Enos Documentation

Release 3.0.0

discovery

Nov 03, 2017

Contents:

1 Enos workflow **3**
- 1.1 Getting Started . . . 3
- 1.2 Provider . . . 4
- 1.3 Enos command line . . . 12
- 1.4 Benchmarks . . . 12
- 1.5 Customizations . . . 14
- 1.6 Network Emulation . . . 16
- 1.7 Analysis . . . 18
- 1.8 Contribute . . . 19

2 Why Enos ? **21**

3 License **23**

4 Indices and tables **25**

Hint: The source code is available at https://github.com/BeyondTheClouds/enos

Enos deploys OpenStack and targets reproducible experiments. It allows easy:

- deployment of the system
- customization of the system
- benchmarking of the system
- visualization of various metrics

Enos is developed in the context of the Discovery initiative.

CHAPTER 1

Enos workflow

A typical experiment using Enos is the sequence of several phases:

- `enos up` : Enos will read the configuration file, get machines from the resource provider and will prepare the next phase
- `enos os` : Enos will deploy OpenStack on the machines. This phase rely highly on Kolla deployment.
- `enos init-os` : Enos will bootstrap the OpenStack installation (default quotas, security rules, ...)
- `enos bench` : Enos will run a list of benchmarks. Enos support Rally and Shaker benchmarks.
- `enos backup` : Enos will backup metrics gathered, logs and configuration files from the experiment.

1.1 Getting Started

1.1.1 Installation

```
$ pip install enos
```

You may prefer to go with a virtualenv. Please refer to the virtualenv documentation and the rest of this section for further information.

If virtualenv is missing:

```
$ pip install virtualenv --user     # Install virtualenv
$ export PATH=~/.local/bin/:${PATH} # Put it into your path
```

Then install enos inside a virtualenv:

```
$ mkdir my-experiment && cd my-experiment
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install enos
```

Note: The latest *packaged* version of enos will install the latest *stable* version of OpenStack. If you want to install the development version of OpenStack, you should install enos from sources (see *Contribute*).

1.1.2 Configuration

To get started you can get the sample configuration file and edit it:

```
$ enos new > reservation.yaml
$ <editor> reservation.yaml
```

The configuration may vary from one provider to another, please refer to the dedicated *Provider* configuration

Note: If a key is defined several times in the configuration file, only the last occurence will be taken into account. In particular to switch from one provider to another, you can move down the key `provider` and its associated `resources` key.

1.1.3 Deployment

Once your configuration is done, you can launch the deployment :

```
(venv) $ enos deploy
```

The deployment is the combination of the following three phases:

1. Acquire the raw resources that are necessary for the deployment of OpenStack. Enos acquires resources according to the `provider` and `resources` information in the reservation file. One can perform this phase by calling `enos up`.
2. Deploy OpenStack to the resources acquired during the previous phase. Enos uses the resource list provided by the previous phase and combines it with the information specified in the file targeted by the `inventory` key to produce a file that gives a mapping of which OpenStack services have to be deployed to which resources. Enos then calls the Kolla-Ansible tool with this file to deploy the containerized OpenStack services to the right resources. One can perform this phase by calling `enos os`.

   Note: If you don't provide an `inventory` in your current working directory, then Enos uses a default one. You can view it on GitHub at enos/inventories/inventory.sample. Note that the produced file is available at `cwd/current/multinode` (with `cwd` referencing to your current working directory).

3. Initialize the freshly deployed OpenStack. Enos initializes OpenStack with the bare necessities, i.e., install a `member` role, download and install a cirros image, install default flavors (m1.tiny, ..., m1.xlarge) and setup a network (one public/one private). One can perform this phase by calling `enos init`.

1.2 Provider

Enos offers to test different OpenStack deployments over some resources. In the context of Enos, a resource is anything Enos can SSH on and start a Docker daemon. Thus, a resource could be a bare-metal machine, a virtual machine, or a container resource depending on the testbed used for conduction the experiments. To get these resources, Enos relies on a notion of *provider* and already implements the followings:

1.2.1 Static

The static provider reuses already available resources (machines, network) to deploy OpenStack on.

Installation

Refer to the *Getting Started* section to install Enos.

Configuration

The static provider requires already running resources to deploy OpenStack on. Information in the provider description tells Enos where these resources are and how to access to them. Concretely, you have to ensure that following information are present in the `reservation.yaml` file from the *Getting Started* section to properly configure Enos with already running resources.

```
provider:  # Configuration for the static provider
           # All keys are mandatory
  type: static
  network:
    extra_ips: # Array of (virtual) IPs to be assigned
               # during the deployment (e.g HAProxy)
               # At least 5 IPs routable through the
               # APIs interface
               - 198.51.100.49
               - 198.51.100.50
               - 198.51.100.51
               - 198.51.100.52
               - 198.51.100.53
    # Openstack external network will be configured according
    # to the following keys/values. IPs have to be routable through
    # the external interface
    start:   203.0.113.17     # First available IP
    end:     203.0.113.31     # Last available IP
    cidr:    203.0.113.16/28  # cidr notation to describe the pool of IPs
    gateway: 203.0.113.254    # IP address of the gateway
    dns:     203.0.113.253    # IP address of the DNS
  eths:
    - eth1   # Name of the APIs interface
    - eth2   # Name of the external interface
resources: # An object with roles and there associated resources
  control:   # The role control, followed by resource info:
    address:   192.0.2.17  # IP of the control node
    alias:     control   # Name for the node (optional, default to address)
    user:      user      # ssh user name (optional, default to ssh config)
    keyfile:   keyfile   # path to ssh private key (optional, default to ssh config)
    port:      22        # ssh port (optional, default to ssh config)
    extra:   # Extra variables passed to ansible (optional, default to none)
      ansible_become: yes  # (optional, use "yes" if the user need to use sudo)
  compute:   # A role can also have a list of resources
    - address: 192.0.2.33  # IP address of the first compute
      alias:   compute1  # ...
      user:    user      # ...
      keyfile: keyfile   # ...
      port:    22        # ...
    - address: 192.0.2.34  # IP address of the second compute
    - address: 192.0.2.35  # IP address of the third compute
```

```
  network:
    - address: 192.0.2.17  # IP address of the first network host
```

All the hosts must have an IP address already configured on the APIs interface matching the networks defined in `reservation.yml` under `network`, for example: 198.51.100.17 for the `control` host, 198.51.100.33 for the `compute1` host and so on.

In the `resources` there must be at least one host entry for each of the following names:

- control
- compute
- network

The same host can be assigned to different `resources`. In this case using the `yaml` syntax with anchors and alias will help you to reach a DRY configuration:

```
# Four our convenience we define all the hosts here, and then map them
# to the resources, keys are arbitrary, yaml syntax does not support
# anchoring an item in a list

hosts:
  1: &h1
    address: 192.0.2.17
    user: ubuntu
    extra:
      ansible_become: yes
  2: &h2
    address: 192.0.2.33
    user: ubuntu
    extra:
      ansible_become: yes
  3: &h3
    address: 192.0.2.34
    user: ubuntu
    extra:
      ansible_become: yes

resources:
  control:
    - *h1
  compute:
    - *h2
    - *h3
  network:
    - *h1
  storage:
    - *h2
    - *h3
```

1.2.2 Grid'5000

Installation

Connect to the frontend of your choice. Then, refer to the *Getting Started* section to install Enos.

Configuration

The provider relies on cluster names to group the wanted resources. For example the following is a valid resources description :

```
provider: g5k

resources:
  paravance:
    control: 1
    network: 1
  parsilo:
    compute: 10
```

Deployment

We suggest you to run the deployment from a dedicated node (especially for large deployment). For now the recommended way to do so is to reserve one node prior of your reservation. In the case of an interactive deployment:

```
frontend) oarsub -I -l 'walltime=2:00:00'
node) source venv/bin/activate
node) <edit configuration>
node) enos deploy
```

Enos will need to access the G5K api from this node. In order to do so you need to instruct execo to use the external endpoint of the API :

```
cat ~/.execo.conf.py:

g5k_configuration = {
    'api_username': '<your login>',
    'api_uri': "https://api-ext.grid5000.fr/3.0/",
}
```

You will be asked for your password at start.

Default provider configuration

The provider comes with the following default options :

```
provider:
  type: g5k
  name: Enos
  walltime: 02:00:00
  env_name: jessie-x64-min
  reservation: None
  vlans: '{rennes: "{type=kavlan}/vlan=1"}'
  role_distribution: strict
  single_interface: false
  user: root
```

They can be overriden in the configuration file.

1.2.3 Vagrant

Installation

To get started with the vagrant provider, you need to install

- Vagrant

Then, refer to the *Getting Started* section to install Enos.

Configuration

The provider relies on virtual machine sizes to group the wanted resources. For example the following is a valid resources description:

```
provider: vagrant

resources:
  medium:
    control: 1
    network: 1
  small:
    compute:1
```

The list of the sizes may be found here.

By default virtualbox will be used. See below to learn how to change the default virtualbox backend.

Use libvirt as the backend for Vagrant

Declaring your provider options as the following will spin up virtual machines using libvirt. Note that vagrant libvirt must be installed on your system.

```
provider:
  type: vagrant
  backend: libvirt
```

Default provider configuration

The provider comes with the following default options:

```
provider:
  type: vagrant
  backend: virtualbox
  box: debian/jessie64
  user: root
```

They can be overriden in the configuration file.

1.2.4 Openstack

The OpenStack provider allows you to use Enos on an OpenStack cloud. In other words this lets you run OpenStack on OpenStack. In the following, the under-cloud is the underlying OpenStack infrastructure, the over-cloud is the OpenStack configured by Enos.

The over-cloud configured by Enos needs a set of resources to be present on the under-cloud. The first step in the deployment workflow consists in checking or creating such resources. Some resources are mandatory and must be present before the deployment (base image, keypairs, ...), some others can be created or reused during the deployment (private networks). For the latter, you can use the default values set by the provider.

For specific under-clouds (e.g Chameleon), specific providers deriving from the OpenStack provider may be used. They will enforce more default values that fit the under-cloud specificities (e.g specific DNS, base image, ...)

Installation

Please refer to *Getting Started*.

Configuration

The provider relies on flavor names to group the wanted resources. For example the following is probably a valid resources description.

```
provider:
  type: openstack
  <options see below>

resources:
  m1.medium:
    control: 1
    network: 1
  m1.small:
    compute: 10
```

Default provider configuration

The OpenStack provider is shipped with the following default options. These options will be set automatically and thus may be omitted in the configuration file.

```
provider:
  type: openstack
  # True if Enos needs to create a dedicated network to work with
  # False means you already have a network, subnet and router to
  # your ext_net configured
  configure_network: true

  # Name of the network to use or to create
  # It will be use as external network for the upper-cloud
  network: {'name': 'enos-network'}

  # Name of the subnet to use or to create
  subnet: {'name': 'enos-subnet', 'cidr': '10.87.23.0/24'}

  # DNS server to use when creating the network
  dns_nameservers: ['8.8.8.8', '8.8.4.4']

  # Floating ips pool
  allocation_pool: {'start': '10.87.23.10', 'end': '10.87.23.100'}

  # Whether one machine must act as gateway
  # - False means that you can connect directly to all the machines
```

```
  # started by Enos
  # - True means that one machine will be assigned a floating ip and used
  # as gateway to the others
  gateway: true
```

These options can be overriden in the provider config.

Mandatory provider configuration

The following parameters must be present in your configuration file

```
provider:
  type: openstack
  key_name:           # the ssh keypair name to use
  image_name:         # base image to use to boot the machines from
                      # debian8 or ubuntu16.04
  user:               # user to use when connecting to the machines
  network_interface: # network interface of the deployed machines
```

Deployment

Enos will interact with the remote OpenStack APIs. In order to get authenticated you must source your rc file. To use Enos on Openstack there are two distinct cases :

- If you have direct access to all your machines (`gateway: false`), you can launch the deployment with :

```
enos deploy
```

Hint: In this case, prior to the Enos deployment, you have probably started a machine to act as a frontend. This machine is in the same network as those used by Enos

- If you don't have direct access to all your machines (`gateway: true`)

init phase rely on accessing the OpenStack APIs of the over Cloud. If you choose to deploy from your local machine, those APIs are probably unreachable. Check the floating ip assigned to one of your machine and create a proxy socks.

```
# terminal 1
enos up
enos os

# terminal 2
ssh -ND 2100 user@<floating-ip>

# terminal 1
export http_proxy=socks5://127.0.0.1:2100
pip install requests[socks] # install requests support to socks
enos init
```

Note that the proxy socks allows you to use any *openstack* command directly to the over-cloud.

1.2.5 Chameleon Cloud (Bare Metal)

This provider is an OpenStack based provider where some options are set to fit the following platforms :

- https://chi.uc.chameleoncloud.org/
- https://chi.tacc.chameleoncloud.org/

Deployment

You need to install *python-blazarclient* to interact with the lease system of Chameleon :

```
pip install git+https://github.com/openstack/python-blazarclient
```

Configuration

As more default values can be enforced automatically, the following is a valid resources description.

```
provider:
  type: chameleonbaremetal
  key_name: 'enos-key' # must be present prior to the execution

  resources:
    storage: # use "storage" machine type for these roles
      control: 1
      network: 1
    compute: # use "compute" machine type for these roles
      compute: 10
```

Note that on Chameleon, they are two groups of machines : compute and storage.

Default provider configuration

The following options will be set automatically and thus may be omitted in the configuration file.

```
provider:
  type: chameleonbaremetal
  # Name of the Blazar lease to use
  lease_name: enos-lease
  image_name: CC-Ubuntu16.04
  user: cc
  configure_network: False
  network: {name: sharednet1}
  subnet: {name: sharednet1-subnet}
  dns_nameservers: [130.202.101.6, 130.202.101.37]
  # Name of the network interface available on the nodes
  network_interface: eno1
  # Experiment duration
  walltime: "02:00:00"
```

These options can be overriden in the provider config.

On https://chi.tacc.chameleoncloud.org/ the subnet must be `subnet: {'name': 'shared-subnet1'}`

Warning: A shared-network is used and may limit the features of the over-cloud (e.g floating ips)

1.2.6 Chameleon Cloud (KVM)

This provider is an OpenStack based provider where some options are set to fit the folllowing platform :

- https://openstack.tacc.chameleoncloud.org

Configuration

As more default values can be enforced automatically, the following is a valid resources description.

```
provider:
  type: chameleonkvm
  key_name: 'enos-key' # must be present prior to the execution

  resources:
    m1.large:
      control: 1
      network: 1
    m1.medium:
      compute: 10
```

Default provider configuration

The following options will be set automatically and thus may be omitted in the configuration file :

```
provider:
  type: chameleonkvm
  image_name: CC-Ubuntu16.04
  user: cc
  dns_nameservers: [129.114.97.1, 129.114.97.2, 129.116.84.203]
  network_interface: ens3
```

These options can be overriden in the provider config.

1.2.7 Custom Provider

See *Write a new provider*.

1.3 Enos command line

Once installed Enos give you access to its command line. Please refer to the output of `enos -h`. For a specific command you can use `enos <command> -h`

1.4 Benchmarks

Benchmarks are run by Enos by the mean of a workload description. A workload is a set of scenarios grouped by type. A workload is launched with the following command:

```
(venv) $ enos bench --workload=workload
```

enos will look into the `workload` directory for a file named `run.yml`. This file is the description of the workload to launch. One example is given below:

```
rally:
  enabled: true  # default is true
  args:
    concurrency:
      - 1
      - 2
      - 4
    times:
      - 100
  scenarios:
    - name: boot and list servers
      enabled: true # default is true
      file: nova-boot-list-cc.yml
      args:
        sla_max_avg_duration: 30
        times: 50
```

This will launch all the scenarios described under the scenarios keys with all the possible parameters. The parameters are calculated using the cartesian product of the parameters given under the args keys. Locally defined args (scenario level) shadow globally defined args (top level). The same mechanism is applied to the `enabled` values. The scenario must be parameterized accordingly. The key (rally here) defines the type of benchmark to launch: in the future we may support other type of scenarios.

After running the workload, a backup of the environment can be done through `enos backup`.

1.4.1 Rally

Enos supports natively Rally scenarios. Please refer to the Rally documentation for any further information on this benchmarking tool.

Supported keys :

- `name`: the name of the scenario. Can be any string.
- `file`: must be the path to the scenario file. The path is relative to the `workload` directory
- `enabled`: Whether to run this scenario
- `args`: Any parameters that can be understood by the rally scenario
- `plugin`: must be the path to the plugin. The path is relative to the workload directory

1.4.2 Shaker

Enos supports natively Shaker scenarios. Please refer to the Shaker documentation for any further information on this benchmarking tool.

Supported keys :

- `name`: the name of the scenario. Can be any string.
- `file`: must be the alias of the scenario. Enos don't support custom scenario yet.
- `enabled`: Whether to run this scenario

1.4.3 Osprofiler

Supporting OSProfiler in Rally benchmarks is planned for Q3 2017.

1.5 Customizations

1.5.1 Changing Kolla / Ansible variables

Custom Kolla / Ansible parameters can be put in the configuration file under the key `kolla`. The complete list of Kolla variables can be found here.

For instance, Kolla uses the `openstack_release` parameter to fix the OpenStack version to deploy. So, Enos tells Kolla to deploy the `4.0.0` version with:

```
kolla:
  openstack_release: "4.0.0"
```

Note that the Kolla code varies from one version of OpenStack to another. You should always target a version of Kolla code that support the deployment of the expected OpenStack. To do so, you can change the git repository/reference of Kolla code with:

```
kolla_repo: "https://git.openstack.org/openstack/kolla-ansible"
kolla_ref: "stable/ocata"
```

You can also your own local clone of kolla-ansible with:

```
kolla_repo: "file:///path/to/local/kolla-ansible"
```

Note on the network interfaces:

Providers do their best to configure the network decently. This probably doesn't cover all the possible use cases. But, if you know what interfaces are configured by the provider you can specify a more precise allocation under the `kolla` key. For instance:

```
kolla:
  network_interface: eth1
  neutron_external_interface: eth2
  tunnel_interface: eth3
```

Running from kolla/master

if you want to live on the bleeding edge you can run the latest Kolla code with the latest built kolla images.

```
kolla_repo: "https://git.openstack.org/openstack/kolla-ansible"
kolla_ref: "master"

kolla:
  docker_namespace: "beyondtheclouds"
  openstack_release: "latest"
```

1.5.2 Changing the topology

Let's assume you want to run the `nova-conductor` in a dedicated node:

1. Add a new node reservation in the configuration file:

```
paravance:
  control: 1
  network: 1
  compute: 1
  conductor-node: 1
```

2. Create an new inventory file in the `inventories` subdirectory (copy paste the sample inventory) and change the group of the conductor service:

```
[nova-conductor:children]
conductor-node
```

3. In the configuration file, points the inventory to use to this new inventory.

4. Launch the deployment as usual, and you'll get the `nova-conductor` on a dedicated node.

1.5.3 Configuration tuning

At some point, Kolla default parameters won't fit your needs. Kolla provides a mechanism to override custom section of configuration files but isn't applicable in our case (at least in the corresponding branch). So we implement a *quick and dirty* way of patching Kolla code to enable custom configuration files to be used (and by extension custom kolla code). See the possible patch declaration in `ansible/group_vars/all.yml`. Patches should be added in the configuration file of the experiment and you can rely on the `{{ cwd }}` key to link patches in your current working directory.

For instance, adding the following in the configuration file tells enos to look into the current working directory for a file called `mariadb_boostrap.yml` that will replace the kolla-ansible mariadb start playbook.

```
patches:
- name: Patch mariadb start
  src: "{{ cwd }}"/mariadb_bootsrap.yml
  dst: kolla/ansible/roles/mariadb/tasks/start.yml
  enabled: "yes"
```

1.5.4 Ansible configuration

By default, Enos loads its own `ansible.cfg`. To use another Ansible configuration file, the `ANSIBLE_CONFIG` environment variable can be used. Further information can be found : see here.

1.5.5 Docker registry configuration

EnOS can deploy a docker registry in different ways. This is controlled by the configuration file.

No registry

```
registry:
  type: none
```

With the above configuration, EnOS won't deploy any registry. Any docker agent in the deployment will use Docker Hub.

Internal Registry

```
registry:
  type: internal
```

With the above configuration, EnOS deploys a fresh registry that acts as a private docker registry mirroring the official one and cache images close to your deployment resources.

This kind of registry can be made persistent by making sure the underlying storage backend is persitent. Historically, it has been provided on Grid5000 by linking a Ceph Rados Block to the registry backend. Thus you can use the following:

```
registry:
  type: internal
  ceph: true
  ceph_keyring: path to your keyring
  ceph_id: your ceph id
  ceph_rbd: rbd in the form "pool/rbd"
  ceph_mon_host: list of ceph monitor addresses
```

Note: The `reservation.yaml.sample` file provides an example of Ceph configuration that relies on the G5k Ceph of Rennes. The G5k Ceph tutorial will guide you to create your own Rados Block Device.

External Registry

```
registry:
  type: external
  ip: 192.168.142.253
  port: 5000
```

With the above configuration, EnOS will configure all the docker agents to access the registry located at *registry.ip:registry:port*. Note that registry must be an insecure registry.

Note: If you deploy the external registry on the controller node of OpenStack, make sure the port 5000 don't collide with the port of Keystone.

When using EnOS locally, it's a good idea to keep a separated external registry to speed up the deployment.

1.6 Network Emulation

1.6.1 Links description

Enos allows to enforce network emulation in terms of latency and bandwidth limitations.

Network constraints (latency/bandwidth limitations) are enabled by the use of groups of nodes. Resources *must* be described using a `topology` description instead of a `resources` description. The following example will define 4 groups named `grp1`, `grp2`, `grp3` and `grp4` respectively:

```
topology:
  grp1:
    paravance:
      control: 1
      network: 1
  grp[2-4]:
    paravance:
      compute: 1
```

Constraints are then described under the `network_constraints` key in the configuration file:

```
network_constraints:
  enable: true
  default_delay: 25ms
  default_rate: 100mbit
  default_loss: 0.1%
  constraints:
    - src: grp1
      dst: grp[2-4]
      delay: 10ms
      rate: 1gbit
      loss: 0%
      symetric: true
```

To enforce the constraints, you can invoke:

```
enos tc
```

Note that The machines must be available, thus the *up* phase must have been called before.

As a result

- the network delay between every machines of `grp1` and the machines of the other groups will be 20ms (2x10ms: symetric)
- the bandwidth between every machines of `grp1` and the machines of the other groups will be 1 Gbit/s
- the packet loss percentage between every machines of `grp1` and the machines of the other groups will be 0%.
- the network delay between every machines of `grp2` and `grp3` (resp. `grp2` and `grp4`) (resp. `grp3` and `grp4`) will be 50ms
- the bandwidth between every machines of `grp2` and `grp3` (resp. `grp2` and `grp4`) (resp. `grp3` and `grp4`) will be 100Mbit/s
- the packet loss percentage between every machines of `grp2` and `grp3` (resp. `grp2` and `grp4`) (resp. `grp3` and `grp4`) will be 0.1%

1.6.2 Checking the constraints

Invoking

```
enos tc --test
```

will generate various reports to validate the constraints. They are based on `fping` and `flent` latency and bandwidth measurements respectively. The reports will be located in the result directory.

1.6.3 Notes

- `default_delay`, `default_rate`, `default_loss` are mandatory
- To disable the network constraints you can specify `enable: false` under the `network_constraints` key and launch again `enos tc`
- To exclude a group from any tc rule, you can add an optionnal `except` key to the `network_constraints`:

```
network_constraints:
  enable: true
  default_delay: 25ms
  default_rate: 100mbit
  default_loss: 0%
  constraints:
    - src: grp[1-3]
      dst: grp[4-6]
      delay: 10ms
      rate: 1gbit
      symetric: true
  except:
    - grp1
```

1.7 Analysis

1.7.1 Real-time

Setting `enable_monitoring: true` in the configuration file will deploy a monitoring stack composed of:

- Cadvisor and Collectd agents
- InfluxDB for metrics collection
- Graphana for the visualization / exploration

All these services are accessible on their default ports. For instance you'll be able to access grafana dashboards on port `3000` of the node hosting grafana.

Some dashboards are available here.

1.7.2 Post-mortem

TODO

1.7.3 Annotations

Enos embeds an Ansible plugin to add annotations in Grafana. These annotations are marked points which provide rich information about events when hovered over. Enos uses the `ansible.cfg` file that loads the plugin. The plugin can be disabled by editing the line `callback_whitelist = influxdb_events` in the `ansible.cfg`. Note also that the plugin is automaticaly disabled when the monitoring tools are not deployed (i.e. when *enable_monitoring = false* is set in the configuration file).

Once the deployment is finished, a compatible dashboard must be used in Grafana to display annotations. An example of such dashboard is available here.

1.8 Contribute

All contributions are welcome on BeyondTheClouds/enos. For any questions, feature requests, issues please use the GitHub issue tracker.

1.8.1 Install from sources and make them editable

```
$ git clone https://github.com/BeyondTheClouds/enos.git
$ cd enos
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install -e .
```

1.8.2 Get tox

```
(venv) $ pip install tox
```

1.8.3 Running the tests

```
(venv) $ tox
```

1.8.4 Running syntax checker

```
(venv) $ tox -e pep8
```

1.8.5 Generate the documentation

```
(venv) $ tox -e docs
```

1.8.6 Other Topics

Write a new provider

The actual implementation gives providers for *Static* resources, *Vagrant*, *Grid'5000* and *Openstack* itself. If you want to support another testbed, then implementing a new provider is easy as 500 lines of Python code.

The new provider should follow the provider.py interface which consists in three methods: `init`, `destroy` and `default_config`. Another good starting point is the simple static implementation.

Init Method

The `init` method provides resources and provisions the environment. To let the provider knows what kind and how many resources should be provided, the method is fed with the `config` object that maps the reservations file. So a provider can access the resource description with:

```
rsc = config['resources']

# Use rsc to book resources ...
```

At the end of the `init`, the provider should return a list of host.py that Enos can SSH on, together with a pool of available IP for OpenStack Network.

Destroy Method

The `destoy` method destroys resources that have been used for the deployment. The provider can rely on the environment variable to get information related to its deployment.

Default Provider Configuration Methods

The `default_config` specifies keys used to configure the provider with a `dict`. A key could be *optional* and so should be provided with a default value, or *required* and so should be set to `None`. The user then can override these keys in the reservation file, under the `provider` key. Keys marked as `None` in the `default_config` are automatically tested for overriding in the reservation file.

Provider Instantiation

Enos automatically instantiates a provider based on the name specified in the `reservation.yaml`. For instance, based on the following reservation file,

```
provider: "my-provider"
```

Enos seeks for a file called `my-provider.py` into `enos/provider` and instantiates its main class. But sometimes, the provider requires extra information for its initialisation. The good place for this information is to put it under the provider key. In this case, the provider name should be accessible throughout the `type` key:

```
provider:
  type: "my-provider"
  extra-var: ...
  ...
```

Then the provider can access `extra-var` with `config['provider']['extra-var']`. Supported extra information is documented into the provider documentation.

CHAPTER 2

Why Enos ?

https://en.wikipedia.org/wiki/Enos_(chimpanzee)

CHAPTER 3

License

CHAPTER 4

Indices and tables

- genindex
- modindex
- search