015**

Details

This release contains new features, new 42 plugins, 90 bug fixes, various code and API improvements.

New Features & API changes

- **Improved installation script**
  - Add parameters:
    - `--develop` parameter to install rally in editable (develop) mode
    - `--no-color` to switch off output colorizing useful for automated output parsing and terminals that don't support colors.
  - Puts rally.conf under virtualenv etc/rally/ so you can have several rally installations in virtualenv
  - Many fixes related to access of different file, like: rally.conf, rally db file in case of sqlite
  - Update pip before Rally installation
  - Fix reinstallation
- **Separated Rally plugins & framework**

  Now plugins are here: https://github.com/openstack/rally/tree/master/rally/plugins

  Plugins are as well separated common/* for common plugins that can be use no matter what is tested and OpenStack related plugins
- **New Rally Task framework**
  - All plugins has the same Plugin base: rally.common.plugin.pluing.Plugin They have the same mechanisms for: discovering, providing information based on docstrings, and in future they will use the same deprecation/rename mechanism.
  - Some of files are moved:
    - rally/benchmark -> rally/task

      This was done to unify naming of rally task command and actually code that implements it.
    - rally/benchmark/sla/base.py -> rally/task/sla.py
    - rally/benchmark/context/base.py -> rally/task/context.py
    - rally/benchmark/scenarios/base.py -> rally/task/scenario.py
    - rally/benchmark/runners/base.py -> rally/task/runner.py
    - rally/benchmark/scenarios/utils.py -> rally/task/utils.py

    This was done to:

    - avoid doing rally.benchamrk.scenarios import base as scenario_base
    - remove one level of nesting
    - simplify framework structure
- Some of classes and methods were renamed
    - Plugin configuration:
        - context.context() -> context.configure()
        - scenario.scenario() -> scenario.configure()
        - Introduced runner.configure()
        - Introduced sla.configure()

      This resolves 3 problems:
        - Unifies configuration of different types of plugins
        - Simplifies plugin interface
        - **Looks nice with new modules path:**

```
>>> from rally.task import scenario
>>> @scenario.configure()
```

- Atomic Actions were changed:
    - New rally.task.atomic module

      This allow us in future to reuse atomic actions in Context plugins
    - Renames:

      rally.benchmark.scenarios.base.AtomicAction -> rally.task.atomic.ActionTimer

      rally.benchmark.scenarios.base.atomic_action() -> rally.task.atomic.action_timer()
- **Context plugins decide how to map their data for scenario**

  Now Context.map_for_scenario method can be override to decide how to pass context object to each iteration of scenario.
- Samples of NEW vs OLD context, sla, scenario and runner plugins:
    - Context

```
# Old
from rally.benchmark.context import base

@base.context(name="users", order=100)
class YourContext(base.Context):

    def setup(self):
        # ...

    def cleanup(self):
        # ...

# New
from rally.task import context

@context.configure(name="users", order=100)
```

```
class YourContext(context.Context):

    def setup(self):
        # ...

    def cleanup(self):
        # ...

    def map_for_scenario(self):
        # Maps context object to the scenario context object
        # like context["users"] -> context["user"] and so on.
```

* Scenario

```
# Old Scenario

from rally.benchmark.scenarios import base
from rally.benchmark import validation

class ScenarioPlugin(base.Scenario):

    @base.scenario()
    def some(self):
        self._do_some_action()


    @base.atomic_action_timer("some_timer")
    def _do_some_action(self):
        # ...

# New Scenario

from rally.task import atomic
from rally.task import scenario
from rally.task import validation

# OpenStack scenario has different base now:
# rally.plugins.openstack.scenario.OpenStackScenario
class ScenarioPlugin(scenario.Scenario):

    @scenario.configure()
    def some(self):
        self._do_some_action()

    @atomic.action_timer("some_action")
    def _do_some_action(self):
        # ...
```

* Runner

```
## Old

from rally.benchmark.runners import base

class SomeRunner(base.ScenarioRunner):

     __execution_type__ = "some_runner"

    def _run_scenario(self, cls, method_name, context, args)
```

```
        # Load generation

    def abort(self):
        # Method that aborts load generation

## New

from rally.task import runner

@runner.configure(name="some_runner")
class SomeRunner(runner.ScenarioRunner):

    def _run_scenario(self, cls, method_name, context, args)
        # Load generation

    def abort(self):
        # Method that aborts load generation
```

  * SLA

```
# Old

from rally.benchmark import sla

class FailureRate(sla.SLA):
    # ...

# New

from rally.task import sla

@sla.configure(name="failure_rate")
class FailureRate(sla.SLA):
    # ...
```

- **Rally Task aborted command**

  Finally you can gracefully shutdown running task by calling:

  `rally task abort <task_uuid>`

- **Rally CLI changes**
  - [add] `rally --plugin-paths` specify the list of directories with plugins
  - [add] `rally task report --junit` - generate a JUnit report This allows users to feed reports to tools such as Jenkins.
  - [add] `rally task abort` - aborts running Rally task when run with the `--soft` key, the `rally task abort` command is waiting until the currently running subtask is finished, otherwise the command interrupts subtask immediately after current scenario iterations are finished.
  - [add] `rally plugin show` prints detailed information about plugin
  - [add] `rally plugin list` prints table with rally plugin names and titles
  - [add] `rally verify genconfig` generates tempest.conf without running it.
  - [add] `rally verify install` install tempest for specified deployment
  - [add] `rally verify reinstall` removes tempest for specified deployment

  - [add] `rally verify uninstall` uninstall tempest of specified deployment
  - [fix] `rally verify start --no-use` –no-use was always turned on
  - [remove] `rally use` now each command has subcommand `use`
  - [remove] `rally info`
  - [remove] `rally-manage tempest` now it is covered by `rally verify`
- **New Rally task reports**
  - New code is based on OOP style which is base step to make plugable Reports
  - Reports are now generated for only one iteration over the resulting data which resolves scalability issues when we are working with large amount of iterations.
  - New Load profiler plot that shows amount of iterations that are working in parallel
  - Failed iterations are shown as a red areas on stacked are graphic.

Non backward compatible changes

- [remove] `rally use` cli command
- [remove] `rally info` cli command
- [remove] `--uuid` parameter from `rally deployment <any>`
- [remove `--deploy-id` parameter from: `rally task <any>`, `rally verify <any>`, `rally show <any>`

Specs & Feature requests

[feature request] Explicitly specify existing users for scenarios

[feature request] Improve install script and add –unistall and –version

[feature request] Allows specific repos & packages in install-rally.sh

[feature request] Add abbility to caputre logs from tested services

[feature request] Check RPC queue perfdata

[spec] Refactoring Rally cleanup

[spec] Consistent resource names

Plugins

- **Scenarios**:

  [new] CinderVolumes.create_volume_backup

  [new] CinderVolumes.create_and_restore_volume_backup

  [new] KeystoneBasic.add_and_remove_user_role

  [new] KeystoneBasic.create_and_delete_role

  [new] KeystoneBasic.create_add_and_list_user_roles

  [new] FuelEnvironments.list_environments

  [new] CinderVolumes.modify_volume_metadata

[new] NovaServers.boot_and_delete_multiple_servers

[new] NeutronLoadbalancerV1.create_and_list_pool

[new] ManilaShares.list_shares

[new] CeilometerEvents.create_user_and_get_event

[new] CeilometerEvents.create_user_and_list_event_types

[new] CeilometerEvents.create_user_and_list_events

[new] CeilometerTraits.create_user_and_list_trait_descriptions

[new] CeilometerTraits.create_user_and_list_traits

[new] NeutronLoadbalancerV1.create_and_delete_pools

[new] NeutronLoadbalancerV1.create_and_update_pools

[new] ManilaShares.create_and_delete_share

[new] ManilaShares.create_share_network_and_delete

[new] ManilaShares.create_share_network_and_list

[new] HeatStacks.create_and_delete_stack

[new] ManilaShares.list_share_servers

[new] HeatStacks.create_snapshot_restore_delete_stack

[new] KeystoneBasic.create_and_delete_ec2credential

[new] KeystoneBasic.create_and_list_ec2credentials

[new] HeatStacks.create_stack_and_scale

[new] ManilaShares.create_security_service_and_delete

[new] KeystoneBasic.create_user_set_enabled_and_delete

[new] ManilaShares.attach_security_service_to_share_network

[new] IronicNodes.create_and_delete_node

[new] IronicNodes.create_and_list_node

[new] CinderVolumes.create_and_list_volume_backups

[new] NovaNetworks.create_and_list_networks

[new] NovaNetworks.create_and_delete_network

[new] EC2Servers.list_servers

[new] VMTasks.boot_runcommand_delete_custom_imagea

[new] CinderVolumes.create_and_update_volume

- **Contexts**:

  [new] ManilaQuotas

  > Add context for setting up Manila quotas: shares, gigabytes, snapshots, snapshot_gigabytes, share_networks

  [new] ManilaShareNetworks

Context for share networks that will be used in case of usage deployment with existing users. Provided share networks via context option "share_networks" will be balanced between all share creations of scenarios.

[new] Lbaas

Context to create LBaaS-v1 resources

[new] ImageCommandCustomizerContext

Allows image customization using side effects of a command execution. E.g. one can install an application to the image and use these image for 'boot_runcommand_delete' scenario afterwards.

[new] EC2ServerGenerator

Context that creates servers using EC2 api

[new] ExistingNetwork

This context lets you use existing networks that have already been created instead of creating new networks with Rally. This is useful when, for instance, you are using Neutron with a dumb router that is not capable of creating new networks on the fly.

- **SLA**:

  [remove] max_failure_rate - use failure_rate instead

Bug fixes 90 bugs were fixed, the most critical are:

- Many fixes related that fixes access of rally.conf and DB files
- Incorrect apt-get "-yes" parameter in install_rally.sh script
- Rally bash completion doesn't exist in a virtualenv
- Rally show networks CLI command worked only with nova networks
- RPS runner was not properly generating load
- Check is dhcp_agent_scheduler support or not in network cleanup
- NetworkContext doesn't work with Nova V2.1
- Rally task input file was not able to use jinja2 include directive
- Rally in docker image was not able to
- Rally docker image didn't contain samples
- Do not update the average duration when iteration failed

Documentation

- **Add plugin reference page**

  Rally Plugins Reference page page contains a full list with

- **Add maintainers section on project info page**

  Rally Maintainers section contains information about core contributors of OpenStack Rally their responsibilities and contacts. This will help us to make our community more transparent and open for newbies.

- **Added who is using section in docs**
- **Many small fixes**

1.11.2 Rally v0.1.0

Information

Commits	**355**
Bug fixes	**90**
Dev cycle	**132 days**
Release date	**25/September/2015**

Details

This release contains new features, new 42 plugins, 90 bug fixes, various code and API improvements.

New Features & API changes

- **Improved installation script**
  - Add parameters:
    - `--develop` parameter to install rally in editable (develop) mode
    - `--no-color` to switch off output colorizing useful for automated output parsing and terminals that don't support colors.
  - Puts rally.conf under virtualenv etc/rally/ so you can have several rally installations in virtualenv
  - Many fixes related to access of different file, like: rally.conf, rally db file in case of sqlite
  - Update pip before Rally installation
  - Fix reinstallation
- **Separated Rally plugins & framework**

  Now plugins are here: https://github.com/openstack/rally/tree/master/rally/plugins

  Plugins are as well separated common/* for common plugins that can be use no matter what is tested and OpenStack related plugins
- **New Rally Task framework**
  - All plugins has the same Plugin base: rally.common.plugin.pluing.Plugin They have the same mechanisms for: discovering, providing information based on docstrings, and in future they will use the same deprecation/rename mechanism.
  - Some of files are moved:
    - rally/benchmark -> rally/task

      This was done to unify naming of rally task command and actually code that implements it.
    - rally/benchmark/sla/base.py -> rally/task/sla.py
    - rally/benchmark/context/base.py -> rally/task/context.py
    - rally/benchmark/scenarios/base.py -> rally/task/scenario.py
    - rally/benchmark/runners/base.py -> rally/task/runner.py
    - rally/benchmark/scenarios/utils.py -> rally/task/utils.py

    This was done to:

    - avoid doing rally.benchamrk.scenarios import base as scenario_base
    - remove one level of nesting
    - simplify framework structure
- Some of classes and methods were renamed
    - Plugin configuration:
        - context.context() -> context.configure()
        - scenario.scenario() -> scenario.configure()
        - Introduced runner.configure()
        - Introduced sla.configure()

      This resolves 3 problems:
        - Unifies configuration of different types of plugins
        - Simplifies plugin interface
        - **Looks nice with new modules path:**

```
>>> from rally.task import scenario
>>> @scenario.configure()
```

- Atomic Actions were changed:
    - New rally.task.atomic module

      This allow us in future to reuse atomic actions in Context plugins
    - Renames:

      rally.benchmark.scenarios.base.AtomicAction -> rally.task.atomic.ActionTimer

      rally.benchmark.scenarios.base.atomic_action() -> rally.task.atomic.action_timer()
- **Context plugins decide how to map their data for scenario**

  Now Context.map_for_scenario method can be override to decide how to pass context object to each iteration of scenario.
- Samples of NEW vs OLD context, sla, scenario and runner plugins:
    - Context

```
# Old
from rally.benchmark.context import base

@base.context(name="users", order=100)
class YourContext(base.Context):

    def setup(self):
        # ...

    def cleanup(self):
        # ...

# New
from rally.task import context

@context.configure(name="users", order=100)
```

```
class YourContext(context.Context):

    def setup(self):
        # ...

    def cleanup(self):
        # ...

    def map_for_scenario(self):
        # Maps context object to the scenario context object
        # like context["users"] -> context["user"] and so on.
```

* Scenario

```
# Old Scenario

from rally.benchmark.scenarios import base
from rally.benchmark import validation

class ScenarioPlugin(base.Scenario):

    @base.scenario()
    def some(self):
        self._do_some_action()


    @base.atomic_action_timer("some_timer")
    def _do_some_action(self):
        # ...

# New Scenario

from rally.task import atomic
from rally.task import scenario
from rally.task import validation

# OpenStack scenario has different base now:
# rally.plugins.openstack.scenario.OpenStackScenario
class ScenarioPlugin(scenario.Scenario):

    @scenario.configure()
    def some(self):
        self._do_some_action()

    @atomic.action_timer("some_action")
    def _do_some_action(self):
        # ...
```

* Runner

```
## Old

from rally.benchmark.runners import base

class SomeRunner(base.ScenarioRunner):

     __execution_type__ = "some_runner"

    def _run_scenario(self, cls, method_name, context, args)
```

```
        # Load generation

    def abort(self):
        # Method that aborts load generation

## New

from rally.task import runner

@runner.configure(name="some_runner")
class SomeRunner(runner.ScenarioRunner):

    def _run_scenario(self, cls, method_name, context, args)
        # Load generation

    def abort(self):
        # Method that aborts load generation
```

  * SLA

```
# Old

from rally.benchmark import sla

class FailureRate(sla.SLA):
    # ...

# New

from rally.task import sla

@sla.configure(name="failure_rate")
class FailureRate(sla.SLA):
    # ...
```

- **Rally Task aborted command**

  Finally you can gracefully shutdown running task by calling:

  `rally task abort <task_uuid>`

- **Rally CLI changes**
  - [add] `rally --plugin-paths` specify the list of directories with plugins
  - [add] `rally task report --junit` - generate a JUnit report This allows users to feed reports to tools such as Jenkins.
  - [add] `rally task abort` - aborts running Rally task when run with the `--soft` key, the `rally task abort` command is waiting until the currently running subtask is finished, otherwise the command interrupts subtask immediately after current scenario iterations are finished.
  - [add] `rally plugin show` prints detailed information about plugin
  - [add] `rally plugin list` prints table with rally plugin names and titles
  - [add] `rally verify genconfig` generates tempest.conf without running it.
  - [add] `rally verify install` install tempest for specified deployment
  - [add] `rally verify reinstall` removes tempest for specified deployment

  - [add] `rally verify uninstall` uninstall tempest of specified deployment
  - [fix] `rally verify start --no-use` –no-use was always turned on
  - [remove] `rally use` now each command has subcommand `use`
  - [remove] `rally info`
  - [remove] `rally-manage tempest` now it is covered by `rally verify`
- **New Rally task reports**
  - New code is based on OOP style which is base step to make plugable Reports
  - Reports are now generated for only one iteration over the resulting data which resolves scalability issues when we are working with large amount of iterations.
  - New Load profiler plot that shows amount of iterations that are working in parallel
  - Failed iterations are shown as a red areas on stacked are graphic.

Non backward compatible changes

- [remove] `rally use` cli command
- [remove] `rally info` cli command
- [remove] `--uuid` parameter from `rally deployment <any>`
- [remove `--deploy-id` parameter from: `rally task <any>`, `rally verify <any>`, `rally show <any>`

Specs & Feature requests

[feature request] Explicitly specify existing users for scenarios

[feature request] Improve install script and add –unistall and –version

[feature request] Allows specific repos & packages in install-rally.sh

[feature request] Add abbility to caputre logs from tested services

[feature request] Check RPC queue perfdata

[spec] Refactoring Rally cleanup

[spec] Consistent resource names

Plugins

- **Scenarios**:

  [new] CinderVolumes.create_volume_backup

  [new] CinderVolumes.create_and_restore_volume_backup

  [new] KeystoneBasic.add_and_remove_user_role

  [new] KeystoneBasic.create_and_delete_role

  [new] KeystoneBasic.create_add_and_list_user_roles

  [new] FuelEnvironments.list_environments

[new] CinderVolumes.modify_volume_metadata

[new] NovaServers.boot_and_delete_multiple_servers

[new] NeutronLoadbalancerV1.create_and_list_pool

[new] ManilaShares.list_shares

[new] CeilometerEvents.create_user_and_get_event

[new] CeilometerEvents.create_user_and_list_event_types

[new] CeilometerEvents.create_user_and_list_events

[new] CeilometerTraits.create_user_and_list_trait_descriptions

[new] CeilometerTraits.create_user_and_list_traits

[new] NeutronLoadbalancerV1.create_and_delete_pools

[new] NeutronLoadbalancerV1.create_and_update_pools

[new] ManilaShares.create_and_delete_share

[new] ManilaShares.create_share_network_and_delete

[new] ManilaShares.create_share_network_and_list

[new] HeatStacks.create_and_delete_stack

[new] ManilaShares.list_share_servers

[new] HeatStacks.create_snapshot_restore_delete_stack

[new] KeystoneBasic.create_and_delete_ec2credential

[new] KeystoneBasic.create_and_list_ec2credentials

[new] HeatStacks.create_stack_and_scale

[new] ManilaShares.create_security_service_and_delete

[new] KeystoneBasic.create_user_set_enabled_and_delete

[new] ManilaShares.attach_security_service_to_share_network

[new] IronicNodes.create_and_delete_node

[new] IronicNodes.create_and_list_node

[new] CinderVolumes.create_and_list_volume_backups

[new] NovaNetworks.create_and_list_networks

[new] NovaNetworks.create_and_delete_network

[new] EC2Servers.list_servers

[new] VMTasks.boot_runcommand_delete_custom_imagea

[new] CinderVolumes.create_and_update_volume

- **Contexts**:

  [new] ManilaQuotas

  > Add context for setting up Manila quotas: shares, gigabytes, snapshots, snapshot_gigabytes, share_networks

  [new] ManilaShareNetworks

Context for share networks that will be used in case of usage deployment with existing users. Provided share networks via context option "share_networks" will be balanced between all share creations of scenarios.

[new] Lbaas

Context to create LBaaS-v1 resources

[new] ImageCommandCustomizerContext

Allows image customization using side effects of a command execution. E.g. one can install an application to the image and use these image for 'boot_runcommand_delete' scenario afterwards.

[new] EC2ServerGenerator

Context that creates servers using EC2 api

[new] ExistingNetwork

This context lets you use existing networks that have already been created instead of creating new networks with Rally. This is useful when, for instance, you are using Neutron with a dumb router that is not capable of creating new networks on the fly.

- **SLA**:

  [remove] max_failure_rate - use failure_rate instead

Bug fixes

90 bugs were fixed, the most critical are:

- Many fixes related that fixes access of rally.conf and DB files
- Incorrect apt-get "-yes" parameter in install_rally.sh script
- Rally bash completion doesn't exist in a virtualenv
- Rally show networks CLI command worked only with nova networks
- RPS runner was not properly generating load
- Check is dhcp_agent_scheduler support or not in network cleanup
- NetworkContext doesn't work with Nova V2.1
- Rally task input file was not able to use jinja2 include directive
- Rally in docker image was not able to
- Rally docker image didn't contain samples
- Do not update the average duration when iteration failed

Documentation

- **Add plugin reference page**

  Rally Plugins Reference page page contains a full list with

- **Add maintainers section on project info page**

  Rally Maintainers section contains information about core contributors of OpenStack Rally their responsibilities and contacts. This will help us to make our community more transparent and open for newbies.

- **Added who is using section in docs**

- **Many small fixes**