Murano

Release 2015.1.1.dev32

November 21, 2015

Contents

1 Introduction **1**

1.1 Murano Installation Guide . . . 1
1.2 Prepare A Lab For Murano . . . 14
1.3 Installing and Running the Development Version . . . 17
1.4 Installing and Running Manually . . . 17
1.5 MuranoPL: Murano Programming Language . . . 26
1.6 Dynamic UI Definition specification . . . 46
1.7 Murano workflow . . . 50
1.8 Murano Policy Enforcement . . . 52
1.9 Composing application package manual . . . 58
1.10 Uploading HOT templates to the Application Catalog . . . 62
1.11 Building Murano Image . . . 64
1.12 Murano Automated Tests Description . . . 73
1.13 Murano client . . . 75
1.14 Murano package repository . . . 78
1.15 Contributing to Murano . . . 79
1.16 Development Guidelines . . . 80
1.17 Murano TroubleShooting and Debug Tips . . . 81
1.18 Migrating applications from Murano v0.5 to Stable/Juno . . . 82
1.19 Murano API v1 specification . . . 83

2 Indices and tables **109**

CHAPTER 1

Introduction

Murano Project introduces an application catalog, which allows application developers and cloud administrators to publish various cloud-ready applications in a browsable categorised catalog. It may be used by the cloud users (including the unexperienced ones) to pick-up the needed applications and services and composes the reliable environments out of them in a "push-the-button" manner.

Key goal is to provide UI and API which allows to compose and deploy composite environments on the Application abstraction level and then manage their lifecycle.

Murano consists of several source code repositories:

- murano - is the main repository. It contains code for Murano API server, Murano engine and MuranoPL
- murano-agent - agent which runs on guest VMs and executes deployment plan
- murano-dashboard - Murano UI implemented as a plugin for OpenStack Dashboard
- python-muranoclient - Client library and CLI client for Murano

This documentation offers information on how Murano works and how to contribute to the project.

Installation

1.1 Murano Installation Guide

1.1.1 Content

Prepare A Lab For Murano

This section provides basic information about lab's system requirements. It also contains a description of a test which you may use to check if your hardware fits the requirements. To do this, run the test and compare the results with baseline data provided.

System prerequisites

Supported Operation Systems

- Ubuntu Server 12.04 LTS
- RHEL/CentOS 6.4

System packages are required for Murano

Ubuntu

- gcc
- python-pip
- python-dev
- libxml2-dev
- libxslt-dev
- libffi-dev
- libmysqlclient-dev
- libpq-dev
- python-openssl
- mysql-client
- python-mysqldb

CentOS

- gcc
- python-pip
- python-devel
- libxml2-devel
- libxslt-devel
- libffi-devel
- postgresql-devel
- pyOpenSSL
- mysql
- MySQL-python

Lab Requirements

Criteria	Minimal	Recommended
CPU	4 core @ 2.4 GHz	24 core @ 2.67 GHz
RAM	8 GB	24 GB or more
HDD	2 x 500 GB (7200 rpm)	4 x 500 GB (7200 rpm
RAID	Software RAID-1 (use mdadm as it will improve read performance almost two times)	Hardware RAID-10

Table: Hardware requirements

There are a few possible storage configurations except the shown above. All of them were tested and were working well.

- 1x SSD 500+ GB

- **1x HDD (7200 rpm) 500+ GB and 1x SSD 250+ GB (install the system onto** the HDD and mount the SSD drive to folder where VM images are)
- 1x HDD (15000 rpm) 500+ GB

Test Your Lab Host Performance

We have measured time required to boot 1 to 5 instances of Windows system simultaneously. You can use this data as the baseline to check if your system is fast enough.

You should use sysprepped images for this test, to simulate VM first boot.

Steps to reproduce test:

1. Prepare Windows 2012 Standard (with GUI) image in QCOW2 format. Let's assume that its name is ws-2012-std.qcow2
2. Ensure that there is NO KVM PROCESSES on the host. To do this, run command:

```
>#ps aux | grep kvm
```

3. Make 5 copies of Windows image file:

```
>#for i in $(seq 5); do \
cp ws-2012-std.qcow2 ws-2012-std-$i.qcow2; done
```

4. Create script start-vm.sh in the folder with .qcow2 files:

```
#!/bin/bash
[ -z $1 ] || echo "VM count not provided!"; exit 1
for i in $(seq $1); do
echo "Starting VM $i ..."
kvm -m 1024 -drive file=ws-2012-std-$i.qcow2,if=virtio -net user -net nic,model=virtio -nographi
```

5. Start ONE instance with command below (as root) and measure time between VM's launch and the moment when Server Manager window appears. To view VM's desktop, connect with VNC viewer to your host to VNC screen :1 (port 5901):

```
>#./start-vm.sh 1
```

6. Turn VM off. You may simply kill all KVM processes by

```
>#killall kvm
```

7. Start FIVE instances with command below (as root) and measure time interval between ALL VM's launch and the moment when LAST Server Manager window appears. To view VM's desktops, connect with VNC viewer to your host to VNC screens :1 thru :5 (ports 5901-5905):

```
>#./start-vm.sh 5
```

8. Turn VMs off. You may simply kill all KVM processes by

```
>#killall kvm
```

Baseline Data

The table below provides baseline data which we've got in our environment.

Avg. Time refers to the lab with recommended hardware configuration, while **Max. Time** refers to minimal hardware configuration.

	Boot ONE instance	Boot FIVE instances
Avg. Time	3m:40s	8m
Max. Time	5m	20m

Host Optimizations

Default KVM installation could be improved to provide better performance.

The following optimizations may improve host performance up to 30%:

- change default scheduler from **CFQ** to **Deadline**
- use **ksm**
- use **vhost-net**

Installing and Running the Development Version

The contrib/devstack directory contains the files necessary to integrate Murano with Devstack.

1. Follow Devstack documentation to setup a host for Devstack. Then clone Devstack source code.
2. Copy Murano integration scripts to Devstack either by setting environment variable or providing complete path to devstack directory. Below one is using environment variable:

```
$ export DEVSTACK_DIR=<complete path to devstack directory(clonned)>
$ cp lib/murano ${DEVSTACK_DIR}/lib
$ cp lib/murano-dashboard ${DEVSTACK_DIR}/lib
$ cp extras.d/70-murano.sh ${DEVSTACK_DIR}/extras.d
```

3. Create a `localrc` file as input to devstack.
4. The Murano, Neutron and Heat services are not enabled by default, so they must be enabled in `localrc` before running `stack.sh`. This example `localrc` file shows all of the settings required for Murano:

```
# Enable Neutron
ENABLED_SERVICES+=,q-svc,q-agt,q-dhcp,q-l3,q-meta,neutron

# Enable Heat
enable_service heat h-api h-api-cfn h-api-cw h-eng

# Enable Murano
enable_service murano murano-api murano-engine
```

5. Deploy your OpenStack Cloud with Murano:

```
$ ./stack.sh
```

Installing and Running Manually

Prepare Environment

Install Prerequisites First you need to install a number of packages with your OS package manager. The list of packages depends on the OS you use.

Ubuntu

```
$ sudo apt-get install python-pip python-dev \
> libmysqlclient-dev libpq-dev \
> libxml2-dev libxslt1-dev \
> libffi-dev
```

Fedora

Note: Fedora support wasn't thoroughly tested. We do not guarantee that Murano will work on Fedora.

```
$ sudo yum install gcc python-setuptools python-devel python-pip
```

CentOS

```
$ sudo yum install gcc python-setuptools python-devel
$ sudo easy_install pip
```

Install tox

```
$ sudo pip install tox
```

Install And Configure Database Murano can use various database types on backend. For development purposes SQLite is enough in most cases. For production installations you should use MySQL or PostgreSQL databases.

> **Warning:** Although Murano could use PostgreSQL database on backend, it wasn't thoroughly tested and should be used with caution.

To use MySQL database you should install it and create an empty database first:

```
$ apt-get install python-mysqldb mysql-server

$ mysql -u root -p
mysql> CREATE DATABASE murano;
mysql> GRANT ALL PRIVILEGES ON murano.* TO 'murano'@'localhost' \
    IDENTIFIED BY 'MURANO_DBPASS';
mysql> exit;
```

Install the API service and Engine

1. Create a folder which will hold all Murano components.

   ```
   $ mkdir ~/murano
   ```

2. Clone the Murano git repository to the management server.

   ```
   $ cd ~/murano
   $ git clone git://git.openstack.org/openstack/murano
   ```

3. Set up Murano config file

   Murano has common config file for API and Engine servicies.

   First, generate sample configuration file, using tox

```
$ cd ~/murano/murano
$ tox -e genconfig
```

And make a copy of it for further modifications

```
$ cd ~/murano/murano/etc/murano
$ ln -s murano.conf.sample murano.conf
```

4. Edit `murano.conf` with your favorite editor. Below is an example which contains basic settings your are likely need to configure.

Note: The example below uses SQLite database. Edit **[database]** section if you want to use other database type.

```
[DEFAULT]
debug = true
verbose = true
rabbit_host = %RABBITMQ_SERVER_IP%
rabbit_userid = %RABBITMQ_USER%
rabbit_password = %RABBITMQ_PASSWORD%
rabbit_virtual_host = %RABBITMQ_SERVER_VIRTUAL_HOST%
notification_driver = messagingv2

...

[database]
backend = sqlalchemy
connection = sqlite:///murano.sqlite

...

[keystone]
auth_url = 'http://%OPENSTACK_HOST_IP%:5000/v2.0'

...

[keystone_authtoken]
auth_uri = 'http://%OPENSTACK_HOST_IP%:5000/v2.0'
auth_host = '%OPENSTACK_HOST_IP%'
auth_port = 5000
auth_protocol = http
admin_tenant_name = %OPENSTACK_ADMIN_TENANT%
admin_user = %OPENSTACK_ADMIN_USER%
admin_password = %OPENSTACK_ADMIN_PASSWORD%

...

[murano]
url = http://%YOUR_HOST_IP%:8082

[rabbitmq]
host = %RABBITMQ_SERVER_IP%
login = %RABBITMQ_USER%
password = %RABBITMQ_PASSWORD%
virtual_host = %RABBITMQ_SERVER_VIRTUAL_HOST%
```

5. Create a virtual environment and install Murano prerequisites. We will use *tox* for that. Virtual environment will be created under *.tox* directory.

```
$ cd ~/murano/murano
$ tox
```

6. Create database tables for Murano.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-db-manage \
> --config-file ./etc/murano/murano.conf upgrade
```

7. Open a new console and launch Murano API. A separate terminal is required because the console will be locked by a running process.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-api \
> --config-file ./etc/murano/murano.conf
```

8. Import Core Murano Library.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-manage \
> --config-file ./etc/murano/murano.conf \
> import-package ./meta/io.murano
```

9. **Open a new console and launch Murano Engine. A separate terminal is** required because the console will be locked by a running process.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-engine --config-file ./etc/murano/murano.conf
```

Install Murano Dashboard

Murano API & Engine services provide the core of Murano. However, your need a control plane to use it. This section decribes how to install and run Murano Dashboard.

1. Clone the repository with Murano Dashboard.

```
$ cd ~/murano
$ git clone git://git.openstack.org/openstack/murano-dashboard
```

2. Clone horizon repository

```
$ git clone git://git.openstack.org/openstack/horizon
```

3. Create venv and install muranodashboard as editable module.

```
$ cd horizon
$ tox -e venv -- pip install -e ../murano-dashboard
```

4. Copy muranodashboard plugin file.

   This step enables murano panel in horizon dashboard.

```
$ cp ../murano-dashboard/muranodashboard/local/_50_murano.py openstack_dashboard/local/enabled/
```

5. Prepare local settings.

   To get more information, check out official horizon documentation.

```
$ cp openstack_dashboard/local/local_settings.py.example openstack_dashboard/local/local_setting
```

6. Customize local settings according to Openstack installation.

```
...
ALLOWED_HOSTS = '*'

# Provide OpenStack Lab credentials
OPENSTACK_HOST = '%OPENSTACK_HOST_IP%'

...

# Set secret key to prevent it's generation
SECRET_KEY = 'random_string'

...

DEBUG_PROPAGATE_EXCEPTIONS = DEBUG
```

Also, it's better to change default session backend from browser cookies to database to avoid issues with forms during creating applications:

```
...
DATABASES = {
    'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': '/tmp/murano-dashboard.sqlite',
    }
}

SESSION_ENGINE = 'django.contrib.sessions.backends.db'
```

If you do not plan to get murano service from keystone application catalog, provide where murano-api service is running:

```
...
MURANO_API_URL = 'http://localhost:8082'
```

7. Perform database synchronization.

Optional step. Needed in case you set up database as a session backend.

```
$ tox -e venv -- python manage.py syncdb
```

You can reply 'no' since for development purpose separate user is not needed.

8. Run Django server at 127.0.0.1:8000 or provide different IP and PORT parameters.

```
$ tox -e venv -- python manage.py runserver <IP:PORT>
```

Development server will be restarted automatically on every code change.

9. Open dashboard using url http://localhost:8000

Import Murano Applications

Murano provides excellent catalog services, but it also requires applications which to provide. This section describes how to import Murano Applications from Murano App Incubator.

1. Clone Murano App Incubator repository.

```
$ cd ~/murano
$ git clone git://git.openstack.org/openstack/murano-apps
```

2. Import every package you need from this repository, using the command below.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-manage \
> --config-file ./etc/murano/murano.conf \
> import-package ../murano-app-incubator/%APPLICATION_DIRECTORY_NAME%
```

Network Configuration Murano may work in various networking environments and is capable to detect the current network configuration and choose the appropriate settings automatically. However, some additional actions are required to support advanced scenarios.

Nova network support Nova Network is simplest networking solution, which has limited capabilities but is available on any OpenStack deployment without the need to deploy any additional components.

When a new Murano Environment is created, Murano checks if a dedicated networking service (i.e. Neutron) exists in the current OpenStack deployment. It relies on Keystone's service catalog for that. If such a service is not present, Murano automatically falls back to Nova Network. No further configuration is needed in this case, all the VMs spawned by Murano will be joining the same Network.

Neutron support If Neutron is installed, Murano enables its advanced networking features that give you ability to not care about configuring networks for your application.

By default it will create an isolated network for each environment and join all VMs needed by your application to that network. To install and configure application in just spawned virtual machine Murano also requires a router connected to the external network.

Automatic Neutron network configuration To create router automatically, provide the following parameters in config file:

```
[networking]

external_network = %EXTERNAL_NETWORK_NAME%
router_name = %MURANO_ROUTER_NAME%
create_router = true
```

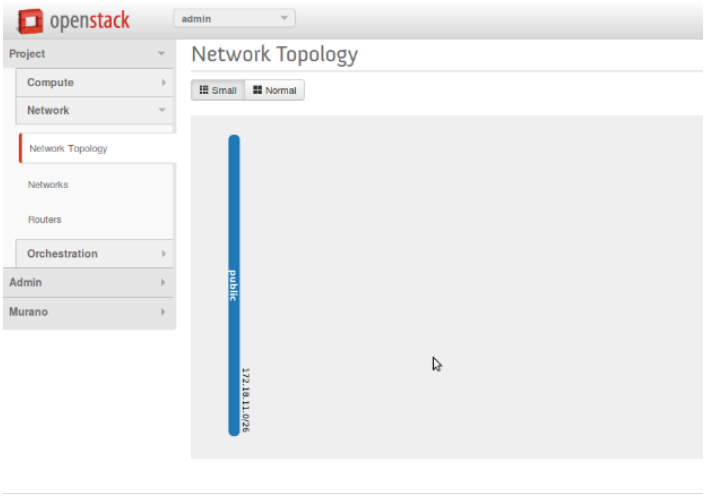
To figure out the name of the external network, perform the following command:

```
$ neutron net-external-list
```

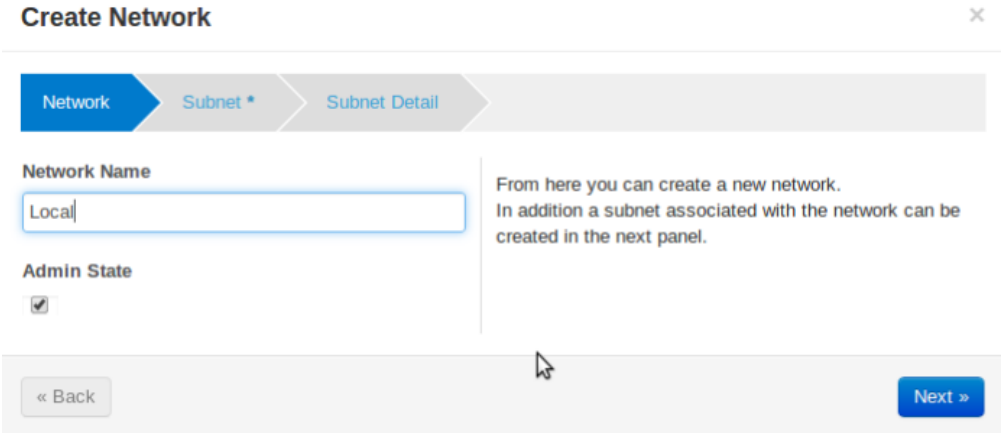
During the first deploy, required networks and router with specified name will be created and set up.

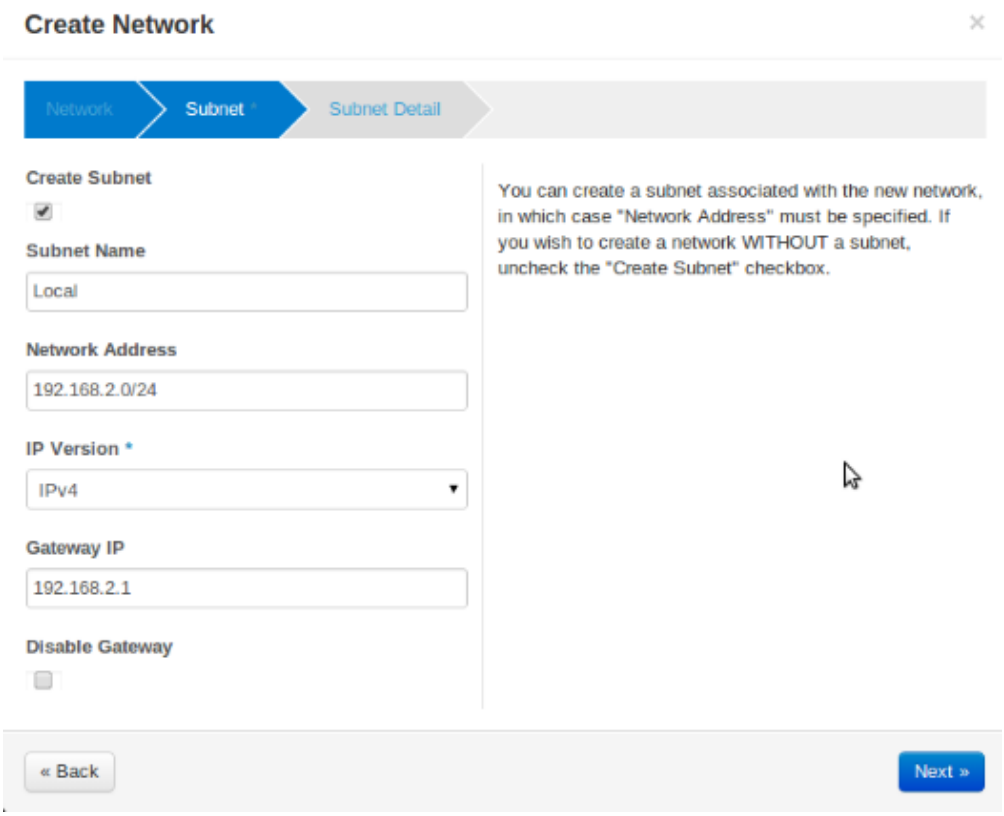
Manual neutron network configuration

- Step 1. Create public network
  - First, you need to check for existence of external networks. Login as admin and go to *Project -> Network -> Network Topology*. And check network type in network details at *Admin -> Networks -> Network name* page. The same action can be done via CLI by running *neutron net-external-list*. To create new external network examine OpenStack documentation.

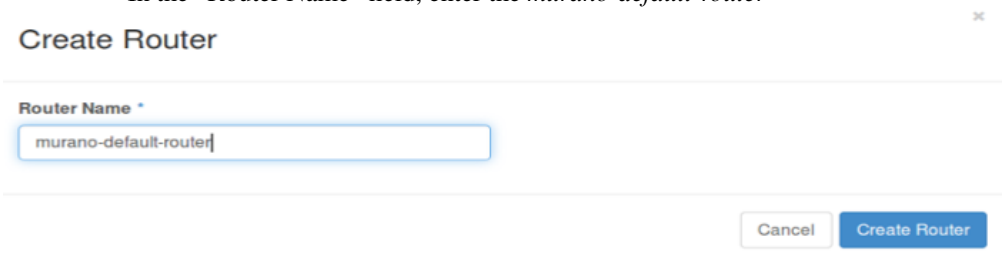


- Step 2. Create local network
  - Go to *Project -> Network -> Networks*.
  - Click *Create Network* and fill the form.





- Step 3. Create router
  - Go to *Project -> Network -> Routers*
  - Click "Create Router"
  - In the "Router Name" field, enter the *murano-default-router*

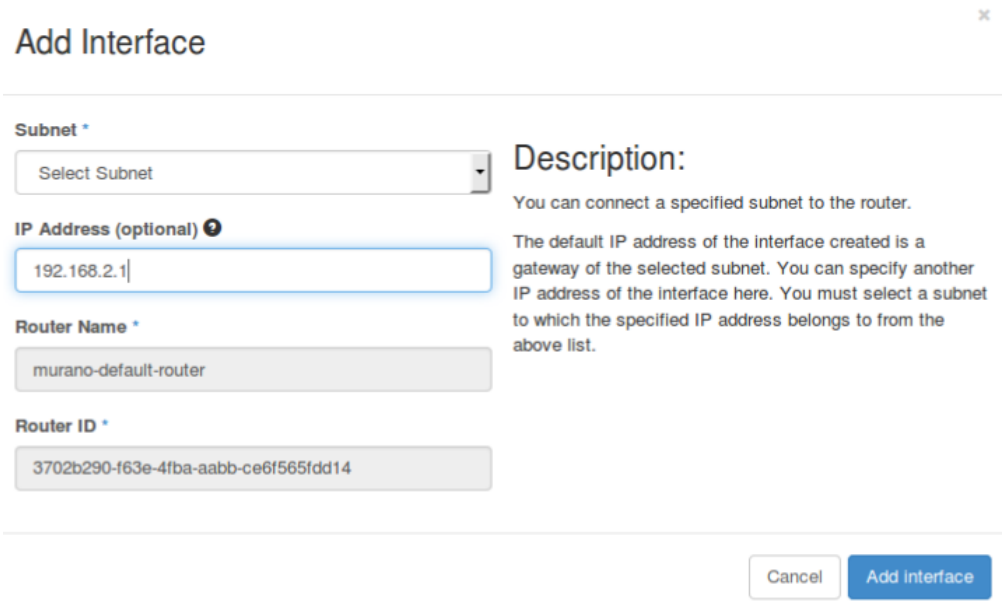


If you specify a name other than *murano-default-router*, it will be necessary to change the following settings in the config file:

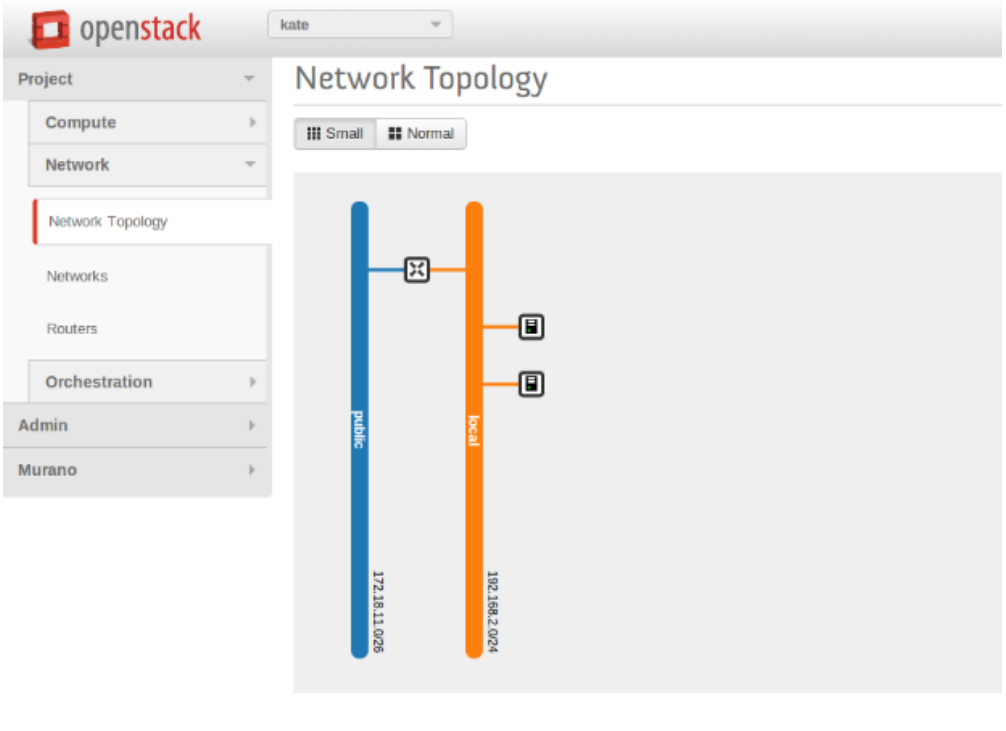
```
[networking]

router_name = %SPECIFIED_NAME%
create_router = false
```

- Click on the specified router name
- In the opened view click "Add interface"
- Specify the subnet and IP address



And check the result in *Network Topology* tab.



SSL configuration

Murano components are able to work with SSL. This chapter will help your to make proper settings with SSL configuration.

HTTPS for Murano API

SSL for Murano API service can be configured in *ssl* section in `/etc/murano/murano.conf`. Just point to a valid SSL certificate. See the example below:

```
[ssl]
cert_file = PATH
key_file = PATH
ca_file = PATH
```

- *cert_file* Path to the certificate file the server should use when binding to an SSL-wrapped socket.
- *key_file* Path to the private key file the server should use when binding to an SSL-wrapped socket.
- *ca_file* Path to the CA certificate file the server should use to validate client certificates provided during an SSL handshake. This is ignored if cert_file and "key_file" are not set.

The use of SSL is automatically started after point to HTTPS protocol instead of HTTP during registration Murano API service in endpoints (Change publicurl argument to start with https://). SSL for Murano API is implemented like in any other Openstack component. This realization is based on ssl python module so more information about it can be found here.

SSL for RabbitMQ

All Murano components communicate with each other by RabbitMQ. This interaction can be encrypted with SSL. By default all messages in Rabbit MQ are not encrypted. Each RabbitMQ Exchange should be configured separately.

Murano API <-> Rabbit MQ exchange <-> Murano Engine

Edit ssl parameters in default section of `/etc/murano/murano.conf`. Set `rabbit_use_ssl` option to *true* and configure ssl kombu parameters. Specify the path to the SSL keyfile and SSL CA certificate in a regular format: /path/to/file without quotes or leave it empty to allow self-signed certificates.

```
# connect over SSL for RabbitMQ (boolean value)
#rabbit_use_ssl=false

# SSL version to use (valid only if SSL enabled). valid values
# are TLSv1, SSLv23 and SSLv3. SSLv2 may be available on some
# distributions (string value)
#kombu_ssl_version=

# SSL key file (valid only if SSL enabled) (string value)
#kombu_ssl_keyfile=

# SSL cert file (valid only if SSL enabled) (string value)
#kombu_ssl_certfile=

# SSL certification authority file (valid only if SSL enabled)
# (string value)
#kombu_ssl_ca_certs=
```

Murano Agent -> Rabbit MQ exchange

In main murano configuration file there is a section ,named *rabbitmq*, that is responsible for set up communication between Murano Agent and Rabbit MQ. Just set *ssl* parameter to True to enable ssl.

```
[rabbitmq]
host = localhost
port = 5672
```

```
login = guest
password = guest
virtual_host = /
ssl = True
```

If you want to configure Murano Agent in a different way change the default template. It can be found in Murano Core Library, located at *http://git.openstack.org/cgit/openstack/murano/tree/meta/io.murano/Resources/Agent-v1.template*. Take a look at appSettings section:

```
<appSettings>
        <add key="rabbitmq.host" value="%RABBITMQ_HOST%"/>
        <add key="rabbitmq.port" value="%RABBITMQ_PORT%"/>
        <add key="rabbitmq.user" value="%RABBITMQ_USER%"/>
        <add key="rabbitmq.password" value="%RABBITMQ_PASSWORD%"/>
        <add key="rabbitmq.vhost" value="%RABBITMQ_VHOST%"/>
        <add key="rabbitmq.inputQueue" value="%RABBITMQ_INPUT_QUEUE%"/>
        <add key="rabbitmq.resultExchange" value=""/>
        <add key="rabbitmq.resultRoutingKey" value="%RESULT_QUEUE%"/>
        <add key="rabbitmq.durableMessages" value="true"/>

        <add key="rabbitmq.ssl" value="%RABBITMQ_SSL%"/>
        <add key="rabbitmq.allowInvalidCA" value="true"/>
        <add key="rabbitmq.sslServerName" value=""/>

    </appSettings>
```

Desired parameter should be set directly to the value of the key that you want to change. Quotes are need to be kept. Thus you can change "rabbitmq.ssl" and "rabbitmq.port" values to make Rabbit MQ work with this exchange in a different from Murano-Engine way. After modification, don't forget to zip and re-upload core library.

SSL for Murano Dashboard

If you are going not to use self-signed certificates additional configuration do not need to be done. Just point https in the URL. Otherwise, set *MURANO_API_INSECURE = True* on horizon config. You can find it in `/etc/openstack-dashboard/local_settings.py`..

1.2 Prepare A Lab For Murano

This section provides basic information about lab's system requirements. It also contains a description of a test which you may use to check if your hardware fits the requirements. To do this, run the test and compare the results with baseline data provided.

1.2.1 System prerequisites

Supported Operation Systems

- Ubuntu Server 12.04 LTS
- RHEL/CentOS 6.4

System packages are required for Murano

Ubuntu

- gcc

- python-pip
- python-dev
- libxml2-dev
- libxslt-dev
- libffi-dev
- libmysqlclient-dev
- libpq-dev
- python-openssl
- mysql-client
- python-mysqldb

CentOS

- gcc
- python-pip
- python-devel
- libxml2-devel
- libxslt-devel
- libffi-devel
- postgresql-devel
- pyOpenSSL
- mysql
- MySQL-python

1.2.2 Lab Requirements

Criteria	Minimal	Recommended
CPU	4 core @ 2.4 GHz	24 core @ 2.67 GHz
RAM	8 GB	24 GB or more
HDD	2 x 500 GB (7200 rpm)	4 x 500 GB (7200 rpm
RAID	Software RAID-1 (use mdadm as it will improve read performance almost two times)	Hardware RAID-10

Table: Hardware requirements

There are a few possible storage configurations except the shown above. All of them were tested and were working well.

- 1x SSD 500+ GB
- **1x HDD (7200 rpm) 500+ GB and 1x SSD 250+ GB (install the system onto** the HDD and mount the SSD drive to folder where VM images are)
- 1x HDD (15000 rpm) 500+ GB

1.2.3 Test Your Lab Host Performance

We have measured time required to boot 1 to 5 instances of Windows system simultaneously. You can use this data as the baseline to check if your system is fast enough.

You should use sysprepped images for this test, to simulate VM first boot.

Steps to reproduce test:

1. Prepare Windows 2012 Standard (with GUI) image in QCOW2 format. Let's assume that its name is ws-2012-std.qcow2
2. Ensure that there is NO KVM PROCESSES on the host. To do this, run command:

```
>#ps aux | grep kvm
```

3. Make 5 copies of Windows image file:

```
>#for i in $(seq 5); do \
cp ws-2012-std.qcow2 ws-2012-std-$i.qcow2; done
```

4. Create script start-vm.sh in the folder with .qcow2 files:

```
#!/bin/bash
[ -z $1 ] || echo "VM count not provided!"; exit 1
for i in $(seq $1); do
echo "Starting VM $i ..."
kvm -m 1024 -drive file=ws-2012-std-$i.qcow2,if=virtio -net user -net nic,model=virtio -nographi
```

5. Start ONE instance with command below (as root) and measure time between VM's launch and the moment when Server Manager window appears. To view VM's desktop, connect with VNC viewer to your host to VNC screen :1 (port 5901):

```
>#./start-vm.sh 1
```

6. Turn VM off. You may simply kill all KVM processes by

```
>#killall kvm
```

7. Start FIVE instances with command below (as root) and measure time interval between ALL VM's launch and the moment when LAST Server Manager window appears. To view VM's desktops, connect with VNC viewer to your host to VNC screens :1 thru :5 (ports 5901-5905):

```
>#./start-vm.sh 5
```

8. Turn VMs off. You may simply kill all KVM processes by

```
>#killall kvm
```

1.2.4 Baseline Data

The table below provides baseline data which we've got in our environment.

Avg. Time refers to the lab with recommended hardware configuration, while **Max. Time** refers to minimal hardware configuration.

	Boot ONE instance	Boot FIVE instances
Avg. Time	3m:40s	8m
Max. Time	5m	20m

1.2.5 Host Optimizations

Default KVM installation could be improved to provide better performance.

The following optimizations may improve host performance up to 30%:

- change default scheduler from **CFQ** to **Deadline**
- use **ksm**
- use **vhost-net**

1.3 Installing and Running the Development Version

The contrib/devstack directory contains the files necessary to integrate Murano with Devstack.

1. Follow Devstack documentation to setup a host for Devstack. Then clone Devstack source code.
2. Copy Murano integration scripts to Devstack either by setting environment variable or providing complete path to devstack directory. Below one is using environment variable:

```
$ export DEVSTACK_DIR=<complete path to devstack directory(clonned)>
$ cp lib/murano ${DEVSTACK_DIR}/lib
$ cp lib/murano-dashboard ${DEVSTACK_DIR}/lib
$ cp extras.d/70-murano.sh ${DEVSTACK_DIR}/extras.d
```

3. Create a `localrc` file as input to devstack.
4. The Murano, Neutron and Heat services are not enabled by default, so they must be enabled in `localrc` before running `stack.sh`. This example `localrc` file shows all of the settings required for Murano:

```
# Enable Neutron
ENABLED_SERVICES+=,q-svc,q-agt,q-dhcp,q-l3,q-meta,neutron

# Enable Heat
enable_service heat h-api h-api-cfn h-api-cw h-eng

# Enable Murano
enable_service murano murano-api murano-engine
```

5. Deploy your OpenStack Cloud with Murano:

```
$ ./stack.sh
```

1.4 Installing and Running Manually

1.4.1 Prepare Environment

Install Prerequisites

First you need to install a number of packages with your OS package manager. The list of packages depends on the OS you use.

Ubuntu

```
$ sudo apt-get install python-pip python-dev \
> libmysqlclient-dev libpq-dev \
> libxml2-dev libxslt1-dev \
> libffi-dev
```

Fedora

Note: Fedora support wasn't thoroughly tested. We do not guarantee that Murano will work on Fedora.

```
$ sudo yum install gcc python-setuptools python-devel python-pip
```

CentOS

```
$ sudo yum install gcc python-setuptools python-devel
$ sudo easy_install pip
```

Install tox

```
$ sudo pip install tox
```

Install And Configure Database

Murano can use various database types on backend. For development purposes SQLite is enough in most cases. For production installations you should use MySQL or PostgreSQL databases.

Warning: Although Murano could use PostgreSQL database on backend, it wasn't thoroughly tested and should be used with caution.

To use MySQL database you should install it and create an empty database first:

```
$ apt-get install python-mysqldb mysql-server
```

```
$ mysql -u root -p
mysql> CREATE DATABASE murano;
mysql> GRANT ALL PRIVILEGES ON murano.* TO 'murano'@'localhost' \
    IDENTIFIED BY 'MURANO_DBPASS';
mysql> exit;
```

1.4.2 Install the API service and Engine

1. Create a folder which will hold all Murano components.

   ```
   $ mkdir ~/murano
   ```

2. Clone the Murano git repository to the management server.

```
$ cd ~/murano
$ git clone git://git.openstack.org/openstack/murano
```

3. Set up Murano config file

   Murano has common config file for API and Engine servicies.

   First, generate sample configuration file, using tox

   ```
   $ cd ~/murano/murano
   $ tox -e genconfig
   ```

   And make a copy of it for further modifications

   ```
   $ cd ~/murano/murano/etc/murano
   $ ln -s murano.conf.sample murano.conf
   ```

4. Edit `murano.conf` with your favorite editor. Below is an example which contains basic settings your are likely need to configure.

   Note: The example below uses SQLite database. Edit **[database]** section if you want to use other database type.

   ```
   [DEFAULT]
   debug = true
   verbose = true
   rabbit_host = %RABBITMQ_SERVER_IP%
   rabbit_userid = %RABBITMQ_USER%
   rabbit_password = %RABBITMQ_PASSWORD%
   rabbit_virtual_host = %RABBITMQ_SERVER_VIRTUAL_HOST%
   notification_driver = messagingv2

   ...

   [database]
   backend = sqlalchemy
   connection = sqlite:///murano.sqlite

   ...

   [keystone]
   auth_url = 'http://%OPENSTACK_HOST_IP%:5000/v2.0'

   ...

   [keystone_authtoken]
   auth_uri = 'http://%OPENSTACK_HOST_IP%:5000/v2.0'
   auth_host = '%OPENSTACK_HOST_IP%'
   auth_port = 5000
   auth_protocol = http
   admin_tenant_name = %OPENSTACK_ADMIN_TENANT%
   admin_user = %OPENSTACK_ADMIN_USER%
   admin_password = %OPENSTACK_ADMIN_PASSWORD%

   ...

   [murano]
   url = http://%YOUR_HOST_IP%:8082
   ```

```
[rabbitmq]
host = %RABBITMQ_SERVER_IP%
login = %RABBITMQ_USER%
password = %RABBITMQ_PASSWORD%
virtual_host = %RABBITMQ_SERVER_VIRTUAL_HOST%
```

5. Create a virtual environment and install Murano prerequisites. We will use *tox* for that. Virtual environment will be created under *.tox* directory.

```
$ cd ~/murano/murano
$ tox
```

6. Create database tables for Murano.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-db-manage \
> --config-file ./etc/murano/murano.conf upgrade
```

7. Open a new console and launch Murano API. A separate terminal is required because the console will be locked by a running process.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-api \
> --config-file ./etc/murano/murano.conf
```

8. Import Core Murano Library.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-manage \
> --config-file ./etc/murano/murano.conf \
> import-package ./meta/io.murano
```

9. **Open a new console and launch Murano Engine. A separate terminal is** required because the console will be locked by a running process.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-engine --config-file ./etc/murano/murano.conf
```

1.4.3 Install Murano Dashboard

Murano API & Engine services provide the core of Murano. However, your need a control plane to use it. This section decribes how to install and run Murano Dashboard.

1. Clone the repository with Murano Dashboard.

```
$ cd ~/murano
$ git clone git://git.openstack.org/openstack/murano-dashboard
```

2. Clone horizon repository

```
$ git clone git://git.openstack.org/openstack/horizon
```

3. Create venv and install muranodashboard as editable module.

```
$ cd horizon
$ tox -e venv -- pip install -e ../murano-dashboard
```

4. Copy muranodashboard plugin file.

This step enables murano panel in horizon dashboard.

```
$ cp ../murano-dashboard/muranodashboard/local/_50_murano.py openstack_dashboard/local/enabled/
```

5. Prepare local settings.

   To get more information, check out official horizon documentation.

   ```
   $ cp openstack_dashboard/local/local_settings.py.example openstack_dashboard/local/local_setting
   ```

6. Customize local settings according to Openstack installation.

   ```
   ...
   ALLOWED_HOSTS = '*'

   # Provide OpenStack Lab credentials
   OPENSTACK_HOST = '%OPENSTACK_HOST_IP%'

   ...

   # Set secret key to prevent it's generation
   SECRET_KEY = 'random_string'

   ...

   DEBUG_PROPAGATE_EXCEPTIONS = DEBUG
   ```

   Also, it's better to change default session backend from browser cookies to database to avoid issues with forms during creating applications:

   ```
   ...
   DATABASES = {
       'default': {
       'ENGINE': 'django.db.backends.sqlite3',
       'NAME': '/tmp/murano-dashboard.sqlite',
       }
   }

   SESSION_ENGINE = 'django.contrib.sessions.backends.db'
   ```

   If you do not plan to get murano service from keystone application catalog, provide where murano-api service is running:

   ```
   ...
   MURANO_API_URL = 'http://localhost:8082'
   ```

7. Perform database synchronization.

   Optional step. Needed in case you set up database as a session backend.

   ```
   $ tox -e venv -- python manage.py syncdb
   ```

   You can reply 'no' since for development purpose separate user is not needed.

8. Run Django server at 127.0.0.1:8000 or provide different IP and PORT parameters.

   ```
   $ tox -e venv -- python manage.py runserver <IP:PORT>
   ```

   Development server will be restarted automatically on every code change.

9. Open dashboard using url http://localhost:8000

1.4.4 Import Murano Applications

Murano provides excellent catalog services, but it also requires applications which to provide. This section describes how to import Murano Applications from Murano App Incubator.

1. Clone Murano App Incubator repository.

```
$ cd ~/murano
$ git clone git://git.openstack.org/openstack/murano-apps
```

2. Import every package you need from this repository, using the command below.

```
$ cd ~/murano/murano
$ tox -e venv -- murano-manage \
> --config-file ./etc/murano/murano.conf \
> import-package ../murano-app-incubator/%APPLICATION_DIRECTORY_NAME%
```

Network Configuration

Murano may work in various networking environments and is capable to detect the current network configuration and choose the appropriate settings automatically. However, some additional actions are required to support advanced scenarios.

Nova network support

Nova Network is simplest networking solution, which has limited capabilities but is available on any OpenStack deployment without the need to deploy any additional components.

When a new Murano Environment is created, Murano checks if a dedicated networking service (i.e. Neutron) exists in the current OpenStack deployment. It relies on Keystone's service catalog for that. If such a service is not present, Murano automatically falls back to Nova Network. No further configuration is needed in this case, all the VMs spawned by Murano will be joining the same Network.

Neutron support

If Neutron is installed, Murano enables its advanced networking features that give you ability to not care about configuring networks for your application.

By default it will create an isolated network for each environment and join all VMs needed by your application to that network. To install and configure application in just spawned virtual machine Murano also requires a router connected to the external network.

Automatic Neutron network configuration

To create router automatically, provide the following parameters in config file:

```
[networking]

external_network = %EXTERNAL_NETWORK_NAME%
router_name = %MURANO_ROUTER_NAME%
create_router = true
```

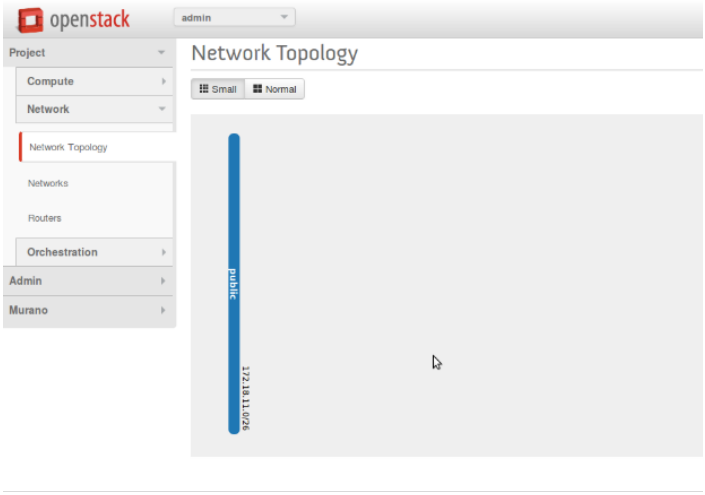
To figure out the name of the external network, perform the following command:

```
$ neutron net-external-list
```

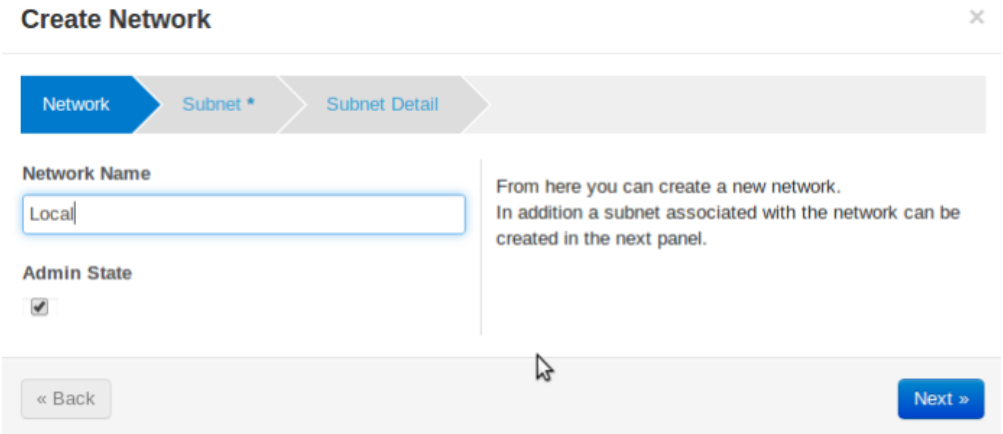
During the first deploy, required networks and router with specified name will be created and set up.

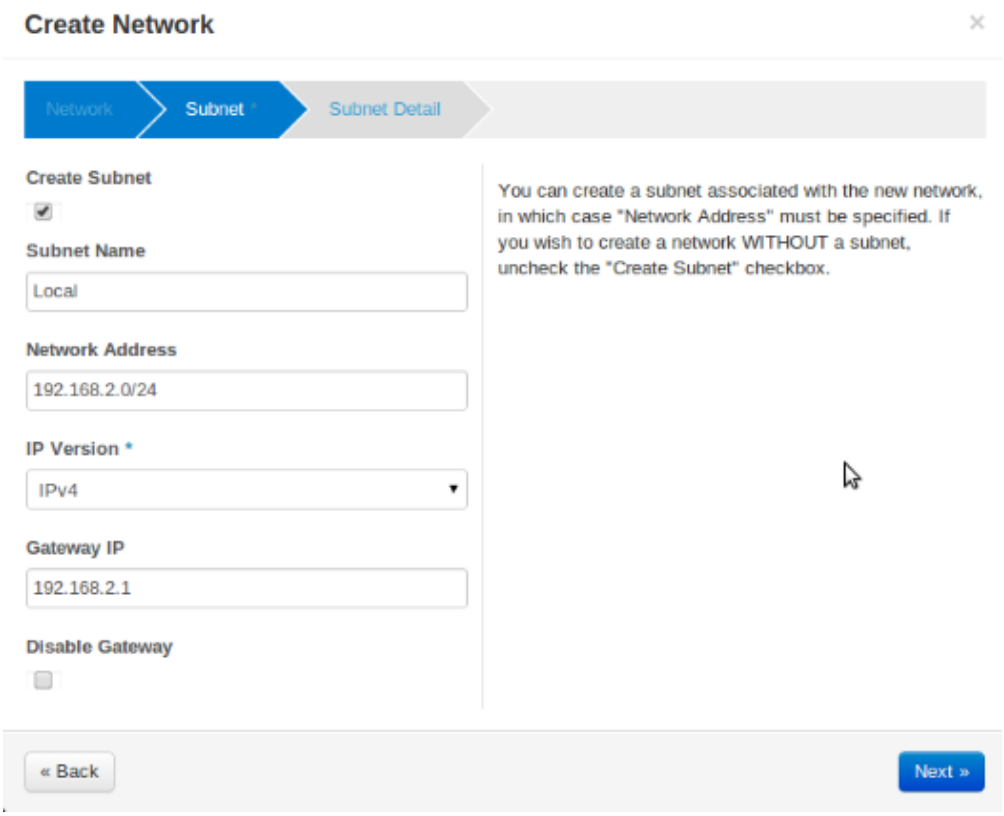
Manual neutron network configuration

- Step 1. Create public network
  - First, you need to check for existence of external networks. Login as admin and go to *Project -> Network -> Network Topology*. And check network type in network details at *Admin -> Networks -> Network name* page. The same action can be done via CLI by running *neutron net-external-list*. To create new external network examine OpenStack documentation.

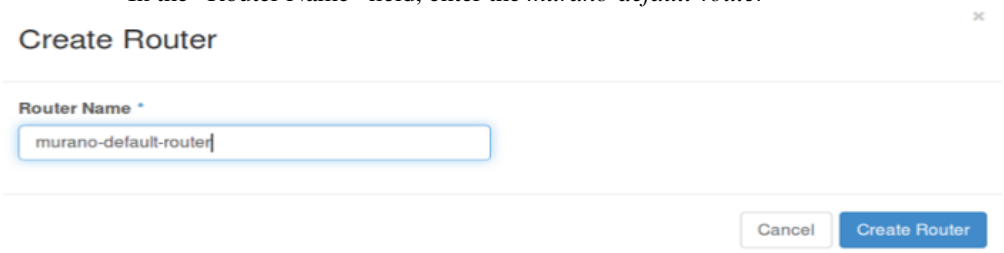


- Step 2. Create local network
  - Go to *Project -> Network -> Networks*.
  - Click *Create Network* and fill the form.





- Step 3. Create router
  - Go to *Project -> Network -> Routers*
  - Click “Create Router”
  - In the “Router Name” field, enter the *murano-default-router*

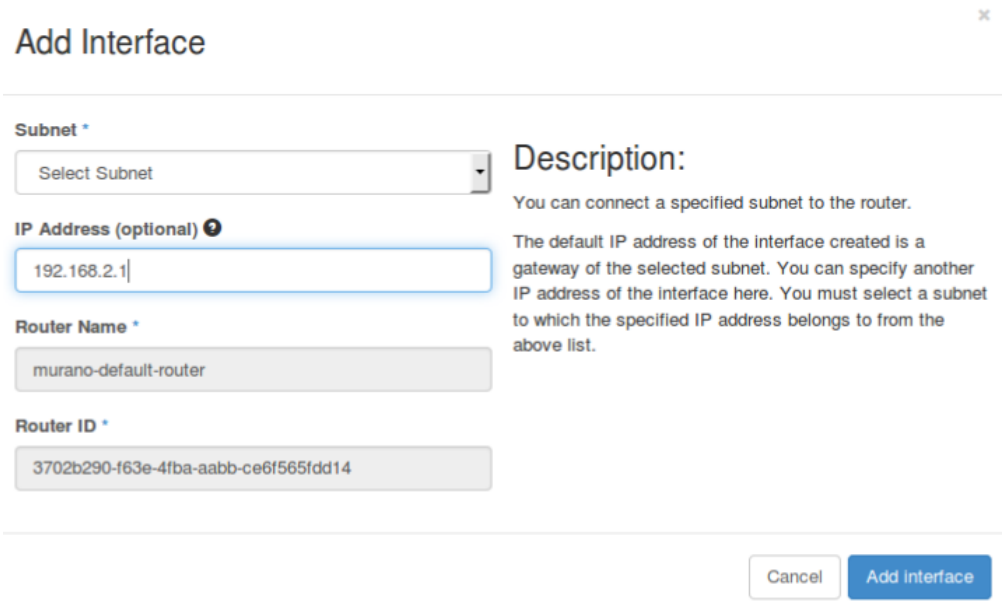


If you specify a name other than *murano-default-router*, it will be necessary to change the following settings in the config file:

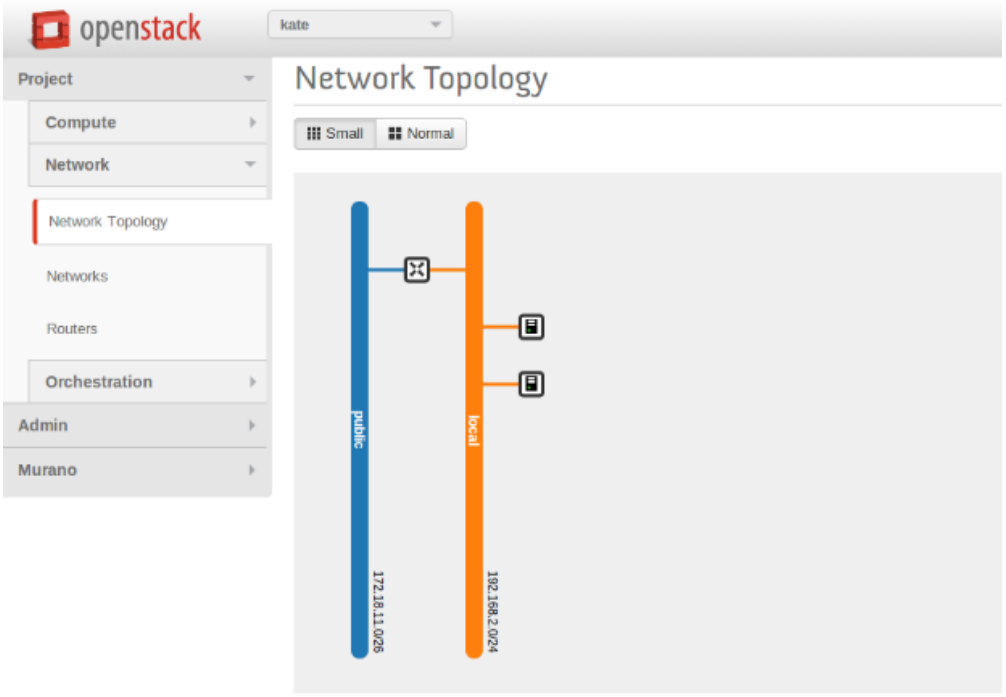
```
[networking]

router_name = %SPECIFIED_NAME%
create_router = false
```

- Click on the specified router name
- In the opened view click “Add interface”
- Specify the subnet and IP address



And check the result in *Network Topology* tab.



Background Concepts for Murano

1.5 MuranoPL: Murano Programming Language

1.5.1 Content

YAML

YAML is human-readable data serialization format that is a superset of JSON. Unlike JSON YAML was designed to be read and written by humans and relies on visual indentation to denote nesting of data structures. This is similar to how Python uses indentation for block structures instead of curly brackets in most C-like languages. Also YAML can contain more data types comparing to JSON. See http://yaml.org/ for detailed description of YAML.

MuranoPL was designed to be representable in YAML so that MuranoPL code could remain readable and structured. Thus usually MuranoPL files are YAML encoded documents. But MuranoPL engine itself doesn't deal directly with YAML documents and it is up to hosting application to locate and deserialize definitions of particular classes. This gives hosting application ability to control where those definitions can be found (file system, database, remote repository etc) and possibly use some other serialization formats instead of YAML.

MuranoPL engine relies on host deserialization code to automatically detect YAQL expressions in source definition and to provi

```
Some text - a string,
$.something() - YAQL
"$.something()" - string (because of quote marks)
!!str $ - a string (because of YAML tag)
!yaql "text" - YAQL  (because of YAML tag)
```

YAQL

YAQL (Yet Another Query Language) is a query language that was also designed as part of Murano project. MuranoPL makes an extensive use of YAQL. YAQL description can be found here: https://github.com/ativelkov/yaql

In simple words YAQL is a language for expression evaluation. `2 + 2, foo() > bar(), true != false` are all valid YAQL expressions. The interesting thing in YAQL is that it has no built in list of functions. Everything YAQL can access is customizable. YAQL cannot call any function that was not explicitly registered to be accessible by YAQL. The same is true for operators. So the result of expression 2 * foo(3, 4) is completely depended on explicitly provided implementations of "foo" and "operator_*". YAQL uses dollar sign ($) to access external variables (that are also explicitly provided by host application) and function arguments. `$variable` is a syntax to get the value of variable "$variable", $1, $2 etc are the names for function arguments. "$" is a name for current object - data on which the expression is evaluated or a name of a single argument. Thus $ in the beginning of expression and $ in middle of it can refer to different things.

YAQL has a lot of functions out of the box that can be registered in YAQL context. For example

`$.where($.myObj.myScalar > 5 and $.myObj.myArray.len() > 0 and $.myObj.myArray.any($ = 4)).select($.myObj.myArray[0])` can be executed on `$ = array` of objects and has a result of another array that is a filtration and projection of a source data. This is very similar to how SQL works but uses more Python-like syntax.

Note that there is no assignment operator in YAQL and '=' means comparision operator that is what '==' means in Python.

Because YAQL has no access to underlying operating system resources and 100% controllable by the host it is secure to execute YAQL expressions without establishing a trust to executed code. Also because of the functions are not predefined different functions may be accessible in different contexts. So the YAQL expressions that are used to specify property contracts are not necessarily valid in workflow definitions.

Common class structure

Here is a common template for class declarations. In sections below I'm going to explain what each section means. Note that it is in YAML format.

```
Name: class name
Namespaces: namespaces specification
Extends: [list of parent classes]
Properties: properties declaration
Workflow:
    methodName:
        Arguments:
            - list
            - of
            - arguments
        Body:
            - list
            - of
            - instructions
```

Thus MuranoPL class is a YAML dictionary with predefined key names. All keys except for Name are optional and can be omitted (but must be valid if present)

Class name

Class names are alphanumeric names of the classes. By tradition all class names begin with upper-case letter and written in PascalCasing.

In Murano all class names are globally unique. This achieved by means of namespaces. Class name may have explicit namespace specification (like ns:MyName) or implicit (just MyName which would be equal to =:MyName if = was a valid in name specification)

Namespaces

Namespaces declaration specifies prefixes that can be used in class body to make long class names shorter.

```
Namespaces:
    =: io.murano.services.windows
    srv: io.murano.services
    std: io.murano
```

In example above class name srv:Something would be automatically translated to "io.murano.services.Something".

"=" means "current namespace" so that "MyClass" would mean "io.murano.services.windows.MyClass" in example above.

If class name contains period sign (.) in its name then it is assumed to be already fully namespace-qualified and is not expanded. Thus ns.Myclass would remain as is.

To make class names globally unique it is recommended to have developer's domain name as part of namespace (as in example, similar to Java)

Extends

MuranoPL supports multiple inheritance. If present, Extends section lists base classes that are extended. If the list consists of single entry then it may be written as a scalar string instead of array. If no parents specified (or a key is

omitted) then "io.murano.Object" is assumed making it the root class for all class hierarchies.

Properties

Properties are class attributes that together with methods form public class interface. Usually (but not always) properties are the values and references to other objects that are required to be entered in environment designer prior to workflow invocation.

Properties have the following declaration format:

```
propertyName:
    Contract: property contract
    Usage: property usage
    Default: property default
```

Contract Contracts are YAQL expressions that say what type of value is expected for the property as well as additional constraints imposed on the property.

Operation	Definition
$.int()	integer value (may be null). String values that consist of digits would be converted to integer
$.int().notNull()	mandatory integer
$.string()	the same for strings. If the supplied value is not a string it will be converted to string
$.string().notNull()	
$.bool()	bools are true and false. 0 is converted to false, other integers to true
$.bool().notNull()	
$.class(ns:ClassName)	value must be a reference to an instance of specified class name
$.class(ns:ClassName).notNull()	
$.class(ns:ClassName, ns:DefaultClassName)	create instance of ns:DefaultClassName class if no instance provided
$.class(ns:Name).check($.p = 12)	value must be of type ns:Name and have a property 'p' equal to 12
[$.int()]	array of integers. Similar for other types
[$.int().notNull()]	
[$.int().check($ > 0)]	array of positive integers (thus not null)
[$.int(), $.string()]	array that has at least two elements, first is int and others are strings
[$.int(), 2]	array of ints with at least 2 items
[$.int(), 2, 5]	... and maximum of 5 items
{ A: $.int(), B: [$.string()] }	dictionary with 'A' key of type int and 'B' - array of strings
$	any scalar or data structure as is any array any dictionary
[]	
{}	
{ $.string().notNull(): $.int().notNull() }	dictionary string -> int
A: StringMap $.string().notNull(): $	dictionary with 'A' key that must be equal to 'StringMap' and other keys be any scalar or data structure

Usage Usage states purpose of the property. This implies who and how can access it. The following usages are available:

Property	Explanation
`In`	Input property. Values of such properties are obtained from user and cannot be modified in MuranoPL workflows. This is default value for Usage key
`Out`	The value is obtained from executing MuranoPL workflow and cannot be modified by the user
`InOut`	Value can be edited by both user and workflow
`Const`	The same as In but once workflow is executed the property cannot be changed neither by user not the workflow
`Runtime`	Property is visible only from within workflows. It neither read from input neither serialized to workflow output

Usage attribute is optional and can be omitted (which implies In).

If the workflow tries to write to a property that is not declared with one of the types above it is considered to be private and accessible only to that class (and not serialized to output and thus would be lost upon next deployment). Attempt to read property that wasn't initialized causes exception to be thrown.

Default Default is a value that would be used if the property value wasn't mentioned in input object model (but not when it is provided as null). Default (if specified) must conform to declared property contract. If Default is not specified then null is the default.

For properties that are references to other classes Default can modify default values for referenced value. For example

```
p:
  Contract: $.class(MyClass)
  Default: {a: 12}
```

would override default for 'a' property of MyClass for instance of MyClass that is created for this property.

Workflow

Workflows are the methods that together describe how the entities that are represented by MuranoPL classes are deployed.

In typical scenario root object in input data model is of type io.murano.Environment and has a "deploy" method. Invoking this method causes a series of infrastructure activities (typically by modifying Heat stack) and VM agents commands that cause execution of deployment scripts. Workflow role is to map data from input object model (or result of previously executed actions) to parameters of those activities and to initiate those activities in correct order. Methods have input parameters and can return value to the caller. Methods defined in Workflow section of the class using the following template:

```
methodName:
    Arguments:
        - list
        - of
        - arguments
    Body:
        - list
        - of
        - instructions
```

Arguments are optional and (if specified) declared using the same syntax as class properties except for Usage attribute that is meaningless for method parameters. E.g. arguments also have a contract and optional default.

Method body is an array of instructions that got executed sequentially. There are 3 types of instructions that can be found in workflow body: expressions, assignment and block constructs.

Expressions Expressions are YAQL expressions that are executed for their side effect. All accessible object methods can be called in expression using $obj.methodName(arguments) syntax.

Expression	Explanation
`$.methodName()` `$this.methodName()`	invoke method 'methodName' on this (self) object
`$.property.methodName()` `$this.property.methodName()`	invocation of method on object that is in 'property' property
`$.method(1, 2, 3)`	methods can have arguments
`$.method(1, 2, thirdParameter => 3)`	named parameters also supported
`list($.foo().bar($this.property), $p)`	complex expressions can be constructed

Assignment Assignments are single-key dictionaries with YAQL expression as key and arbitrary structure as a value. Such construct evaluated as assignment.

Assignment	Explanation
`$x: value`	assigns 'value' to local variable $x
`$.x: value $this.x: value`	assign value to object's property
`$.x: $.y`	copy value of property 'y' to property 'x'
`$x: [$a, $b]`	sets $x to array of 2 values $a and $b

`$x:`
`  SomeKey:`
`  NestedKey:`
`  $variable`
	structures of any level of complexity can be evaluated

`$.x[0]: value'`	assign value to a first array entry of property x
`$.x.append(): value`	append value to array in property x
`$.x.insert(1): value`	insert value into position 1
`$.x.key.subKey: value` `$.x[key][subKey]: value`	deep dictionary modification

Block constructs Block constructs control program flow. Block constructs are dictionaries that have strings as all its keys. The following block constructs are available:

Assignment	Explanation
`Return: value`	return value from a method

`If: predicate()`
`Then:`
`- code`
`- block`
`Else:`
`- code`
`- block`
	predicate() is YAQL expressions that must be evaluated to true or false.
	else part is optional
	one-line code blocks can be written as a scalars rather than array.

`While: predicate()` `Do:|- code|- block`	predicate() must be evaluated to true or false

`For: variableName`
`In: collection`
`Do:`
`- code`
`- block`
	collection must be YAQL expression returning iterable collection or evaluatable array as in assignment instructions (like [1, 2, $x])
	inside code block loop variable is accessible as $variableName

`Repeat:`
`Do:`
`- code`
`- block`
	repeat code block specified number of times

Break:	breaks from loop

`Match:`
`case1:`
`- code`
`- block`
`case2:`
`- code`
`- block`
`Value: $valueExpression()`
`Default:`
`- code`
`- block`
	matches result of $valueExpression() against set of possible values (cases). the code block of first matched cased is executed.
	if not case matched and Default key is present (it is optional) than Default code block get executed.
	case values are constant values (not expressions)

`Switch:`
`$predicate1() :`
`- code`
`- block`
`$predicate2() :`
`- code`
`- block`
`Default:`
`- code`
	all code blocks that have their predicate evaluated to true are executed of predicate evaluation is not fixed
	default key is optional.
	if no predicate evaluated to true than Default code block get executed.

Object model

Object model is JSON-serialized representation of objects and their properties. Everything user does in environment builder (dashboard) is reflected in object model. Object model is sent to App Catalog engine upon user decides to deploy built environment. On engine side MuranoPL objects are constructed and initialized from received Object model and predefined method is executed on a root object.

Objects serialized to JSON using the following template:

```
{
    "?": {
        "id": "globally unique object ID (UUID)",
        "type": "fully namespace-qualified class name",

        "optional designer-related entries can be placed here": {
            "key": "value"
        }
    },

    "classProperty1": "propertyValue",
    "classProperty2": 123,
    "classProperty3": ["value1", "value2"],

    "reference1": {
        "?": {
            "id": "object id",
            "type": "object type"
        },

        "property": "value"
    },

    "reference2": "referenced object id"
}
```

Objects can be identified as dictionaries that contain "?" entry. All system fields are hidden in that entry.

There are 2 ways to specify references. The first method ("reference1" in example above) allow inline definition of object. When instance of referenced object is created outer object becomes its parent (owner) that is responsible for the object. The object itself may require that its parent (direct or indirect) be of specified type (like all application require to have Environment somewhere in parent chain).

Second way to reference object is by specifying other object id. That object must be defined somewhere else in object tree. Object references distinguished from strings having the same value by evaluating property contracts. The former case would have $.class(Name) while the later $.string() contract.

Murano PL System Class Definitions

Murano program language has system classes, which make deploying process as convenient as it could be. System classes are used in user class definitions for a custom applications. This article is going to help users to operate with Murano PL classes without any issues. All classes are located in the murano-engine component and don't require particular import.

- *io.murano.system.Resources*
- *io.murano.system.Agent*
- *io.murano.system.AgentListener*

- *io.murano.system.HeatStack*
- *io.murano.system.InstanceNotifier*
- *io.murano.system.NetworkExplorer*
- *io.murano.system.StatusReporter*

io.murano.system.Resources

Used to provide API to all files, located in the Resource directory of application package. Those Resources usually used in an application deployment and needed to be specified in a workflow definition. Available methods:

- *yaml* return resource file in yaml format
- *string* return resource file as string
- *json* return resource in json format

io.murano.system.Agent

Defines Murano Agent and ways of interacting with it. Available methods:

- *call(template, resources)* - send an execution plan template and resource object, and wait for an operation to complete
- *send(template, resources)* - send execution plan template and resource class instance and continue execution without waiting for an end of the execution
- *callRaw(plan)* - send ready-to-perform murano agent execution plan and wait for an operation to complete
- *sendRaw(plan)* - send ready-to-perform murano agent execution plan and continue workflow execution
- *queueName()* - returns name of the queue with which Agent is working

io.murano.system.AgentListener

Used for monitoring Murano Agent.

- *start()* - start to monitor Murano Agent activity
- *stop()* - stop to monitor Murano Agent activity
- *subscribe(message_id, event)* - subscribe to the specified Agent event
- *queueName()* - returns name of the queue with which Agent is working

io.murano.system.HeatStack

Manage Heat stack operations.

- *current()* - returns current heat template
- *parameters()* - returns heat template parameters
- *reload()* - reload heat template
- *setTemplate(template)* - load heat template
- *updateTemplate(template)* - update current template with the specified part of heat stack

- *output()* - result of heat template execution
- *push()* - commit changes (requires after setTemplate and updateTemplate operations)
- *delete()* - delete current heat stack

io.murano.system.InstanceNotifier

Monitor application and instance statistics to provide billing feature.

- *trackApplication(instance,* `title`, `unitCount`) - start to monitor an application activity; title, unitCount - are optional
- *untrackApplication(instance)* - stop to monitor an application activity
- *trackCloudInstance(instance)* - start to monitor an instance activity
- *untrackCloudInstance(instance)* - stop to monitor an instance activity

io.murano.system.NetworkExplorer

Determines and configures network topology.

- *getDefaultRouter()* - determine default router
- *getAvailableCidr(routerId, netId)* - searching for non-allocated CIDR
- *getDefaultDns()* - get dns from config file
- *getExternalNetworkIdForRouter(routerId)* - Check for router connected to the external network
- *getExternalNetworkIdForNetwork(networkId)* - For each router this network is connected to check whether the router has external_gateway set

io.murano.system.StatusReporter

Provides feedback feature. To follow the deployment process in the UI, all status changes should be included in the application configuration.

- *report(instance, msg)* - Send message about an application deployment process
- *report_error(instance, msg)* - Report an error during an application deployment process

MuranoPL Core Library

Some objects and actions could be used in several application deployments. All common parts are grouped into MuranoPL libraries. Murano core library is a set of classes needed in every deployment. Class names from core library could be used in the application definitions. This library is located under the meta directory. The following classes are included into the Murano core library:

io.murano:

- *Class: Object*
- *Class: Application*
- *Class: SecurityGroupManager*
- *Class: Environment*

io.murano.resources:

- *Class: Instance*

  Resources:

  - Agent-v1.template
  - Agent-v2.template
  - linux-init.sh
  - windows-init.sh

- *Class: Network*

io.murano.lib.networks.neutron:

- *Class: NewNetwork*

Class: Object

Parent class for all MuranoPL classes, which implements initialize method, and setAttr and getAttr methods, which are defined in the pythonic part of the Object class. All MuranoPL classes are implicitly inherited from this class.

Class: Application

Defines application itself. All custom applications should be derived from this class. Has two properties:

```
Namespaces:
    =: io.murano

Name: Application

Workflow:
  reportDeployed:
      Arguments:
        - title:
            Contract: $.string()
            Default: null
        - unitCount:
            Contract: $.int()
            Default: null
      Body:
        - $this.find(Environment).instanceNotifier.trackApplication($this, $title, $unitCount)

  reportDestroyed:
      Body:
        - $this.find(Environment).instanceNotifier.untrackApplication($this)
```

Class: SecurityGroupManager

Manages security groups during application deployment.

```
Namespaces:
    =: io.murano.system
    std: io.murano
```

```
Name: SecurityGroupManager

Properties:
  environment:
    Contract: $.class(std:Environment).notNull()

  defaultGroupName:
    Contract: $.string()
    Usage: Runtime
    Default: format('MuranoSecurityGroup-{0}', $.environment.name)

Workflow:
  addGroupIngress:
    Arguments:
      - rules:
          Contract:
            - FromPort: $.int().notNull()
              ToPort: $.int().notNull()
              IpProtocol: $.string().notNull()
              External: $.bool().notNull()
      - groupName:
          Contract: $.string().notNull()
          Default: $this.defaultGroupName
    Body:
      - $ext_keys:
          true:
            ext_key: remote_ip_prefix
            ext_val: '0.0.0.0/0'
          false:
            ext_key: remote_mode
            ext_val: remote_group_id

      - $stack: $.environment.stack
      - $template:
          Resources:
            $groupName:
              Type: 'OS::Neutron::SecurityGroup'
              Properties:
                description: format('Composite security group of Murano environment {0}', $.environme
                rules:
                  - port_range_min: null
                    port_range_max: null
                    protocol: icmp
                    remote_ip_prefix: '0.0.0.0/0'
      - $.environment.stack.updateTemplate($template)

      - $ingress: $rules.select(dict(
            port_range_min => $.FromPort,
            port_range_max => $.ToPort,
            protocol => $.IpProtocol,
            $ext_keys.get($.External).ext_key => $ext_keys.get($.External).ext_val
          ))

      - $template:
          Resources:
            $groupName:
              Type: 'OS::Neutron::SecurityGroup'
              Properties:
```

```
            rules: $ingress
      - $.environment.stack.updateTemplate($template)
```

Class: Environment

Defines an Environment in terms of deployments process. Groups all the Applications and their related infrastructure, able to deploy them at once. Environments is intent to group applications to manage them easily.

- *name* - an environment name
- *applications* - list of applications belonging to an environment
- *agentListener* - property containing a ‘ *io.murano.system.AgentListener* object, which may be used to interact with Murano Agent
- *stack* - a property containing a HeatStack object which may be used to interact with the Heat Service
- *instanceNotifier* - a property containing a *io.murano.system.InstanceNotifier* which may be used to keep track of the amount of deployed instances
- *defaultNetworks* - a property containing user-defined Networks (*io.murano.resources.Network*), which may be used as the default networks for the Instances in this environment
- *securityGroupManager*- a property containing a *SecurityGroupManager* object, which may be used to construct a security group associated with this environment

```
Namespaces:
    =: io.murano
    res: io.murano.resources
    sys: io.murano.system

Name: Environment

Properties:
  name:
    Contract: $.string().notNull()

  applications:
    Contract: [$.class(Application).owned().notNull()]

  agentListener:
    Contract: $.class(sys:AgentListener)
    Usage: Runtime

  stack:
    Contract: $.class(sys:HeatStack)
    Usage: Runtime

  instanceNotifier:
    Contract: $.class(sys:InstanceNotifier)
    Usage: Runtime

  defaultNetworks:
    Contract:
      environment: $.class(res:Network)
      flat: $.class(res:Network)
    Usage: In

  securityGroupManager:
```

```
    Contract: $.class(sys:SecurityGroupManager)
    Usage: Runtime

Workflow:
  initialize:
    Body:
      - $this.agentListener: new(sys:AgentListener, name => $.name)
      - $this.stack: new(sys:HeatStack, name => $.name)
      - $this.instanceNotifier: new(sys:InstanceNotifier, environment => $this)
      - $this.reporter: new(sys:StatusReporter, environment => $this)
      - $this.securityGroupManager: new(sys:SecurityGroupManager, environment => $this)


  deploy:
    Body:
      - $.agentListener.start()
      - If: len($.applications) = 0
        Then:
          - $.stack.delete()
        Else:
          - $.applications.pselect($.deploy())
      - $.agentListener.stop()
```

Class: Instance

Defines virtual machine parameters and manage instance lifecycle: spawning, deploying, joining to the network, applying security group and destroying.

- *name* - instance name
- *flavor* - instance flavor, defining virtual machine 'hardware' parameters
- *image* - instance image, defining operation system
- *keyname* - key pair name, used to make connect easily to the instance; optional
- *agent* - configures interaction with Murano Agent using *MuranoPL system class*
- *ipAddresses* - list of all IP addresses, assigned to an instance
- ***networks* - configures type of networks, to which instance will be joined.** Custom networks, that extends *Network class* could be specified and an instance will be connected to them and for a default environment network or flat network if corresponding values are set to true; without additional configurations, instance will be joined to the default network that are set in the current environment.
- *assignFloatingIp* - determines, if floating IP need to be assigned to an instance, default is false
- *floatingIpAddress* - IP addresses, assigned to an instance after an application deployment
- *securityGroupName* - security group, to which instance will be joined, could be set; optional

```
Namespaces:
  =: io.murano.resources
  std: io.murano
  sys: io.murano.system

Name: Instance

Properties:
  name:
```

```
    Contract: $.string().notNull()
  flavor:
    Contract: $.string().notNull()
  image:
    Contract: $.string().notNull()
  keyname:
    Contract: $.string()
    Default: null

  agent:
    Contract: $.class(sys:Agent)
    Usage: Runtime
  ipAddresses:
    Contract: [$.string()]
    Usage: Out
  networks:
    Contract:
      useEnvironmentNetwork: $.bool().notNull()
      useFlatNetwork: $.bool().notNull()
      customNetworks: [$.class(Network).notNull()]
    Default:
      useEnvironmentNetwork: true
      useFlatNetwork: false
      customNetworks: []
  assignFloatingIp:
    Contract: $.bool().notNull()
    Default: false
  floatingIpAddress:
    Contract: $.string()
    Usage: Out
  securityGroupName:
    Contract: $.string()
    Default: null

Workflow:
  initialize:
    Body:
      - $.environment: $.find(std:Environment).require()
      - $.agent: new(sys:Agent, host => $)
      - $.resources: new(sys:Resources)

  deploy:
    Body:
      - $securityGroupName: coalesce(
            $.securityGroupName,
            $.environment.securityGroupManager.defaultGroupName
          )
      - $.createDefaultInstanceSecurityGroupRules($securityGroupName)

      - If: $.networks.useEnvironmentNetwork
        Then:
          $.joinNet($.environment.defaultNetworks.environment, $securityGroupName)
      - If: $.networks.useFlatNetwork
        Then:
          $.joinNet($.environment.defaultNetworks.flat, $securityGroupName)
      - $.networks.customNetworks.select($this.joinNet($, $securityGroupName))

      - $userData: $.prepareUserData()
```

```
    - $template:
        Resources:
          $.name:
            Type: 'AWS::EC2::Instance'
            Properties:
              InstanceType: $.flavor
              ImageId: $.image
              UserData: $userData
              KeyName: $.keyname

        Outputs:
          format('{0}-PublicIp', $.name):
            Value:
              - Fn::GetAtt: [$.name, PublicIp]
    - $.environment.stack.updateTemplate($template)
    - $.environment.stack.push()
    - $outputs: $.environment.stack.output()
    - $.ipAddresses: $outputs.get(format('{0}-PublicIp', $this.name))
    - $.floatingIpAddress: $outputs.get(format('{0}-FloatingIPaddress', $this.name))
    - $.environment.instanceNotifier.trackApplication($this)

  joinNet:
    Arguments:
      - net:
          Contract: $.class(Network)
      - securityGroupName:
          Contract: $.string()
    Body:
      - If: $net != null
        Then:
          - If: $.assignFloatingIp and (not bool($.getAttr(fipAssigned)))
            Then:
              - $assignFip: true
              - $.setAttr(fipAssigned, true)
            Else:
              - $assignFip: false
          - $net.addHostToNetwork($, $assignFip, $securityGroupName)

  destroy:
    Body:
      - $template: $.environment.stack.current()
      - $patchBlock:
          op: remove
          path: format('/Resources/{0}', $.name)
      - $template: patch($template, $patchBlock)
      - $.environment.stack.setTemplate($template)
      - $.environment.stack.push()
      - $.environment.instanceNotifier.untrackApplication($this)

  createDefaultInstanceSecurityGroupRules:
    Arguments:
      - groupName:
          Contract: $.string().notNull()
    Body:

      - If: !yaql "'w' in toLower($.image)"
        Then:
          - $rules:
```

```
            - ToPort: 3389
              IpProtocol: tcp
              FromPort: 3389
              External: true
      Else:
        - $rules:
            - ToPort: 22
              IpProtocol: tcp
              FromPort: 22
              External: true
    - $.environment.securityGroupManager.addGroupIngress(
        rules => $rules, groupName => $groupName)

getDefaultSecurityRules:
prepareUserData:
  Body:
    - If: !yaql "'w' in toLower($.image)"
      Then:
        - $configFile: $.resources.string('Agent-v1.template')
        - $initScript: $.resources.string('windows-init.ps1')
      Else:
        - $configFile: $.resources.string('Agent-v2.template')
        - $initScript: $.resources.string('linux-init.sh')

    - $configReplacements:
        "%RABBITMQ_HOST%": config(rabbitmq, host)
        "%RABBITMQ_PORT%": config(rabbitmq, port)
        "%RABBITMQ_USER%": config(rabbitmq, login)
        "%RABBITMQ_PASSWORD%": config(rabbitmq, password)
        "%RABBITMQ_VHOST%": config(rabbitmq, virtual_host)
        "%RABBITMQ_SSL%": str(config(rabbitmq, ssl)).toLower()
        "%RABBITMQ_INPUT_QUEUE%": $.agent.queueName()
        "%RESULT_QUEUE%": $.environment.agentListener.queueName()

    - $scriptReplacements:
        "%AGENT_CONFIG_BASE64%": base64encode($configFile.replace($configReplacements))
        "%INTERNAL_HOSTNAME%": $.name
        "%MURANO_SERVER_ADDRESS%": coalesce(config(file_server), config(rabbitmq, host))
        "%CA_ROOT_CERT_BASE64%": ""

    - Return: $initScript.replace($scriptReplacements)
```

Instance class uses the following resources:

- *Agent-v2.template* - Python Murano Agent template (This agent is unified and lately, Windows Agent will be included into it)
- *linux-init.sh* - Python Murano Agent initialization script, which sets up an agent with valid information, containing in updated agent template.
- *Agent-v1.template* - Windows Murano Agent template
- *windows-init.sh* - Windows Murano Agent initialization script

Class: Network

Base abstract class for all MuranoPL classes, representing networks.

```
Namespaces:
    =: io.murano.resources

Name: Network

Workflow:
  addHostToNetwork:
    Arguments:
      - instance:
          Contract: $.class(Instance).notNull()
      - assignFloatingIp:
          Contract: $.bool().notNull()
          Default: false
      - securityGroupName:
          Contract: $.string()
          Default: null
```

Class: NewNetwork

Defining network type, using in Neutron.

- *name* - network name
- *autoUplink* - defines auto uplink network parameter; optional, turned on by default
- *autogenerateSubnet* - defines auto subnet generation; optional, turned on by default
- *subnetCidr* - CIDR, defining network subnet, optional
- *dnsNameserver* - DNS server name, optional
- *useDefaultDns* - defines ether set default DNS or not, optional, turned on by default

```
Namespaces:
  =: io.murano.lib.networks.neutron
  res: io.murano.resources
  std: io.murano
  sys: io.murano.system

Name: NewNetwork

Extends: res:Network

Properties:
  name:
    Contract: $.string().notNull()

  externalRouterId:
    Contract: $.string()
    Usage: InOut

  autoUplink:
    Contract: $.bool().notNull()
    Default: true

  autogenerateSubnet:
    Contract: $.bool().notNull()
    Default: true
```

```
  subnetCidr:
    Contract: $.string()
    Usage: InOut

  dnsNameserver:
    Contract: $.string()
    Usage: InOut

  useDefaultDns:
    Contract: $.bool().notNull()
    Default: true

Workflow:
  initialize:
    Body:
      - $.environment: $.find(std:Environment).require()
      - $.netExplorer: new(sys:NetworkExplorer)

  deploy:
    Body:
      - $.ensureNetworkConfigured()
      - $.environment.instanceNotifier.untrackApplication($this)

  addHostToNetwork:
    Arguments:
      - instance:
          Contract: $.class(res:Instance).notNull()
      - assignFloatingIp:
          Contract: $.bool().notNull()
          Default: false
      - securityGroupName:
          Contract: $.string()
          Default: null
    Body:
      - $.ensureNetworkConfigured()
      - $portname: $instance.name + '-port-to-' + $.id()
      - $template:
          Resources:
            $portname:
              Type: 'OS::Neutron::Port'
              Properties:
                network_id: {Ref: $.net_res_name}
                fixed_ips: [{subnet_id: {Ref: $.subnet_res_name}}]
                security_groups:
                  - Ref: $securityGroupName
            $instance.name:
              Properties:
                NetworkInterfaces:
                  - Ref: $portname
      - $.environment.stack.updateTemplate($template)

      - If: $assignFloatingIp
        Then:
          - $extNetId: $.netExplorer.getExternalNetworkIdForRouter($.externalRouterId)
          - If: $extNetId != null
            Then:
              - $fip_name: $instance.name + '-FloatingIP-' + $.id()
              - $template:
```

```
                Resources:
                  $fip_name:
                    Type: 'OS::Neutron::FloatingIP'
                    Properties:
                      floating_network_id: $extNetId
                  $instance.name + '-FloatingIpAssoc-' + $.id():
                    Type: 'OS::Neutron::FloatingIPAssociation'
                    Properties:
                      floatingip_id:
                        Ref: $fip_name
                      port_id:
                        Ref: $portname
                Outputs:
                  $instance.name + '-FloatingIPaddress':
                    Value:
                      Fn::GetAtt:
                        - $fip_name
                        - floating_ip_address
                    Description: Floating IP assigned
            - $.environment.stack.updateTemplate($template)

ensureNetworkConfigured:
  Body:
    - If: !yaql "not bool($.getAttr(networkConfigured))"
      Then:
        - If: $.useDefaultDns and (not bool($.dnsNameserver))
          Then:
            - $.dnsNameserver: $.netExplorer.getDefaultDns()

        - $.net_res_name: $.name + '-net-' + $.id()
        - $.subnet_res_name: $.name + '-subnet-' + $.id()
        - $.createNetwork()
        - If: $.autoUplink and (not bool($.externalRouterId))
          Then:
            - $.externalRouterId: $.netExplorer.getDefaultRouter()
        - If: $.autogenerateSubnet and (not bool($.subnetCidr))
          Then:
            - $.subnetCidr: $.netExplorer.getAvailableCidr($.externalRouterId, $.id())
        - $.createSubnet()
        - If: !yaql "bool($.externalRouterId)"
          Then:
            - $.createRouterInterface()

        - $.environment.stack.push()
        - $.setAttr(networkConfigured, true)


createNetwork:
  Body:
    - $template:
        Resources:
          $.net_res_name:
            Type: 'OS::Neutron::Net'
            Properties:
              name: $.name
    - $.environment.stack.updateTemplate($template)

createSubnet:
```

```
    Body:
      - $template:
          Resources:
            $.subnet_res_name:
              Type: 'OS::Neutron::Subnet'
              Properties:
                network_id: {Ref: $.net_res_name}
                ip_version: 4
                dns_nameservers: [$.dnsNameserver]
                cidr: $.subnetCidr
      - $.environment.stack.updateTemplate($template)

  createRouterInterface:
    Body:
      - $template:
          Resources:
            $.name + '-ri-' + $.id():
              Type: 'OS::Neutron::RouterInterface'
              Properties:
                router_id: $.externalRouterId
                subnet_id: {Ref: $.subnet_res_name}
      - $.environment.stack.updateTemplate($template)
```

1.6 Dynamic UI Definition specification

The main purpose of Dynamic UI is to generate application creation forms "on-the-fly". Murano dashboard doesn't know anything about what applications can be deployed and which web form are needed to create application instance. So all application definitions should contain a yaml file which tells dashboard how to create an application and what validations are to be applied. This document will help you to compose a valid UI definition for your application.

1.6.1 Structure

UI definition should be a valid yaml file and should contain the following sections (for version 2):

- **Version** - points out to which syntax version is used, optional
- **Templates** - optional, auxiliary section, used together with an Application section, optional
- **Application** - object model description which will be used for application deployment, required
- **Forms** - web form definitions, required

1.6.2 Version

Version of supported dynamic UI syntax. The latest version is 2. This is optional section, default version is set to 1. Version mapping: Murano 0.4 - version 1 Murano 0.5 - version 2

1.6.3 Application and Templates

In the Application section an *application object model* is described. This model will be translated into json and according to that json application will be deployed. Application section should contain all necessary keys that are required by murano-engine to deploy an application. Note that under *?* section goes system part of the model. You can pick parameters you got from the user (they should be described in the Forms section) and pick the right place where

they should be set. To do this YAQL is used. All lines are going to be checked for a yaql expressions. Currently, 2 yaql functions are provided for object model generation:

- **generateHostname** is used for machine hostname generation; it accepts 2 arguments: name pattern (string) and index (integer). If '#' symbol is present in name pattern, it will be replaced with the index provided. If pattern is not given, a random name will be generated.
- **repeat** is used to produce a list of data snippets, given the template snippet (first argument) and number of times it should be reproduced (second argument). Inside that template snippet current step can be referenced as *$index*.

Note that while evaluating YAQL expressions referenced from **Application** section (as well as almost all attributes inside **Forms** section, see later) *$* root object is set to the list of dictionaries with cleaned forms' data. So to obtain cleaned value of e.g. field *name* of form *appConfiguration* , you should reference it as *$.appConfiguration.name*. This context will be called as **standard context** throughout the text.

Example:

```
Templates:
  primaryController:
     ?:
       type: io.murano.windows.activeDirectory.PrimaryController
     host:
       ?:
         type: io.murano.windows.Host
       adminPassword: $.serviceConfiguration.adminPassword
       name: generateHostname($.serviceConfiguration.unitNamingPattern, 1)
       flavor: $.instanceConfiguration.flavor
       image: $.instanceConfiguration.osImage

   secondaryController:
     ?:
       type: io.murano.windows.activeDirectory.SecondaryController
     host:
       ?:
         type: io.murano.windows.Host
       adminPassword: $.serviceConfiguration.adminPassword
       name: generateHostname($.serviceConfiguration.unitNamingPattern, $index + 1)
       flavor: $.instanceConfiguration.flavor
       image: $.instanceConfiguration.osImage

Application:
  ?:
    type: io.murano.windows.activeDirectory.ActiveDirectory
  name: $.serviceConfiguration.name
  primaryController: $primaryController
  secondaryControllers: repeat($secondaryController, $.serviceConfiguration.dcInstances - 1)
```

1.6.4 Forms

This section describes markup elements for defining forms (which are currently rendered and validated with Django). Each form has name, field definitions (mandatory) and validator definitions (optionally). Note that each form is splitted into 2 parts - input area (left side, where all the controls are located) and description area (right side, where descriptions of the controls are located).

Each field should contain:

- **name** - system field name, could be any

- **type** - system field type

Currently supported options for **type** attribute are:

- string - text field (no inherent validations) with one-line text input
- boolean - boolean field, rendered as a checkbox
- text - same as string, but with a multi-line input
- integer - integer field with an appropriate validation, one-line text input
- password - text field with validation for strong password, rendered as two masked text inputs (second one is for password confirmation)
- clusterip - specific text field, used for entering cluster IP address (validations for valid IP address syntax and for that IP to belong to a fixed subnet)
- floatingip - specific boolean field, used for specifying whether or not an instance should have floating IP; *DEPRECATED FIELD* - use boolean field instead
- domain - specific field, used for selecting Active Directory domain from a list (or creating a new Active Directory application); *DEPRECATED FIELD* - use io.murano.windows.ActiveDirectory instead
- databaselist - Specific field, a list of databases (comma-separated list of databases' names, where each name has the following syntax first symbol should be latin letter or underscore; subsequent symbols can be latin letter, numeric, underscore, at the sign, number sign or dollar sign), rendered as one-line text input
- flavor - specific field, used for selection instance flavor from a list
- keypair - specific field, used for selecting keypair from a list
- image- specific field, used for selecting instance image from a list
- azone - specific field, used for selecting instance availability zone from a list
- any other value is considered to be a fully qualified name for some Application package and is rendered as a pair of controls: one for selecting already existing Applications of that type in an Environment, second - for creating a new Application of that type and selecting it

Other arguments (and whether they are required or not) depends on field's type and other attributes values. Among the most common attributes are:

- **label** - name, that will be displayed in the form; defaults to **name** being capitalized.
- **description** - description, that will be displayed in the description area. Use yaml line folding character >- to keep the correct formatting during data transferring.
- **descriptionTitle** - title of the description, defaults to **label**; displayed in the description area
- **hidden** whether field should be visible or not in the input area. Note that hidden field's description will still be visible in the descriptions area (if given). Hidden fields are used storing some data to be used by other, visible fields.
- **minLength**, **maxLength** (for string fields) and **minValue**, **maxValue** (for integer fields) are transparently translated into django validation properties.
- **validators** is a list of dictionaries, each dictionary should at least have *expr* key, under that key either some YAQL expression is stored, either one-element dictionary with *regexpValidator* key (and some regexp string as value). Another possible key of a validator dictionary is *message*, and although it is not required, it is highly desirable to specify it - otherwise, when validator fails (i.e. regexp doesn't match or YAQL expression evaluates to false) no message will be shown. Note that field-level validators use YAQL context different from all other attributes and section: here *$* root object is set to the value of field being validated (to make expressions shorter).

- **widgetMedia** sets some custom *CSS* and *JavaScript* used for the field's widget rendering. Note, that files should be placed to Django static folder in advance. Mostly they are used to do some client-side field enabling/disabling, hiding/unhiding etc. This is a temporary field which will be dropped once Version 3 of Dynamic UI is implemented (since it will transparently translate YAQL expressions into the appropriate *JavaScript*).
- **requirements** is used only with flavor field and prevents user to pick unstable for a deployment flavor. It allows to set minimum ram (in MBs), disk space (in GBs) or virtual CPU quantity.

  Example that shows how to hide items, smaller than regular 'small' flavor in flavor select field:

```
- name: flavor
       type: flavor
       label: Instance flavor
       requirements:
           min_disk: 20
           min_vcpus: 2
           min_memory_mb: 2048
```

Besides field-level validators form-level validators also exist. They use **standard context** for YAQL evaluation and are required when there is need to validate some form's constraint across several fields.

Example

```
Forms:
  - serviceConfiguration:
      fields:
        - name: name
          type: string
          label: Service Name
          description: >-
            To identify your service in logs please specify a service name
        - name: dcInstances
          type: integer
          hidden: true
          initial: 1
          required: false
          maxLength: 15
          helpText: Optional field for a machine hostname template
        - name: unitNamingPattern
          type: string
          label: Hostname template
          description: >-
            For your convenience all instance hostnames can be named
            in the same way. Enter a name and use # character for incrementation.
            For example, host# turns into host1, host2, etc. Please follow Windows
            hostname restrictions.
          required: false
          regexpValidator: '^(([a-zA-Z0-9#][a-zA-Z0-9-#]*[a-zA-Z0-9#])\.)*([A-Za-z0-9#]|[A-Za-z0-9#]
          # FIXME: does not work for # turning into 2-digit numbers
          maxLength: 15
          helpText: Optional field for a machine hostname template
          # temporaryHack
          widgetMedia:
            js: ['muranodashboard/js/support_placeholder.js']
            css: {all: ['muranodashboard/css/support_placeholder.css']}
      validators:
        # if unitNamingPattern is given and dcInstances > 1, then '#' should occur in unitNamingPatte
        - expr: $.serviceConfiguration.dcInstances < 2 or not $.serviceConfiguration.unitNamingPatter
          message: Incrementation symbol "#" is required in the Hostname template
  - instanceConfiguration:
```

```
fields:
  - name: title
    type: string
    required: false
    hidden: true
    descriptionTitle: Instance Configuration
    description: Specify some instance parameters on which service would be created.
  - name: flavor
    type: flavor
    label: Instance flavor
    description: >-
      Select registered in Openstack flavor. Consider that service performance
      depends on this parameter.
    required: false
  - name: osImage
    type: image
    imageType: windows
    label: Instance image
    description: >-
      Select valid image for a service. Image should already be prepared and
      registered in glance.
  - name: availabilityZone
    type: azone
    label: Availability zone
    description: Select availability zone where service would be installed.
    required: false
```

Full example with Active Directory application form definitions is available here *UI Definition Of AD App*

1.7 Murano workflow

What happens when a component is being created in an environment? This document will use the Telnet package referenced elsewhere as an example. It assumes the package has been previously uploaded to Murano.

1.7.1 Step 1. Begin deployment

The API sends a message that instructs murano-engine, the workflow component of Murano, to deploy an environment. The message consists of a JSON document containing the class types required to create the environment, as well as any parameters the user selected prior to deployment. Examples are:

- An *Class: Environment* object (io.murano.Environment) with a *name*
- An object (or objects) referring to networks that need to be created or that already exist
- A list of Applications (e.g. io.murano.apps.linux.Telnet). Each Application will contain, or will reference, anything it requires. The Telnet example, has a property called *instance* whose contract states it must be of type io.murano.resources.Instance. In turn the Instance has properties it requires (like a name, a flavor, a keypair name).

Each object in this *model* has an ID so that the state of each can be tracked.

The classes that are required are determined by the application's manifest. In the *Telnet example* only one class is explicitly required; the telnet application definition.

The *Telnet class definition* refers to several other classes. It extends *Class: Application* and it requires an *Class: Instance*. It also refers to the *Class: Environment* in which it will be contained, sends reports through the environment's

io.murano.system.StatusReporter and adds security group rules to the *Class: SecurityGroupManager*.

1.7.2 Step 2. Load definitions

The engine makes a series of requests to the API to download packages it needs. These requests pass the class names the environment will require, and during this stage the engine will validate that all the required classes exist and are accessible, and will begin creating them. All Classes whose *workflow* sections contain an *initialize* fragment are then initialized. A typical initialization order would be (defined by the ordering in the *model* sent to the murano-engine):

- *Class: Network*
- *Class: Instance*
- *Class: Object*
- *Class: Environment*

1.7.3 Step 3. Deploy resources

The workflow defined in Environment.deploy is now executed. The first step typically is to initialize the messaging component that will pay attention to murano-agent (see later). The next stage is to deploy each application the environment knows about in turn, by running deploy() for each application. This happens concurrently for all the applications belonging to an instance.

In the *Telnet example* (under *Workflow*), the workflow dictates sending a status message (via the environment's *reporter*, and configuring some security group rules. It is at this stage that the engine first contacts Heat to request information about any pre-existing resources (and there will be none for a fresh deploy) before updating the new Heat template with the security group information.

Next it instructs the engine to deploy the *instance* it relies on. A large part of the interaction with Heat is carried out at this stage; the first thing an Instance does is add itself to the environment's network. Since the network doesn't yet exist, murano-engine runs the neutron network workflow which pushes template fragments to Heat. These fragments can define: * Networks * Subnets * Router interfaces

Once this is done the Instance itself constructs a Heat template fragment and again pushes it to Heat. The Instance will include a *userdata* script that is run when the instance has started up, and which will configure and run murano-agent.

1.7.4 Step 4. Software configuration via murano-agent

If the workflow includes murano-agent components (and the telnet example does), typically the application workflow will execute them as the next step.

In the telnet example, the workflow instructs the engine to load *DeployTelnet.yaml* as YAML, and pass it to the murano-agent running on the configured instance. This causes the agent to execute the *EntryPoint* defined in the agent script (which in this case deploys some packages and sets some iptables rules).

1.7.5 Step 5. Done

After execution is finished, the engine sends a last message indicating that fact; the API receives it and marks the environment as deployed.

1.8 Murano Policy Enforcement

1.8.1 Murano Policy Enforcement Example

Introduction

As a part of the policy guided fulfillment, we need to enforce policies on the Murano environment deployment. If the policy enforcement failed, deployment fails. Policies are defined and evaluated in the Congress project. The policy language for Congress is Datalog. The congress policy consists of Datalog rules and facts. The cloud administrator defines policies in Congress. Examples of such policies:

- all VM instances must have at least 2GB of RAM
- all Apache server instances must have given certified version
- data placement policy: all DB instances must be deployed at given geo location (enforcing some law restriction on data placement)

These policies are evaluated over data in the form of tables (Congress data structures). A deployed Murano environment must be decomposed to Congress data structures. The decomposed environment is sent to congress for simulation. Congress simulates whether the resulting state does not violate any defined policy. Deployment is aborted in case of policy violation. Murano uses two predefined policies in Congress:

- *murano_system* contains rules and facts of policies defined by cloud admin.
- *murano* contains only facts/records reflecting resulting state after deployment of an environment.

Records in the *murano* policy are queried by rules from the *murano_system* policy. The congress simulation does not create any records in the *murano* policy. Congress will only give feedback on whether the resulting state violates the policy or not.

Example

In this example we will create rules that prohibit creating VM instances with flavor with more than 2048 MB ram.

Prior creating rules your OpenStack installation has to be configured as described in *Introduction*.

Example rules

1. Create `predeploy_errors` rule

   Policy validation engine checks rule `predeploy_errors` and rules referenced inside this rule are evaluated by congress engine.

   We create example rule which references `flavor_ram` rule we create afterwards. It disables flavors with ram higher than 2048 MB and constructs message returned to the user in *msg* variable.

   ```
   predeploy_errors(eid, obj_id, msg) :-
      murano:objects(obj_id, pid, type),
      murano:objects(eid, tid, "io.murano.Environment"),
      murano:connected(eid, pid),
      murano:properties(obj_id, "flavor", flavor_name),
      flavor_ram(flavor_name, ram),
      gt(ram, 2048),
      murano:properties(obj_id, "name", obj_name),
      concat(obj_name, ": instance flavor has RAM size over 2048MB", msg)
   ```

Use this command to create the rule:

```
congress policy rule create murano_system "predeploy_errors(eid, obj_id, msg) :- murano:obje
```

In this example we used data from policy **murano** which is represented by `murano:properties`. There are stored rows with decomposition of model representing murano application. We also used built-in functions of congress - `gt` - greater-than, and `concat` which joins two strings into variable.

2. Create `flavor_ram` rule

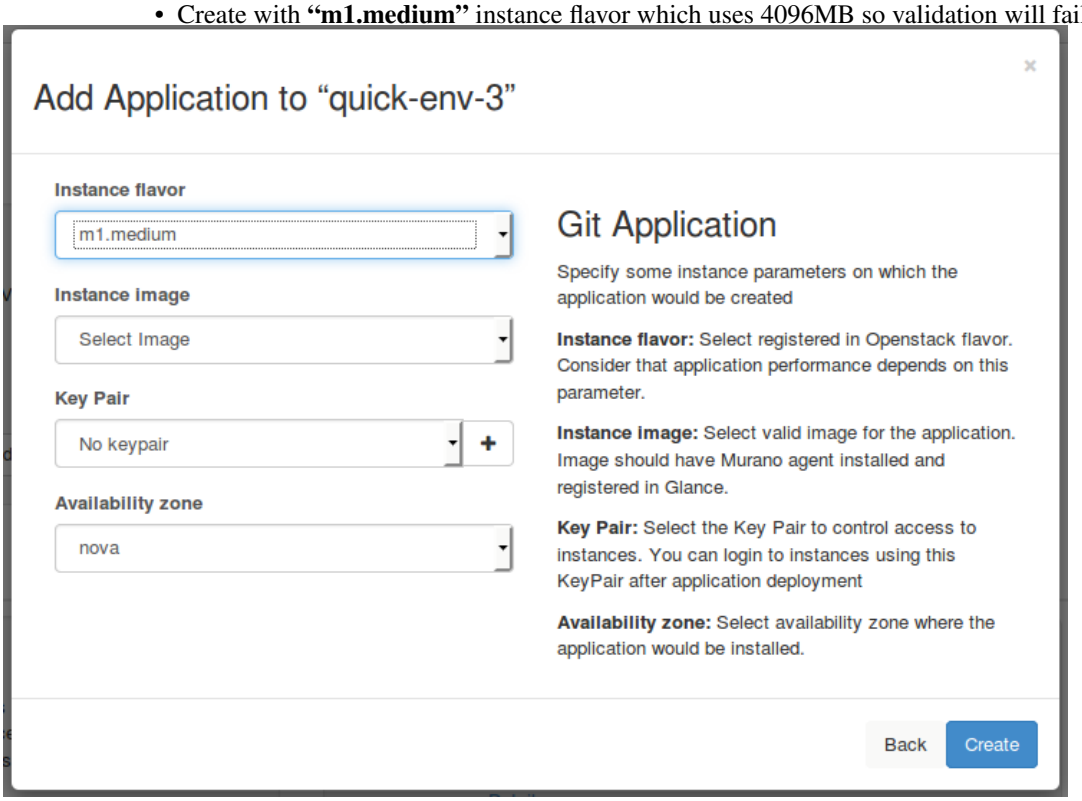
   We create the rule that resolves parameters of flavor by flavor name and returns *ram* parameter. It uses rule *flavors* from *nova* policy. Data in this policy is filled by *nova* datasource driver.

   Use this command to create the rule:

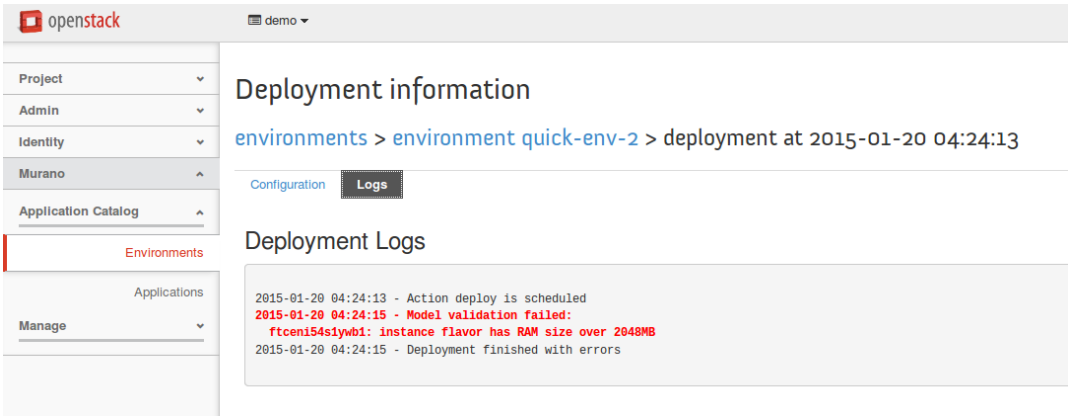
```
congress policy rule create murano_system "flavor_ram(flavor_name, ram) :- nova:flavors(id,
```

Example rules in murano app deployment

1. Create environment with simple application
   - Choose Git application from murano applications
   - Create with **"m1.medium"** instance flavor which uses 4096MB so validation will fail



2. Deploy environment
   - Environment is in Status: **Deploy FAILURE**
   - Check deployment log:



1.8.2 Murano Policy Enforcement Setup Guide

Introduction

Before policy enforcement feature will be used, it has to be configured. It has to be enabled in Murano configuration, and Congress has to have created policy and rules used during policy evaluation.

This document does not cover Murano and Congress configuration options useful for Murano application deployment (e.g., DNS setup, floating IPs, ...).

Setup

This setup uses *openstack* command. You can use copy-paste for commands.

If you are using DevStack installation, you can setup environment using following command.

```
source devstack/openrc admin admin
```

1. **Murano**

   Enable policy enforcement in Murano:

   - edit */etc/murano/murano.conf* to enable **enable_model_policy_enforcer** option:

   ```
   [engine]
   # Enable model policy enforcer using Congress (boolean value)
   enable_model_policy_enforcer = true
   ```

   - restart murano-engine

2. **Congress**

   Policy enforcement uses following policies:

   - **murano** policy

     Policy is created by Congress' Murano datasource driver, which is part of Congress. It has to be configured for the OpenStack tenant where Murano application will be deployed. Datasource driver retrieves deployed Murano environments and populates Congress' murano policy tables (*policyenf_dev*).

     Following commands removes existing **murano** policy, and creates new **murano** policy configured for tenant *demo*.

```
. ~/devstack/openrc admin admin # if you are using devstack, otherwise you have to setup env man

# remove default murano datasource configuration, because it is using 'admin' tenant. We need 'd
openstack congress datasource delete murano
openstack congress datasource create murano murano --config username="$OS_USERNAME" --config ten
```

- **murano_system policy** Policy holds user defined rules for policy enforcement. Rules typically uses tables from other policies (e.g., murano, nova, keystone, ...). Policy enforcement expects *predeploy_errors* table here which is created by creating **predeploy_errors** rules.

  Following command creates **murano_system** rule

```
openstack congress policy create murano_system
```

- **murano_action policy with internal management rules** Following rules are used internally in policy enforcement request. These rules are stored in dedicated **murano_action** policy which is created here. They are important for case when an environment is deployed again.

```
# create murano_action policy
openstack congress policy create murano_action --kind action

# register action deleteEnv
openstack congress policy rule create murano_action 'action("deleteEnv")'

# states
openstack congress policy rule create murano_action 'murano:states-(eid, st) :- deleteEnv(eid),

# parent_types
openstack congress policy rule create murano_action 'murano:parent_types-(tid, type) :- deleteEn
openstack congress policy rule create murano_action 'murano:parent_types-(eid, type) :- deleteEn

# properties
openstack congress policy rule create murano_action 'murano:properties-(oid, pn, pv) :- deleteEn
openstack congress policy rule create murano_action 'murano:properties-(eid, pn, pv) :- deleteEn

# objects
openstack congress policy rule create murano_action 'murano:objects-(oid, pid, ot) :- deleteEnv(
openstack congress policy rule create murano_action 'murano:objects-(eid, tnid, ot) :- deleteEnv

# relationships
openstack congress policy rule create murano_action 'murano:relationships-(sid, tid, rt) :- dele
openstack congress policy rule create murano_action 'murano:relationships-(eid, tid, rt) :- dele

# connected
openstack congress policy rule create murano_action 'murano:connected-(tid, tid2) :- deleteEnv(e
openstack congress policy rule create murano_action 'murano:connected-(eid, tid) :- deleteEnv(ei
```

1.8.3 Murano Policy Enforcement - Developer Guide

This document describes internals of murano policy enforcement.

Model Decomposition

Models of Murano applications are transformed to set of rules that are processed by congress. This represent data for policy validation.

There are several "tables" created in murano policy for different kind of rules:

- `murano:objects(object_id, parent_id, type_name)`
- `murano:properties(object_id, property_name, property_value)`
- `murano:relationships(source, target, name)`
- `murano:connected(source, target)`
- `murano:parent_types(object_id, parent_type_name)`
- `murano:states(environment_id, state)`

murano:objects(object_id, parent_id, type_name)

This rule is used for representation of all objects in Murano model (environment, applications, instances, ...). Value of property `type` is used as `type_name` parameter:

```
name: wordpress-env
'?': {type: io.murano.Environment, id: 83bff5ac}
applications:
- '?': {id: e7a13d3c, type: io.murano.databases.MySql}
```

Transformed to these rules:

- `murano:objects+("83bff5ac", "tenant_id", "io.murano.Environment")`
- `murano:objects+("83bff5ac", "e7a13d3c", "io.murano.databases.MySql")`

Note: The owner of the environment is a tenant

murano:properties(object_id, property_name, property_value)

Each object can have properties. In this example we have application with one property:

```
applications:
- '?': {id: e7a13d3c, type: io.murano.databases.MySql}
database: wordpress
```

Transformed to this rule:

- `murano:properties+("e7a13d3c", "database", "wordpress")`

Inner properties are also supported using dot notation:

```
instance:
'?': {id: 825dc61d, type: io.murano.resources.LinuxMuranoInstance}
networks:
  useFlatNetwork: false
```

Transformed to this rule:

- `murano:properties+("825dc61d", "networks.useFlatNetwork", "False")`

If model contains list of values it is represented as set of multiple rules:

```
instances:
- '?': {id: be3c5155, type: io.murano.resources.LinuxMuranoInstance}
networks:
  customNetworks: [10.0.1.0, 10.0.2.0]
```

Transformed to these rules:

- `murano:properties+("be3c5155", "networks.customNetworks", "10.0.1.0")`
- `murano:properties+("be3c5155", "networks.customNetworks", "10.0.2.0")`

murano:relationships(source, target, name)

Murano app models can contain references to other applications. In this example WordPress application references MySQL in property "database":

```
applications:
- '?':
    id: 0aafd67e
    type: io.murano.databases.MySql
- '?':
    id: 50fa68ff
    type: io.murano.apps.WordPress
  database: 0aafd67e
```

Transformed to this rule:

- `murano:relationships+("50fa68ff", "0aafd67e", "database")`

Note: For property "database" we do not create rule `murano:properties+`.

Also if we define inner object inside other object, they will have relationship between them:

```
applications:
- '?':
    id: 0aafd67e
    type: io.murano.databases.MySql
  instance:
    '?': {id: ed8df2b0, type: io.murano.resources.LinuxMuranoInstance}
```

Transformed to this rule:

- `murano:relationships+("0aafd67e", "ed8df2b0", "instance")`

There are special relationships "services" from the environment to its applications:

- `murano:relationships+("env_id", "app_id", "services")`

murano:connected(source, target)

This table stores both direct and indirect connections between instances. It is derived from the `murano:relationships`:

```
applications:
- '?':
    id: 0aafd67e
    type: io.murano.databases.MySql
  instance:
    '?': {id: ed8df2b0, type: io.murano.resources.LinuxMuranoInstance}
- '?':
    id: 50fa68ff
    type: io.murano.apps.WordPress
  database: 0aafd67e
```

Transformed to rules:

- `murano:connected+("50fa68ff", "0aafd67e")` # WordPress to MySql
- `murano:connected+("50fa68ff", "ed8df2b0")` # WordPress to LinuxMuranoInstance
- `murano:connected+("0aafd67e", "ed8df2b0")` # MySql to LinuxMuranoInstance

`murano:parent_types(object_id, parent_name)`

Each object in murano has class type and these classes can inherit from one or more parents:

e.g. `LinuxMuranoInstance` > `LinuxInstance` > `Instance`

So this model:

```
instances:
- '?': {id: be3c5155, type: LinuxMuranoInstance}
```

Transformed to these rules:

- `murano:objects+("...", "be3c5155", "LinuxMuranoInstance")`
- `murano:parent_types+("be3c5155", "LinuxMuranoInstance")`
- `murano:parent_types+("be3c5155", "LinuxInstance")`
- `murano:parent_types+("be3c5155", "Instance")`

Note: Type of object is also repeated among parent types (`LinuxMuranoInstance` in example) for easier handling of user-created rules.

Note: If type inherits from more than one parent and those parents inherit from one common type, `parent_type` rule is included only once for common type.

`murano:states(environment_id, state)`

Currently only one record for environment is created:

- `murano:states+("uugi324", "pending")`

Tutorials

1.9 Composing application package manual

Murano is Application catalog that supports types of applications. This document intends to make composing application packages easily.

1.9.1 Step 1. Prepare Execution Plans

An *Execution Plan* is a set of metadata that describes the installation process of an application in a virtual machine. It's a minimal unit of execution that can be triggered in Murano Workflows and should be understandable by Murano agent. From *Execution plans* any script can be triggered. It could be any type of scripts which will execute commands and install application components as the result. Each script may consist of one or more files. Scripts may be reused across several Execution Plans. One of the scripts should be an entry point and should be specified in a resource

template file in *Scripts*. Besides the *Scripts* section the following sections must be presented in a resource template file:

- **FormatVersion** - version of *Execution Plan* syntax format
- **Version** - version of *Execution Plan*
- **Name** - human-readable name of the Execution Plan
- **Parameters** - parameters received from MuranoPL
- **Body** - Python statement, should start with | symbol
- **Scripts** - dictionary that maps script names to script definitions.

  Scripts are the building blocks of Execution Plans and they may be executed as a whole (like a single piece of code), expose some functions that can be independently called in scripts. This depends on Deployment Platform and Executor capabilities. One script can be defined with the following properties

  - **Type** Deployment Platform name that script is targeted to.
  - **Version** optional minimum version of deployment platform/executor required by the script.
  - **EntryPoint** relative path to the file that contains a script entry point
  - **Files** This is an optional array of additional files required for the script. Use <> to specify a relative path to the file. The root directory is *Resource/scripts*.
  - **Options** an optional argument of type contains additional options

Example *DeployTelnet.template*

```
FormatVersion: 2.0.0
Version: 1.0.0
Name: Deploy Telnet

Parameters:
  appName: $appName

Body: |
  return deploy(args.appName).stdout

Scripts:
  deploy:
    Type: Application
    Version: 1.0.0
    EntryPoint: deployTelnet.sh
    Files:
      - installer.sh
      - common.sh
    Options:
      captureStdout: true
      captureStderr: false
```

1.9.2 Step 2. Prepare MuranoPL class definitions

MuranoPL classes control application deployment workflow execution. Full information about MuranoPL classes see here: *MuranoPL: Murano Programming Language* Example *telnet.yaml*

```
Namespaces:
  =: io.murano.apps.linux
  std: io.murano
  res: io.murano.resources


Name: Telnet

Extends: std:Application

Properties:
  name:
    Contract: $.string().notNull()

  instance:
    Contract: $.class(res:Instance).notNull()


Workflow:
  deploy:
    Body:
      - $this.find(std:Environment).reporter.report($this, 'Creating VM for Telnet instace.')
      - $.instance.deploy()
      - $this.find(std:Environment).reporter.report($this, 'Instance is created. Setup Telnet service
      - $resources: new('io.murano.system.Resources')
      # Deploy Telnet
      - $template: $resources.yaml('DeployTelnet.template')
      - $.instance.agent.call($template, $resources)
      - $this.find(std:Environment).reporter.report($this, 'Telnet service setup is done.')
```

Note, that

- *io.murano.system.Resources* is a system class, defined in MuranoPL. More information about MuranoPL system classes is available here: *Murano PL System Class Definitions*.
- *io.murano.resources.Instance* is a class, defined in the core Murano library, which is available here. *This library* contains Murano Agent templates and virtual machine initialization scripts.
- $this.find(std:Environment).reporter.report($this, 'Creating VM for Telnet instance.') - this is the way of sending reports to Murano dashboard during deployment

1.9.3 Step 3. Prepare dynamic UI form definition

Create a form definition in a yaml format. Before configuring a form, compose a list of parameters that will be required to set by a user. Some form fields that are responsible for choosing a flavor, image and availability zone are better to use in every application creation wizard. Syntax of Dynamic UI can be found see at the corresponding section: *Dynamic UI Definition specification*. Full example with Telnet application form definition *Telnet Definition*.

1.9.4 Step 4. Prepare application logo

Find or create a simple image (in a .png format) associated with your application. Is should be small and have a square shape. You can specify any name of your image. In our example, let's name it *telnet.png*.

1.9.5 Step 5. Prepare manifest file

General application metadata should be described in the application manifest file. It should be in a yaml format and should have the following sections

- **Format** - version of a manifest syntax format
- **Type** - package type. Valid choices are *Library* and *Application*
- **Name** - human-readable application name
- **Description** - a brief description of an application
- **Author** - person or company name which created an application package
- **Classes** - MuranoPL class list, on which application deployment is based
- **Tags** - list of words, associated with this application. Will be helpful during the search. *Optional* parameter
- **Require** - list of applications with versions, required by this application. Currently only used by repository importing mechanism. *Optional* parameter

Example *manifest.yaml*

```
Format: 1.0
Type: Application
FullName: io.murano.apps.linux.Telnet
Name: Telnet
Description: |
 Telnet is the traditional protocol for making remote console connections over TCP.
Author: 'Mirantis, Inc'
Tags: [Linux, connection]
Classes:
 io.murano.apps.linux.Telnet: telnet.yaml
UI: telnet.yaml
Logo: telnet.png
Require:
  io.murano.apps.TelnetHelper: 0.0.1
```

1.9.6 Step 6. Prepare images.lst file

This step is optional. If you plan on providing images required by your application, you can include `images.lst` file with image specifications

Example *images.lst*

```
Images:
- Name: 'my_image.qcow2'
  Hash: '64d7c1cd2b6f60c92c14662941cb7913'
  Meta:
    title: 'tef'
    type: 'linux'
  DiskFormat: qcow2
  ContainerFormat: bare
- Name: 'my_other_image.qcow2'
  Hash: '64d7c1cd2b6f60c92c14662941cb7913'
  Meta:
    title: 'tef'
    type: 'linux'
  DiskFormat: qcow2
```

```
ContainerFormat: bare
Url: 'http://path.to/images/file.qcow2'
```

If *Url* is omitted - the images would be searched for in the Murano Repository.

1.9.7 Step 7. Compose a zip archive

An application archive should have the following structure

- ***Classes* folder** MuranoPL class definitions should be put inside this folder
- ***Resources* folder** This folder should contain Execution scripts
  - ***Scripts* folder** All script files, needed for an application deployment should be placed here
- ***UI* folder** Place dynamic ui yaml definitions here or skip to use the default name *ui.yaml*
- ***logo.png*** Image file should be placed in the root folder. It can have any name, just specify it in the manifest file or skip to use default *logo.png* name
- ***manifest.yaml*** Application manifest file. It's an application entry point. The file name is fixed.
- ***images.lst*** List of required images. Optional file.

Congratulations! Your application is ready to be uploaded to an Application Catalog.

1.10 Uploading HOT templates to the Application Catalog

Murano is an Application catalog which intends to support applications, defined in different formats. As a first step to universality, heat orchestration template support was added. It means that any heat template could be added as a separate application into the Application Catalog. This could be done in two ways: manual and automatic.

1.10.1 Automatic package composing

Before uploading an application into the catalog, it should be prepared and archived. Murano command line will do all preparation for you. Just choose the desired Heat Orchestration Template and perform the following command:

```
murano package-create  --template wordpress/template.yaml
```

Note, that optional parameters could be specified:

–name Application name, copied from template by default

–logo Application square logo, by default heat logo will be used

–description Text information about an application, by default copied from template

–author Name of application author, by default is set to

–output Name of the output file archive to save locally

–full-name Fully qualified domain name - domain name that specifies exact application location

---

Note: To performing this command python-muranoclient should be installed in the system

---

As the result, application definition archive will be ready for an uploading.

1.10.2 Manual package composing

Application package could be composed manually. Follow the 5 steps below.

- *Step 1. Choose the desired heat orchestration template*
- *Step 2. Rename it to template.yaml*
- *Step 3. Prepare application logo (optional step)*

  It could be any picture associated with the application.

- *Step 4. Create manifest.yaml file*

  All service information about the application is contained here. Specify the following parameters:

  Format Defines application definition format; should be set to `Heat.HOT/1.0`

  Type Defines manifest type, should be set to `Application`

  FullName Unique name which will be used to identify the application in Murano Catalog

  Description Text information about an application

  Author Name of application author or company

  Tags Keywords, associated with the application

  Logo Name of the application logo file

Take a look at the example:

```
Format: Heat.HOT/1.0
Type: Application
FullName: io.murano.apps.linux.Wordpress
Name: Wordpress
Description: |
 WordPress is web software you can use to create a beautiful website or blog.
  This template installs a single-instance WordPress deployment using a local
  MySQL database to store the data.
Author: 'Openstack, Inc'
Tags: [Linux, connection]
Logo: logo.png
```

- *Step 5. Create a zip archive, containing specified files(* `template.yaml`, `manifest.yaml`, `logo.png`)

1.10.3 Package uploading

After application package is ready, it can be uploaded to the catalog in two ways:

- Using murano CLI

  During uploading, it's required to provide category, that application belongs to. To browse all available categories preform:

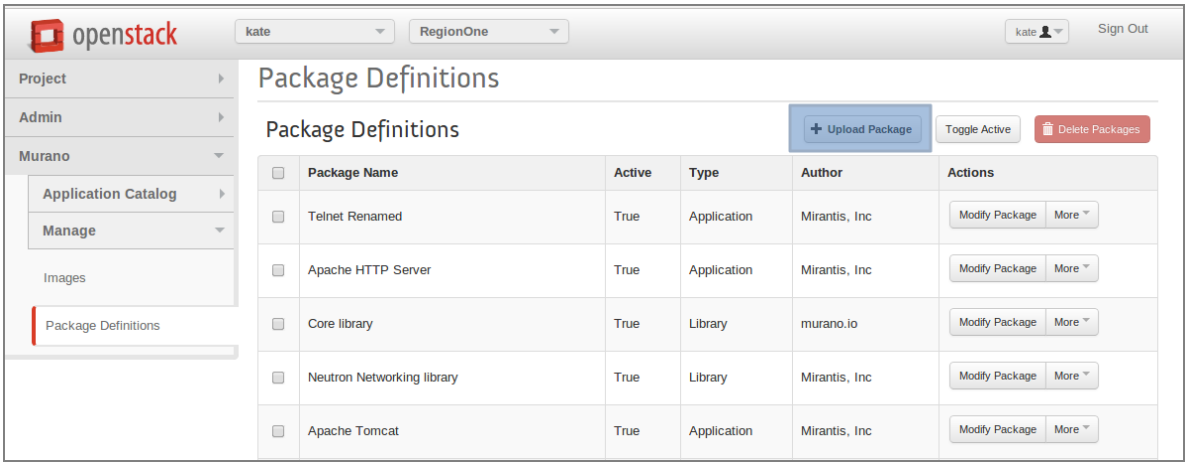
  ```
  murano category-list
  ```

  Specify any suitable category and path to the application archive.

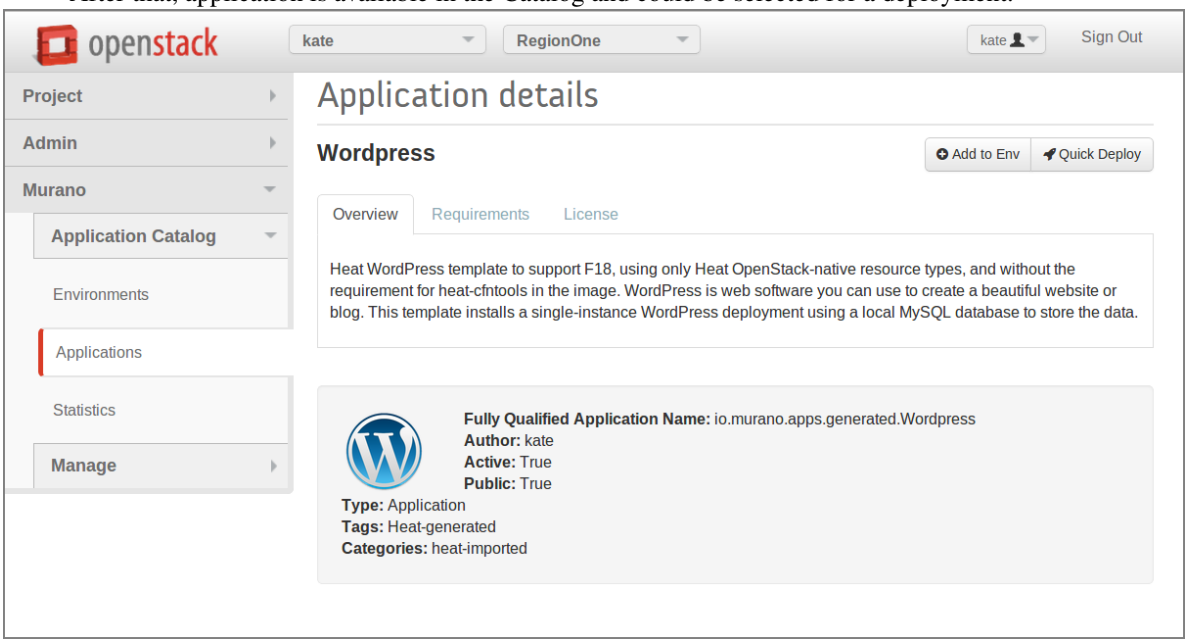
  ```
  murano package-import --category=Web wordpress.zip
  ```

- Using Murano Dashboard

  Package uploading is available for admin users at Murano -> Manage -> Packages page.



After that, application is available in the Catalog and could be selected for a deployment.



1.11 Building Murano Image

1.11.1 Windows Image

Murano requires a Windows Image in QCOW2 format to be built and uploaded into Glance.

The easiest way to build Windows image for Murano is to build it on the host where your OpenStack is installed.

Prepare Image Builder Host

Install KVM

Note: This guide was tested on Ubuntu Server 12.04 x64.

KVM is a default hypervisor in OpenStack, so our build scripts are targeted to this hypervisor only. It may change in future, though.

Install KVM and some additional packages that are required by our scripts

```
# apt-get install qemu-kvm virtinst virt-manager
```

Check that your hardware supports hardware virtualization.

```
$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

If your output differs, check that harware virtualization is enabled in your BIOS settings. You also could try import KVM kernel module

```
$ sudo modprobe kvm-intel
```

or

```
$ sudo modprobe kvm-amd
```

It might be helpful to add an appropriate module name into **/etc/modules** file to auto-load it during system boot. Sometimes it is required on Ubuntu systems.

Configure Shared Resource

Murano Image Builder uses a shared folder located on the host system as an installation source for components. Makefile from image builder will copy required files to their locations, but you have to manually configure samba share. To do this, use the steps below.

- Install samba

  ```
  # apt-get install samba
  ```

- Create folder that will be shared

  ```
  # mkdir -p /opt/samba/share
  # chown -R nobody:nogroup /opt/samba/share
  ```

- Configure samba server (/etc/samba/smb.conf)

```
...
[global]
    ...
  security = share
  ...
[image-builder-share]
     comment = Image Builder Share
     browsable = yes
     path = /opt/image-builder/share
     guest ok = yes
     guest user = nobody
     read only = no
     create mask = 0755
...
```

- Restart samba services

```
# service smbd restart
# service nmbd restart
```

Download Prerequisites

Windows Server Installation ISO

Windows Version	Version String	Save to
Windows Server 2008 R2 [1]	6.1.7601	/opt/image-builder/share/libvirt/images/ws-2008-eval.iso
Windows Server 2012 [2]	6.3.9200	/opt/image-builder/share/libvirt/images/ws-2012-eval.iso

Warning: Windows Server 2008 R2 must include Service Pack 1 updates. This is required to install PowerShell V3 which is required by Murano Agent.

Required Components

Component	Save to
VirtIO drivers for Windows [3]	/opt/image-builder/share/libvirt/images/virtio-win-0.1-74.iso
CloudBase-Init for Windows [4]	/opt/image-builder/share/files/CloudbaseInitSetup_Beta.msi
.NET 4.0 [5]	/opt/image-builder/share/files/dotNetFx40_Full_x86_x64.exe
PowerShell v3 [6]	/opt/image-builder/share/files/Windows6.1-KB2506143-x64.msu
Murano Agent [7]	/opt/image-builder/share/files/MuranoAgent.zip
Git client [8]	/opt/image-builder/share/files/Git-1.8.1.2-preview20130601.exe

Warning: PowerShell V3 is a **mandatory** prerequisite. It is required by Murano Agent. To check your PowerShell version use PowerShell command *Get-Host*.

[1] http://www.microsoft.com/en-us/download/details.aspx?id=11093
[2] http://technet.microsoft.com/en-US/evalcenter/hh670538.aspx?ocid=&wt.mc_id=TEC_108_1_33
[3] http://alt.fedoraproject.org/pub/alt/virtio-win/stable/virtio-win-0.1-74.iso
[4] https://www.cloudbase.it/downloads/CloudbaseInitSetup_Beta.msi
[5] http://www.microsoft.com/en-us/download/details.aspx?id=17718
[6] http://www.microsoft.com/en-us/download/details.aspx?id=34595
[7] https://www.dropbox.com/sh/zthldcxnp6r4flm/AADh6LkVkcw2j8nKZevqedHja/MuranoAgent.zip
[8] https://msysgit.googlecode.com/files/Git-1.8.3-preview20130601.exe

Warning: When downloading VirtIO drivers choose only stable versions. Unstable versions might lead to errors during guest unattended installation. You can check the latest version avaible here: http://alt.fedoraproject.org/pub/alt/virtio-win/stable

Optional Components

These components are not mandatory for Murano Agent to function properly. However, they may help you work with the image after deployment.

Component	Save to
Far Manager [9]	/opt/image-builder/share/files/Far30b3367.x64.20130717.msi
Sysinternals Suite [10]	/opt/image-builder/share/files/SysinternalsSuite.zip
unzip.exe [11]	/opt/image-builder/share/files/unzip.exe
.NET 4.5 [12]	/opt/image-builder/share/files/dotNetFx45_Full_setup.exe

Additional Tools

Tools from this section are not necessary to build an image. However, they may be helpful if you want to create an image with different configuration.

Windows Assessment and Deployment Kit (ADK)

Windows ADK is required if you want to build your own answer files for auto unattended Windows installation.

Download it from http://www.microsoft.com/en-us/download/details.aspx?id=30652

Floppy Image With Unattended File

Floppy image with answer file for unattended installation is needed to automate Windows installation process.

- Create emtpy floppy image in your home folder

```
$ mkdir ~/flp/files
$ mkdir ~/flp/mnt

$ dd bs=512 count=2880 if=/dev/zero of=~/flp/floppy.img
$ mkfs.msdos ~/flp/floppy.img
```

[9] http://www.farmanager.com/files/Far30b3525.x64.20130717.msi
[10] http://download.sysinternals.com/files/SysinternalsSuite.zip
[11] https://www.dropbox.com/sh/zthldcxnp6r4flm/AACwiyfcrlGDt3ygCFHrbwMra/unzip.exe
[12] http://www.microsoft.com/en-us/download/details.aspx?id=30653

- Mount the image

```
$ mkdir ~/flp/mnt
$ sudo mount -o loop ~/floppy.img ~/flp/mnt
```

- Download **autounattend.xml.template** file from https://github.com/openstack/murano-deployment/tree/master/contrib/windows/image-builder/share/files

  This folder contains unatteded files for several Windows versions, choose one that matches your Windows version.

- Copy that file to mounted floppy image

```
$ cp ~/autounattend.xml.template ~/flp/mnt/autounattend.xml
```

- Replace string **%_IMAGE_BUILDER_IP_%** in that file with **192.168.122.1**
- Unmount the image

```
$ sudo umount ~/flp/mnt
```

Build Windows Image with Murano

Build Windows Image Using Image Builder Script

- Clone **murano-deployment** repository

```
$ git clone git://git.openstack.org/cgit/openstack/murano-deployment.git
```

- Change directory to image-builder folder

```
$ cd murano-deployment/contrib/windows/image-builder
```

- Create folder structure for image builder

```
$ sudo make build-root
```

- Download build prerequisites, and copy them to correct folders
  - *Windows Server Installation ISO*
  - *Required Components*
  - *Optional Components* (Optional)
- Test that all required files are in place

```
$ sudo make test-build-files
```

- Get list of available images

```
$ make
```

- Run image build process (e.g. to build Windows Server 2012)

```
$ sudo make ws-2012-std
```

- Wait until process finishes
- The image file **ws-2012-std.qcow2** should be stored inside **/opt/image-builder/share/images** folder.

Build Windows Image Manualy

Note: Please note that the preferred way to build images is to use Image Builder scripts, see *Build Windows Image Using Image Builder Script*

Get Post-Install Scripts

There are a few scripts which perform all the required post-installation tasks.

They all are located in http://git.openstack.org/cgit/openstack/murano-deployment/tree/contrib/windows/image-builder/share/scripts

Note: There are subfolders for each supported Windows Version. Choose one that matches Windows Version you are building.

This folder contains several scripts

Script Name	Description
wpi.ps1	Handles component installation and system configuration tasks
Start-Sysprep.ps1	Prepares system to be syspreped (cleans log files, stops some services and so on), and starts sysprep
Start-AtFirstBoot.ps1	Performes basic after-installation tasks

Download these scripts and save them into /opt/image-builder/share/scripts

Create a VM

Now you need a virtual machine instance. There are two possible ways to create it - using CLI or GUI tools. We describe both in this section.

Using CLI Tools

1. Preallocate disk image

```
$ qemu-img create -f raw /var/lib/libvirt/images/ws-2012.img 40G
```

2. Start the VM

```
# virt-install --connect qemu:///system --hvm --name WinServ \
> --ram 2048 --vcpus 2 --cdrom /opt/samba/share/9200.16384.WIN8_RTM\
>.120725-1247_X64FRE_SERVER_EVAL_EN-US-HRM_SSS_X64FREE_EN-US_DV5.ISO \
> --disk path=/opt/samba/share/virtio-win-0.1-52.iso,device=cdrom \
> --disk path=/opt/samba/share/floppy.img,device=floppy \
> --disk path=/var/lib/libvirt/images/ws-2012.qcow2\
>,format=qcow2,bus=virtio,cache=none \
> --network network=default,model=virtio \
> --memballoon model=virtio --vnc --os-type=windows \
> --os-variant=win2k8 --noautoconsole \
> --accelerate --noapic --keymap=en-us --video=cirrus --force
```

Using virt-manager UI

1. Launch *virt-manager* from shell as root
2. Set a name for VM and select Local install media
3. Add one cdrom and attach Windows Server ISO image to it
4. Select OS type **Windows**
5. Set CPU and RAM amount
6. Deselect option **Enable storage for this virtual machine**
7. Add second cdrom for ISO image with virtio drivers
8. Add a floppy drive and attach our floppy image to it
9. Add (or create new) HDD image with Disk bus **VirtIO** and storage format **RAW**
10. Set network device model **VirtIO**
11. Start installation process and open guest vm screen through **Console** button

Install OS

Launch your virtual machine, connect to its virtual console and complete OS installation. At the end of this step you should have Windows Server system that you are able to log into.

Install Prerequisites and Murano

- Create folders where Murano components will be installed

Path	Description
C:\Murano	Root directory for Murano
C:\Murano\Agent	Murano Agent installation directory
C:\Murano\Modules	PowerShell modules required by Murano
C:\Murano\Scripts	PowerShell scrtips and other files required by Murano

- Open **Explorer** and navigate to **\\192.168.122.1share 192.168.122.1** is an IP address of KVM hypervisor assigned by default.
- Copy Murano Agent files into C:MuranoAgent
- Copy CoreFunctions directory (entire directory!) into C:MuranoModules
- Install .NET 4.0
- Register Murano Agent

```
> cd C:\Murano\Agent
> .\WindowsMuranoAgent.exe /install
```

- Change PowerShell execution policy to less restricted

```
Set-ExecutionPolicy RemoteSigned
```

- Register CoreFunctions modules

```
Import-Module C:\Murano\Modules\CoreFunctions\CoreFunctions.psm1 -ArgumentList register
```

- Install CloudInit

- Run Sysprep

  ```
  C:\Murano\Scripts\Start-Sysprep.ps1 -BatchExecution
  ```

- Wait until sysprep phase finishes and system powers off.

Convert the image from RAW to QCOW2 format

The image must be converted from RAW format to QCOW2 before being imported into Glance.

```
# qemu-img convert -O qcow2 /var/lib/libvirt/images/ws-2012.raw \
> /var/lib/libvirt/images/ws-2012-ref.qcow2
```

1.11.2 Linux Image

At the moment the best way to build a Linux image with Murano agent is to use disk image builder.

Note: Disk image builder requires sudo rights

The process is quite simple. Let's assume that you use a directory ~/git for cloning git repositories:

```
$ export GITDIR=~/git
$ mkdir -p $GITDIR
```

Clone the components required to build an image to that directory:

```
$ cd $GITDIR
$ git clone git://git.openstack.org/cgit/openstack/murano
$ git clone git://git.openstack.org/cgit/openstack/murano-agent
$ git clone git://git.openstack.org/cgit/openstack/diskimage-builder
```

Checkout a change request that allows to build an image using disk image builder completely installed to virtual environment:

```
$ cd $GITDIR/diskimage-builder
$ git fetch https://review.openstack.org/openstack/diskimage-builder refs/changes/02/168002/2 && git
```

Install additional packages required by disk image builder:

```
$ sudo apt-get install qemu-utils curl python-tox
```

Export paths where additional dib elements are located:

```
$ export ELEMENTS_PATH=$GITDIR/murano/contrib/elements:$GITDIR/murano-agent/contrib/elements
```

And build Ubuntu-based image with Murano agent:

```
$ cd $GITDIR/diskimage-builder
$ tox -e venv -- disk-image-create vm ubuntu murano-agent -o ../murano-agent.qcow2
```

If you need Fedora based image replace 'ubuntu' to 'fedora' in the last command.

It'll take a while (up to 30 minutes if your hard drive and internet connection are slow).

When done upload murano-agent.qcow2 image to Glance and play :)

1.11.3 Upload Image Into Glance

To deploy applications with Murano, virtual machine images should be uploaded into Glance in a special way - *murano_image_info* property should be set.

1. Use the glance image-create command to import your disk image to Glance:

```
$ glance image-create --name <NAME>  --is-public true \
> --disk-format qcow2 --container-format bare \
> --file <IMAGE_FILE> --property <IMAGE_METADATA>
```

Replace the command line arguments to glance image-create with the appropriate values for your environment and disk image:

- Replace **<NAME>** with the name that users will refer to the disk image by. E.g. **ws-2012-std**
- Replace **<IMAGE_FILE>** with the local path to the image file to upload. E.g. **ws-2012-std.qcow2**.
- Replace **<IMAGE_METADATA>** with the following property string

```
murano_image_info='{"title": "Windows 2012 Standart Edition", "type": "windows.2012"}'
```

where:

- **title** - user-friendly description of the image
- **type** - murano image type, see *Murano Image Types*

2. To update metadata of the existing image run the command:

```
$ glance image-update <IMAGE_ID> --property <IMAGE_MATADATA>
```

- Replace **<IMAGE_ID>** with image id from the previous command output.
- Replace **<IMAGE_METADATA>** with murano_image_info property, e.g.

```
murano_image_info='{"title": "Windows 2012 Standart Edition", "type": "windows.2012"}'
```

> **Warning:** The value of the **–property** argument (named **murano_image_info**) is a JSON string. Only double quotes are valid in JSON, so please type the string exactly as in the example above.

Note: Already existing image could be marked in a simple way in Horizon UI with Murano dashboard installed. Navigate to *Murano -> Manage -> Images -> Mark Image* and fill up a form:

- **Image** - ws-2012-std
- **Title** - My Prepared Image
- **Type** - Windows Server 2012

After these steps desired image can be chosen in application creation wizard.

Murano Image Types

Type Name	Description
windows.2012	Windows Server 2012
linux	Generic Linux images, Ubuntu / Debian, RedHat / Centos, etc
cirrus.demo	Murano demo image, based on CirrOS

1.12 Murano Automated Tests Description

This page describes automated tests for a Murano project:

- where tests are located
- how they are run
- how execute tests on a local machine
- how to find the the root of problems with FAILed tests

1.12.1 Murano Continuous Integration Service

Murano project has separate CI server, which runs tests for all commits and verifies that new code does not break anything.

Murano CI uses OpenStack QA cloud for testing infrastructure.

Murano CI url: https://murano-ci.mirantis.com/jenkins/ Anyone can login to that server, using launchpad credentials.

There you can find each job for each repository: one for the **murano** and another one for **murano-dashboard**.

- "gate-murano-dashboard-selenium*" verifies each commit to murano-dashboard repository
- "gate-murano-integration*" verifies each commit to murano repository

Other jobs allow to build and test Murano documentation and perform another usefull work to support Murano CI infrastructure. All jobs are run on fresh installation of operation system and all components are installed on each run.

1.12.2 Murano Automated Tests: UI Tests

Murano project has a Web User Interface and all possible user scenarios should be tested. All UI tests are located at the https://git.openstack.org/cgit/openstack/murano-dashboard/tree/muranodashboard/tests/functional

Automated tests for Murano Web UI are written in Python using special Selenium library. This library is used to automate web browser interaction from Python. For more information please visit https://selenium-python.readthedocs.org/

Prerequisites:

- Install Python module, called nose performing one of the following commands **easy_install nose** or **pip install nose** This will install the nose libraries, as well as the nosetests script, which you can use to automatically discover and run tests.
- Install external Python libraries, which are required for Murano Web UI tests: **testtools** and **selenium**

Download and run tests:

First of all make sure that all additional components are installed.

- Clone murano-dashboard git repository:
  - git clone git://git.openstack.org/openstack/murano-dashboard*
- Change default settings:
  - Copy muranodashboard/tests/functional/config/config.conf.example to config.conf

- Set appropriate urls and credentials for your OpenStack lab. Only admin users are appropriate.

```
[murano]

horizon_url = http://localhost/horizon
murano_url = http://localhost:8082
user = ***
password = ***
tenant = ***
keystone_url = http://localhost:5000/v2.0/
```

All tests are kept in *sanity_check.py* and divided into 5 test suites:

- TestSuiteSmoke - verification of Murano panels; check, that could be open without errors.
- TestSuiteEnvironment - verification of all operations with environment are finished successfully.
- TestSuiteImage - verification of operations with images.
- TestSuiteFields - verification of custom fields validators.
- TestSuitePackages - verification of operations with Murano packages.
- TestSuiteApplications - verification of Application Catalog page and of application creation process.

To specify which tests/suite to run, pass test/suite names on the command line:

- to run all tests: `nosetests sanity_check.p`
- to run a single suite: `nosetests sanity_check.py:<test suite name>`
- to run a single test: `nosetests sanity_check.py:<test suite name>.<test name>`

In case of SUCCESS execution, you should see something like this:

```
.........................

Ran 34 tests in 1.440s

OK
```

In case of FAILURE, folder with screenshots of the last operation of tests that finished with errors would be created. It's located in *muranodashboard/tests/functional* folder.

There are also a number of command line options that can be used to control the test execution and generated outputs. For more details about *nosetests*, try:

```
$ nosetests -h
```

1.12.3 Murano Automated Tests: Tempest Tests

All Murano services have tempest-based automated tests, which allow to verify API interfaces and deployment scenarious.

Tempest tests for Murano are located at the: https://git.openstack.org/cgit/openstack/murano/tree/murano/tests/functional

The following Python files are contain basic tests suites for different Murano components.

API Tests

Murano API tests are run on devstack gate and located at https://git.openstack.org/cgit/openstack/murano/tree/murano/tests/functional/ap

- *test_murano_envs.py* contains test suite with actions on murano's environments(create, delete, get and etc.)

- *test_murano_sessions.py* contains test suite with actions on murano's sessions(create, delete, get and etc.)
- *test_murano_services.py* contains test suite with actions on murano's services(create, delete, get and etc.)
- *test_murano_repository.py* contains test suite with actions on murano's package repository

Engine Tests

Murano Engine Tests are run on murano-ci : https://git.openstack.org/cgit/openstack/murano/tree/murano/tests/functional/engine

- *base.py* contains base test class and tests with actions on deploy Murano services such as 'Telnet' and 'Apache'.

Command Line Tests

Murano CLI tests case are currently in the middle of creation. The current scope is read only operations on a cloud that are hard to test via unit tests.

All tests have description and execution steps in there docstrings.

Client

1.13 Murano client

Module python-muranoclient comes with CLI *murano* utility, that interacts with Murano application catalog

1.13.1 Installation

To install latest murano CLI client run the following command in your shell:

```
pip install python-muranoclient
```

Alternatively you can checkout the latest version from https://git.openstack.org/cgit/openstack/python-muranoclient

1.13.2 Using CLI client

In order to use the CLI, you must provide your OpenStack username, password, tenant name or id, and auth endpoint. Use the corresponding arguments (`--os-username`, `--os-password`, `--os-tenant-name` or `--os-tenant-id`, `--os-auth-url` and `--murano-url`) or set corresponding environment variables:

```
export OS_USERNAME=user
export OS_PASSWORD=password
export OS_TENANT_NAME=tenant
export OS_AUTH_URL=http://auth.example.com:5000/v2.0
export MURANO_URL=http://murano.example.com:8082/
```

Once you've configured your authentication parameters, you can run `murano help` to see a complete listing of available commands and arguments and `murano help <sub_command>` to get help on specific subcommand.

1.13.3 Bash completion

To get the latest bash completion script download murano.bash_completion from the source repository and add it to your completion scripts.

1.13.4 Listing currently installed packages

To get list of currently installed packages run:

```
murano package-list
```

To show details about specific package run:

```
murano package-show <PKG_ID>
```

1.13.5 Importing packages in Murano

`package-import` subcommand can install packages in several different ways:

- from a locall file
- from a http url
- from murano app repository

When creating a package you can specify it's categories with `-c/--categories` and set it's publicity with `--public`

To import a local package run:

```
murano package-import /path/to/package.zip
```

To import a package from http url run:

```
murano package-import http://example.com/path/to/package.zip
```

And finally you can import a package from Murano repository. To do so you have to specify base url for the repository with `--murano-repo-url` or with the corresponding `MURANO_REPO_URL` environment variable. After doing so, running:

```
murano --murano-repo-url="http://example.com/" package-import io.app.foo
```

would access specified repository and download app `io.app.foo` from it's app directory. This option supports an optional `--version` parameter, that would instruct murano client to download package of a specific version.

`package-import` inspects package requirements specified in the package's manifest under *Require* section and attempts to import them from Murano Repository. `package-import` also inspects any image prerequisites, mentioned in the *images.lst* file in the package. If there are any image requirements client would inspect images already present in the image database. Unless image with the specific name and hash is present client would attempt to download it.

For more info about specifiying images and requirements for the package see package creation docs: *Composing application package manual*.

If any of the packages, being installed is already registerd in Murano, client would ask you what do do with it. You can specify the default action with `--exists-action`, passing *s* for skip, *u* for update, and *a* for abort.

1.13.6 Importing bundles of packages in Murano

`package-import` subcommand can install packages in several different ways:

- from a locall file
- from a http url

- from murano app repository

When creating a package you can specify it's categories with `-c/--categories` and set it's publicity with `--public`

To import a local bundle run:

```
murano bundle-import /path/to/bundle
```

To import a bundle from http url run:

```
murano bundle-import http://example.com/path/to/bundle
```

To import a bundle from murano repository run:

```
murano bundle-import bundle_name
```

Note: When importing from a local file packages would first be searched in a directory, relative to the directory containing the bundle file itself. This is done to facilitate installing bundles in an environment with no access to the repository itself.

1.13.7 Deleting packages from murano

To delete a package run:

```
murano package-delete <PKG_ID>
```

1.13.8 Downloading package file

Running:

```
murano package-download <PKG_ID> > file.zip
```

would download the zip arhive with specified package

1.13.9 Creating a package

Murano client is able to create application packages from package source files/directories. To find out more about this command run:

```
murano help package-create
```

This command is useful, when application package files are spread across several directories, and for auto-generating packages from heat templates For more info about package composition please see package creation docs: *Composing application package manual*.

1.13.10 Managing Environments

It is possible to create/update/delete environments with following commands:

```
murano environment-create <NAME>
murano environment-delete <NAME_OR_ID>
murano environment-list
murano environment-rename <OLD_NAME_OR_ID> <NEW_NAME>
murano environment-show <NAME_OR_ID>
```

You can get list of deployments for environmet with:

```
murano deployment-list <NAME_OR_ID>
```

1.13.11 Managing Categories

It is possible to create/update/delete categories with following commands:

```
murano category-create <NAME>
murano category-delete <ID> [<ID> ...]
murano category-list
murano category-show <ID>
```

1.13.12 Managing Environmet Templates

It is possible to manage environment templates with following commands:

```
murano env-template-create <NAME>
murano env-template-add-app <NAME> <FILE>
murano env-template-del-app <NAME> <FILE>
murano env-template-delete <ID>
murano env-template-list
murano env-template-show <ID>
murano env-template-update <ID> <NEW_NAME>
```

Repository

1.14 Murano package repository

Murano client and dashboard are both capable of installing packages and bundles of packages from murano repository. To do so you should set `MURANO_REPO_URL` settings in murano dashboard or `MURANO_REPO_URL` env variable for the CLI client and use the respective command for package import. These commands would then automatically import all the prerequisites for the app being installed along with any images, mentioned in said apps.

For more info about importing from repository see *Murano client*.

1.14.1 Setting up your own repository

It's fairly easy to set up your own murano package repository. To do so you need a web server, that would serve 3 directories:

- /apps/
- /bundles/
- /images/

When importing an app by name client would append any version info, if present to the app name, `.zip` file extension and search for that file in the `apps` directory.

When importing a bundle by name client would append `.bundle` file extension to the bundle name and search it in the `bundles` directory. Bundle file is a json or a yaml file with the following structure:

```
{"Packages":
    [
        {"Name": "io.murano.apps.ApacheHttpServer"},
        {"Version": "", "Name": "io.murano.apps.Nginx"},
        {"Version": "0.0.1", "Name": "io.murano.apps.Lighttpd"}
    ]
}
```

Glance images can be auto-imported by client, when mentioned in `images.lst` inside the package. Please see *Composing application package manual* for more info about pakcage composition. When importing images from `image.lst` file client simply searches for a file with the same name as the `Name` attribute of the image in the `images` directory of the repository.

Guidelines

1.15 Contributing to Murano

If you're interested in contributing to the Murano project, the following will help get you started.

1.15.1 Contributor License Agreement

In order to contribute to the Murano project, you need to have signed OpenStack's contributor's agreement:

- http://docs.openstack.org/infra/manual/developers.html
- http://wiki.openstack.org/CLA

1.15.2 Project Hosting Details

- **Bug tracker** https://launchpad.net/murano
- **Mailing list (prefix subjects with `[Murano]` for faster responses)** http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-dev
- **Wiki** https://wiki.openstack.org/wiki/Murano
- **IRC channel**
  - #murano at FreeNode
  - https://wiki.openstack.org/wiki/Meetings#Murano_meeting
- **Code Hosting**
  - https://git.openstack.org/cgit/openstack/murano
  - https://git.openstack.org/cgit/openstack/murano-agent
  - https://git.openstack.org/cgit/openstack/murano-dashboard
  - https://git.openstack.org/cgit/openstack/python-muranoclient
  - https://git.openstack.org/cgit/openstack/murano-apps
- **Code Review**
  - https://review.openstack.org/#/q/murano+AND+status:+open,n,z
  - http://docs.openstack.org/infra/manual/developers.html#development-workflow

1.16 Development Guidelines

1.16.1 Coding Guidelines

For all the code in Murano we have a rule - it should pass PEP 8.

To check your code against PEP 8 run:

```
$ tox -e pep8
```

See also:

- https://pep8.readthedocs.org/en/latest/
- https://flake8.readthedocs.org
- http://docs.openstack.org/developer/hacking/

1.16.2 Testing Guidelines

Murano has a suite of tests that are run on all submitted code, and it is recommended that developers execute the tests themselves to catch regressions early. Developers are also expected to keep the test suite up-to-date with any submitted code changes.

Unit tests are located at `muranoapi/tests`.

Murano's suite of unit tests can be executed in an isolated environment with Tox. To execute the unit tests run the following from the root of Murano repo on Python 2.7:

```
$ tox -e py27
```

For Python 2.6:

```
$ tox -e py26
```

1.16.3 Documentation Guidelines

Murano dev-docs are written using Sphinx / RST and located in the main repo in `doc` directory.

The documentation in docstrings should follow the PEP 257 conventions (as mentioned in the PEP 8 guidelines).

More specifically:

1. Triple quotes should be used for all docstrings.
2. If the docstring is simple and fits on one line, then just use one line.
3. For docstrings that take multiple lines, there should be a newline after the opening quotes, and before the closing quotes.
4. Sphinx is used to build documentation, so use the restructured text markup to designate parameters, return values, etc. Documentation on the sphinx specific markup can be found here:

Run the following command to build docs locally.

```
$ tox -e docs
```

1.17 Murano TroubleShooting and Debug Tips

During installation and setting environment of new projects you can run into different problems. This section intends to reduce the time spent on the solution of these problems.

1.17.1 Problems during configuration

Log location

Murano is a multi component project, there several places where logs could be found.

The location of the log file completely depends on the setting in the config file of the corresponding component. *log_file* parameter points to the log file, and if it's omitted or commented logging will be sent to stdout.

Possible problem list

- *murano-db-manage* failed to execute
  - Check *connection* parameter in provided config file. It should be a connection string.
- Murano Dashboard is not working
  - Make sure, that *prepare_murano.sh* script was executed and *murano* file located in *enabled* folder under openstack_dashboard repository.
  - Check, that murano data is not inserted twice in the settings file and as a plugin.

1.17.2 Problems during deployment

Besides identifying errors from log files, there is another and more flexible way to browse deployment errors - directly from UI. After *Deploy Failed* status is appeared navigate to environment components and open *Deployment History* page. Click on the *Show details* button located at the corresponding deployment row of the table. Then go to the *Logs* tab. You can see steps of the deployments and the one that failed would have red color.

- Deployment freeze after `Begin execution: io.murano.system.Agent.call` problem with connection between Murano Agent and spawned instance.
- Need to check transport access to the virtual machine (check router has gateway).
- Check for rabbitMq settings: verify that agent has been obtained valid rabbit parameters. Go to the spawned virtual machine and open */etc/murano/agent.conf* or *C:MuranoAgentagent.conf* on Windows-based machine. Also, you can examine agent logs, located by default at */var/log/murano-agent.log* The first part of the log file will contain reconnection attempts to the rabbit - since the valid rabbit address and queue have not been obtained yet.
- Check that *notification_driver* option is set to *messagingv2*
- Check that linux image name is not starts with 'w' letter
- `[exceptions.EnvironmentError]: Unexpected stack state NOT_FOUND` - problem with heat stack creation, need to examine Heat log file. If you are running the deployment on new tenant check that router exists and it has gateway to the external network.
- `Router could not be created, no external network found` - Find *external_network* parameter in config file and check that specified external network is really exist via UI or by executiong *neutron net-external-list* cimmand.

- `NoPackageForClassFound: Package for class io.murano.Environment is not found` - Check that murano core package is uploaded. If no, the content of *meta/io.murano* folder should be zipped and uploaded to Murano.

1.18 Migrating applications from Murano v0.5 to Stable/Juno

Applications created for Murano v0.5, unfortunately, are not suported in Murano stable/juno. This document provides the application code changes required for compatibility with the stable/juno Murano version.

1.18.1 Rename *'Workflow'* to *'Methods'*

In stable/juno the name of section containing class methods is renamed to *Methods*, as the latter is more OOP and doesn't cause confusion with Mistral. So, you need to change it in *app.name/Classes* in all classes describing workflow of your app.

For example:

```
Workflow:
  deploy:
    Body:
      - $._environment.reporter.report($this, 'Creating VM')
```

Should be changed to:

```
Methods:
  deploy:
    Body:
      - $._environment.reporter.report($this, 'Creating VM')
```

1.18.2 Change the Instance type in the UI definition 'Application' section

The Instance class was too generic and contained some dirty workarounds to differently handle Windows and Linux images, to bootstrap an instance in a number of ways, etc. To solve these problems more classes were added to the *Instance* inheritance hierarchy.

Now, base *Instance* class is abstract and agnostic of the desired OS and agent type. It is inherited by two classes: *LinuxInstance* and *WindowsInstance*.

- *LinuxInstance* adds a default security rule for Linux, opening a standard SSH port;
- *WindowsInstance* adds a default security rule for Windows, opening an RDP port. At the same time WindowsInstance prepares a user-data allowing to use Murano v1 agent.

LinuxInstance is inherited by two other classes, having different software config method:

- *LinuxMuranoInstance* adds a user-data preparation to configure Murano v2 agent;
- *LinuxUDInstance* adds a custom user-data field allowing the services to supply their own user data.

You need to specify the instance type which is required by your app. It specifies a field in UI, where user can select an image matched to the instance type. This change must be added to UI form definition in *app.name/UI/ui.yaml*.

For example, if you are going to install your application on Ubuntu, you need to change:

```
Application:
  ?:
  instance:
    ?:
      type: io.murano.resources.Instance
```

to:

```
Application:
  ?:
  instance:
    ?:
      type: io.murano.resources.LinuxMuranoInstance
```

API specification

1.19 Murano API v1 specification

1.19.1 General information

- **Introduction**

  Murano Service API is a programmatic interface used for interaction with Murano. Other interaction mechanisms like Murano Dashboard or Murano CLI should use API as underlying protocol for interaction.

- **Allowed HTTPs requests**
  - *POST* : To create a resource
  - *GET* : Get a resource or list of resources
  - *DELETE* : To delete resource
  - *PATCH* : To update a resource
- **Description Of Usual Server Responses**
  - 200 `OK` - the request was successful.
  - 201 `Created` - the request was successful and a resource was created.
  - 204 `No Content` - the request was successful but there is no representation to return (i.e. the response is empty).
  - 400 `Bad Request` - the request could not be understood or required parameters were missing.
  - 401 `Unauthorized` - authentication failed or user didn't have permissions for requested operation.
  - 403 `Forbidden` - access denied.
  - 404 `Not Found` - resource was not found
  - 405 `Method Not Allowed` - requested method is not supported for resource.
  - 409 `Conflict` - requested method resulted in a conflict with the current state of the resource.
- **Response of POSTs and PUTs**

  All POST and PUT requests by convention should return the created object (in the case of POST, with a generated ID) as if it was requested by GET.

- **Authentication**

All requests include a Keystone authentication token header (X-Auth-Token). Clients must authenticate with Keystone before interacting with the Murano service.

1.19.2 Glossary

- **Environment**

  Environment is a set of applications managed by a single tenant. They could be related logically with each other or not. Applications within single Environment may comprise some complex configuration while applications in different Environments are always independent from one another. Each Environment is associated with single OpenStack project (tenant).

- **Session**

  Since Murano environment are available for local modification for different users and from different locations, it's needed to store local modifications somewhere. So sessions were created to provide this opportunity. After user adds application to the environment - new session is created. After user sends environment to deploy, session with set of applications changes status to *deploying* and all other open sessions for that environment becomes invalid. One session could be deployed only once.

- **Object Model**

  Applications are defined in MuranoPL object model, which is defined as JSON object. Murano API doesn't know anything about it.

- **Package**

  A .zip archive, containing instructions for an application deployment.

- **Environment-Template** The Environment template is the specification of a set of applications managed by a single tenant, which are related each other. The environment template is stored in a environment template catalogue, and it can be managed by the user (creation, deletion, updating...). Finally, it can be deployed on Openstack by translating into an environment.

1.19.3 Environment API

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name
created	datetime	Creation date and time in ISO format
updated	datetime	Modification date and time in ISO format
tenant_id	string	OpenStack tenant ID
version	int	Current version
networking	string	Network settings
status	string	Deployment status: ready, pending, deploying

Common response codes

Code	Description
200	Operation completed successfully
401	User is not authorized to perform the operation

List Environments

Request

Method	URI	Description
GET	/environments	Get a list of existing Environments

Response

This call returns list of environments. Only the basic properties are returned.


```
{
    "environments": [
        {
            "status": "ready",
            "updated": "2014-05-14T13:02:54",
            "networking": {},
            "name": "test1",
            "created": "2014-05-14T13:02:46",
            "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
            "version": 0,
            "id": "2fa5ab704749444bbeafe7991b412c33"
        },
        {
            "status": "ready",
            "updated": "2014-05-14T13:02:55",
            "networking": {},
            "name": "test2",
            "created": "2014-05-14T13:02:51",
            "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
            "version": 0,
            "id": "744e44812da84e858946f5d817de4f72"
        }
    ]
}
```


Create Environment

Attribute	Type	Description
name	string	Environment name; only alphanumeric characters and '-'

Request

Method	URI	Description
POST	/environments	Create new Environment

- **Content-Type** application/json
- **Example** {"name": "env_name"}

Response


```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "env_name",
    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:23:44Z",
    "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
    "version": 0
}
```

Update Environment

Attribute	Type	Description
name	string	Environment name; only alphanumeric characters and '-'

Request

Method	URI	Description
PUT	/environments/<env_id>	Update an existing Environment

- **Content-Type** application/json
- **Example** {"name": "env_name_changed"}

Response

Content-Type application/json

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "env_name_changed",
    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:45:54Z",
    "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
    "version": 0
}
```

Get Environment Details

Request

Return information about environment itself and about applications, including to this environment.

Method	URI	Header	Description
GET	/environments/{id}	X-Configuration-Session (optional)	Response detailed information about Environment including child entities

Response

Content-Type application/json

```
{
    "status": "ready",
    "updated": "2014-05-14T13:12:26",
    "networking": {},
    "name": "quick-env-2",
    "created": "2014-05-14T13:09:55",
    "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
    "version": 1,
    "services": [
        {
            "instance": {
                "flavor": "m1.medium",
                "image": "cloud-fedora-v3",
                "name": "exgchhv6nbika2",
                "ipAddresses": [
                    "10.0.0.200"
                ],
                "?": {
                    "type": "io.murano.resources.Instance",
                    "id": "14cce9d9-aaa1-4f09-84a9-c4bb859edaff"
```

```
                }
            },
            "name": "rewt4w56",
            "?": {
                "status": "ready",
                "_26411a1861294160833743e45d0eaad9": {
                    "name": "Telnet"
                },
                "type": "io.murano.apps.linux.Telnet",
                "id": "446373ef-03b5-4925-b095-6c56568fa518"
            }
        }
    ],
    "id": "20d4a012628e4073b48490a336a8acbf"
}
```


Delete Environment

Request

Method	URI	Description
DELETE	/environments/{id}	Remove specified Environment.

1.19.4 Environment Configuration API

Multiple sessions could be opened for one environment simultaneously, but only one session going to be deployed. First session that starts deploying is going to be deployed; other ones become invalid and could not be deployed at all. User could not open new session for environment that in *deploying* state (that's why we call it "almost lock free" model).

Attribute	Type	Description
id	string	Session unique ID
environment_id	string	Environment that going to be modified during this session
created	datetime	Creation date and time in ISO format
updated	datetime	Modification date and time in ISO format
user_id	string	Session owner ID
version	int	Environment version for which configuration session is opened
state	string	Session state. Could be: open, deploying, deployed

Configure Environment / Open session

During this call new working session is created, and session ID should be sent in a request header with name `X-Configuration-Session`.

Request

Method	URI	Description
POST	/environments/<env_id>/configure	Creating new configuration session

Response

Content-Type application/json

```
{
    "updated": datetime.datetime(2014, 5, 14, 14, 17, 58, 949358),
    "environment_id": "744e44812da84e858946f5d817de4f72",
    "ser_id": "4e91d06270c54290b9dbdf859356d3b3",
    "created": datetime.datetime(2014, 5, 14, 14, 17, 58, 949305),
    "state": "open", "version": 0L, "id": "257bef44a9d848daa5b2563779714820"
 }
```

Code	Description
200	Session created successfully
401	User is not authorized to access this session
403	Could not open session for environment, environment has deploying status

Deploy Session

With this request all local changes made within environment start to deploy on Openstack.

Request

Method	URI	Description
POST	/environments/<env_id>/sessions/ <session_id>/deploy	**Deploy changes made in session** with specified session_id

Response

Code	Description
200	Session status changes to *deploying*
401	User is not authorized to access this session
403	Session is already deployed or deployment is in progress

Get Session Details

Request

Method	URI	Description
GET	/environments/<env_id>/sessions/ <session_id>	Get details about session with specified session_id

Response

```
{
    "id": "4aecdc2178b9430cbbb8db44fb7ac384",
    "environment_id": "4dc8a2e8986fa8fa5bf24dc8a2e8986fa8",
    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:23:54Z",
    "user_id": "d7b501094caf4daab08469663a9e1a2b",
    "version": 0,
    "state": "deploying"
}
```

Code	Description
200	Session details information received
401	User is not authorized to access this session
403	Session is invalid

Delete Session

Request

Method	URI	Description
DELETE	/environments/<env_id>/sessions/ <session_id>	Delete session with specified session_id

Response

Code	Description
200	Session is deleted successfully
401	User is not authorized to access this session
403	Session is in deploying state and could not be deleted

1.19.5 Environment Deployments API

Environment deployment API allows to track changes of environment status, deployment events and errors. It also allows to browse deployment history.

List Deployments

Returns information about all deployments of the specified environment.

Request

Method	URI	Description
GET	/environments/<env_id>/deployments	Get list of environment deployments

Response

Content-Type application/json

```
{
    "deployments": [
        {
            "updated": "2014-05-15T07:24:21",
            "environment_id": "744e44812da84e858946f5d817de4f72",
            "description": {
                "services": [
                    {
                        "instance": {
                            "flavor": "m1.medium",
                            "image": "cloud-fedora-v3",
                            "?": {
                                "type": "io.murano.resources.Instance",
                                "id": "ef729199-c71e-4a4c-a314-0340e279add8"
                            },
                            "name": "xkaduhv7qeg4m7"
                        },
                        "name": "teslnet1",
                        "?": {
                            "_26411a1861294160833743e45d0eaad9": {
                                "name": "Telnet"
                            },
                            "type": "io.murano.apps.linux.Telnet",
                            "id": "6e437be2-b5bc-4263-8814-6fd57d6ddbd5"
                        }
                    }
```

```
                ],
                "defaultNetworks": {
                    "environment": {
                        "name": "test2-network",
                        "?": {
                            "type": "io.murano.lib.networks.neutron.NewNetwork",
                            "id": "b6a1d515434047d5b4678a803646d556"
                        }
                    },
                    "flat": null
                },
                "name": "test2",
                "?": {
                    "type": "io.murano.Environment",
                    "id": "744e44812da84e858946f5d817de4f72"
                }
            },
            "created": "2014-05-15T07:24:21",
            "started": "2014-05-15T07:24:21",
            "finished": null,
            "state": "running",
            "id": "327c81e0e34a4c93ad9b9052ef42b752"
        }
    ]
}
```


Code	Description
200	Deployments information received successfully
401	User is not authorized to access this environment

1.19.6 Application Management API

All applications should be created within an environment and all environment modifications are held within the session. Local changes apply only after successful deployment of an environment session.

Get Application Details

Using GET requests to applications endpoint user works with list containing all applications for specified environment. User can request whole list, specific application, or specific attribute of specific application using tree traversing. To request specific application, user should add to endpoint part an application id, e.g.: */environments/<env_id>/services/<application_id>*. For selection of specific attribute on application, simply appending part with attribute name will work. For example to request application name, user should use next endpoint: */environments/<env_id>/services/<application_id>/name*

Request

Method	URI	Header
GET	/environments/<env_id>/services<app_id>	X-Configuration-Session (optional)

Parameters:

- *env_id* - environment ID, required
- *app_id* - application ID, optional

Response

Content-Type application/json

```
{
    "instance": {
        "flavor": "m1.medium",
        "image": "cloud-fedora-v3",
        "?": {
            "type": "io.murano.resources.Instance",
            "id": "060715ff-7908-4982-904b-3b2077ff55ef"
        },
        "name": "hbhmyhv6qihln3"
    },
    "name": "dfg34",
    "?": {
        "status": "pending",
        "_26411a1861294160833743e45d0eaad9": {
            "name": "Telnet"
        },
        "type": "io.murano.apps.linux.Telnet",
        "id": "6e7b8ad5-888d-4c5a-a498-076d092a7eff"
    }
}
```

POST applications

New application can be added to the Murano environment using session. Result JSON is calculated in Murano dashboard, which based on UI definition

Request

Content-Type application/json

Method	URI	Header
POST	/environments/<env_id>/services	X-Configuration-Session

```
{
  "instance": {
    "flavor": "m1.medium",
    "image": "clod-fedora-v3",
    "?": {
      "type": "io.murano.resources.Instance",
      "id": "bce8308e-5938-408b-a27a-0d3f0a2c52eb"
    },
    "name": "nhekhv6r7mhd4"
  },
  "name": "sdf34sadf",
  "?": {
    "_26411a1861294160833743e45d0eaad9": {
      "name": "Telnet"
    },
    "type": "io.murano.apps.linux.Telnet",
    "id": "190c8705-5784-4782-83d7-0ab55a1449aa"
  }
}
```

Response

Created application returned

Content-Type application/json

```
{
    "instance": {
        "flavor": "m1.medium",
        "image": "cloud-fedora-v3",
        "?": {
            "type": "io.murano.resources.Instance",
            "id": "bce8308e-5938-408b-a27a-0d3f0a2c52eb"
        },
        "name": "nhekhv6r7mhd4"
    },
    "name": "sdf34sadf",
    "?": {
        "_26411a1861294160833743e45d0eaad9": {
            "name": "Telnet"
        },
        "type": "io.murano.apps.linux.Telnet",
        "id": "190c8705-5784-4782-83d7-0ab55a1449a1"
    }
}
```

Delete application from environment

Delete one or all applications from the environment

Request

Method	URI	Header
DELETE	/environments/<env_id>/services/<app_id>	X-Configuration-Session(optional)

Parameters:

- *env_id* - environment ID, required
- *app_id* - application ID, optional

1.19.7 Statistic API

Statistic API intends to provide billing feature

Instance Environment Statistics

Request

Get information about all deployed instances in the specified environment

Method	URI
GET	/environments/<env_id>/instance-statistics/raw/<instance_id>

Parameters:

- *env_id* - environment ID, required
- *instance_id* - ID of the instance for which need to provide statistic information, optional

Response

Attribute	Type	Description
type	int	Code of the statistic object; 200 - instance, 100 - application
type_name	string	Class name of the statistic object
instance_id	string	Id of deployed instance
active	bool	Instance status
type_title	string	User-friendly name for browsing statistic in UI
duration	int	Seconds of instance uptime

Content-Type application/json

```
[
    {
        "type": 200,
        "type_name": "io.murano.resources.Instance",
        "instance_id": "ef729199-c71e-4a4c-a314-0340e279add8",
        "active": true,
        "type_title": null,
        "duration": 1053,
    }
]
```

Request

Method	URI
GET	/environments/<env_id>/instance-statistics/aggregated

Response

Attribute	Type	Description
type	int	Code of the statistic object; 200 - instance, 100 - application
duration	int	Amount uptime of specified type objects
count	int	Quantity of specified type objects

Content-Type

application/json

```
[
    {
        "duration": 720,
        "count": 2,
        "type": 200
    }
]
```

General Request Statistics

Request

Method	URI
GET	/stats

Response

Attribute	Type	Description
requests_per_tenant	int	Number of incoming requests for user tenant
errors_per_second	int	Class name of the statistic object
errors_count	int	Class name of the statistic object
requests_per_second	float	Average number of incoming request received in one second
requests_count	int	Number of all requests sent to the server
cpu_percent	bool	Current cpu usage
cpu_count	int	Available cpu power is `cpu_count * 100%`
host	string	Server host-name
average_response_time	float	Average time response waiting, seconds

Content-Type application/json


```
[
    {
        "updated": "2014-05-15T08:26:17",
        "requests_per_tenant": "{\"726ed856965f43cc8e565bc991fa76c3\": 313}",
        "created": "2014-04-29T13:23:59",
        "cpu_count": 2,
        "errors_per_second": 0,
        "requests_per_second": 0.0266528,
        "cpu_percent": 21.7,
        "host": "fervent-VirtualBox",
        "error_count": 0,
        "request_count": 320,
        "id": 1,
        "average_response_time": 0.55942
    }
]
```


1.19.8 Application Catalog API

Manage application definitions in the Application Catalog. You can browse, edit and upload new application packages (.zip.package archive with all data that required for a service deployment).

1.19.9 Packages

Methods for application package management

Package Properties

- `id`: guid of a package (`fully_qualified_name` can also be used for some API functions)
- `fully_qualified_name`: fully qualified domain name - domain name that specifies exact application location
- `name`: user-friendly name
- `type`: package type, “library” or “application”
- `description`: text information about application
- `author`: name of application author
- `tags`: list of short names, connected with the package, which allows to search applications easily
- `categories`: list of application categories
- `class_definition`: list of class names used by a package

- `is_public`: determines whether the package is shared for other tenants
- `enabled`: determines whether the package is browsed in the Application Catalog
- `owner_id`: id of a tenant that owns the package

List packages

/v1/catalog/packages?{marker}{limit}{order_by}{type}{category}{fqn}{owned}{catalog}{class_name} [GET]

This is the compound request to list and search through application catalog. If there are no search parameters all packages that is_public, enabled and belong to the user's tenant will be listed. Default order is by 'created' field. For an admin role all packages are available.

Parameters

Attribute	Type	Description
`catalog`	bool	If false (default) - search packages, that current user can edit (own for non-admin, all for admin) If true - search packages, that current user can deploy (i.e. his own + public)
`marker`	string	A package identifier marker may be specified. When present only packages which occur after the identifier ID will be listed
`limit`	string	When present the maximum number of results returned will not exceed the specified value. The typical pattern of limit and marker is to make an initial limited request and then to use the ID of the last package from the response as the marker parameter in a subsequent limited request.
`order_by`	string	Allows to sort packages by: *fqn*, *name*, *created*. Created is default value.
`type`	string	Allows to point a type of package: *application*, *library*
`category`	string	Allows to point a categories for a search
`fqn`	string	Allows to point a fully qualified package name for a search
`owned`	bool	Search only from packages owned by current tenant
`include_disabled`	bool	Include disabled packages in a the result
`search`	string	Gives opportunity to search specified data by all the package parameters
`class_name`	string	Search only for packages, that use specified class

Response 200 (application/json)

```
{"packages": [
              {
                "id": "fed57567c9fa42c192dcbe0566f8ea33",
                 "fully_qualified_name" : "com.example.murano.services.linux.telnet",
                 "is_public": false,
                 "name": "Telnet",
                 "type": "linux",
                 "description": "Installs Telnet service",
                 "author": "Openstack, Inc.",
                 "created": "2014-04-02T14:31:55",
                 "enabled": true,
                 "tags": ["linux", "telnet"],
                 "categories": ["Utility"],
                 "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
              },
              {
                "id": "fed57567c9fa42c192dcbe0566f8ea31",
                "fully_qualified_name": "com.example.murano.services.windows.WebServer",
                "is_public": true,
                "name": "Internet Information Services",
                "type": "windows",
                "description": "The  Internet Information Service sets up an IIS server and joins it
```

```
                "author": "Openstack, Inc.",
                "created": "2014-04-02T14:31:55",
                "enabled": true,
                "tags": ["windows", "web"],
                "categories": ["Web"],
                "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
              }]
 }
```

Upload a new package[POST]

/v1/catalog/packages

See the example of multipart/form-data request, It should contain two parts - text (json string) and file object

Request (multipart/form-data)

```
Content-type: multipart/form-data, boundary=AaB03x
Content-Length: $requestlen

--AaB03x
content-disposition: form-data; name="submit-name"

--AaB03x
Content-Disposition: form-data; name="JsonString"
Content-Type: application/json

{"categories":["web"] , "tags": ["windows"], "is_public": false, "enabled": false}
`categories` - array, required
`tags` - array, optional
`name` - string, optional
`description` - string, optional
`is_public` - bool, optional
`enabled` - bool, optional

--AaB03x
content-disposition: file; name="file"; filename="test.tar"
Content-Type: targz
Content-Transfer-Encoding: binary

$binarydata
--AaB03x--
```

Response 200 (application/json)

```
{
    "updated": "2014-04-03T13:00:13",
    "description": "A domain service hosted in Windows environment by using Active Directory Role",
    "tags": ["windows"],
    "is_public": true,
    "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
    "categories": ["test1"],
    "name": "Active Directory",
    "author": "Mirantis, Inc",
    "created": "2014-04-03T13:00:13",
    "enabled": true,
    "class_definition": [
        "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
```

```
        "com.mirantis.murano.windows.activeDirectory.SecondaryController",
        "com.mirantis.murano.windows.activeDirectory.Controller",
        "com.mirantis.murano.windows.activeDirectory.PrimaryController"
    ],
    "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "type": "Application",
    "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```


Get package details

/v1/catalog/packages/{id} [GET]

Display details for a package.

Parameters

`id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/json)


```
{
    "updated": "2014-04-03T13:00:13",
    "description": "A domain service hosted in Windows environment by using Active Directory Role",
    "tags": ["windows"],
    "is_public": true,
    "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
    "categories": ["test1"],
    "name": "Active Directory",
    "author": "Mirantis, Inc",
    "created": "2014-04-03T13:00:13",
    "enabled": true,
    "class_definition": [
        "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
        "com.mirantis.murano.windows.activeDirectory.SecondaryController",
        "com.mirantis.murano.windows.activeDirectory.Controller",
        "com.mirantis.murano.windows.activeDirectory.PrimaryController"
    ],
    "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "type": "Application",
    "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```


Response 403

- In attempt to get non-public package by user whose tenant is not an owner of this package.

Response 404

- In case specified package id doesn't exist.

1.19.10 Update a Package

/v1/catalog/packages/{id} [PATCH]

Allows to edit mutable fields (categories, tags, name, description, is_public, enabled). See the full specification here.

Parameters

`id` (required) Hexadecimal *id* (or fully qualified name) of the package

Content type

application/murano-packages-json-patch

Allowed operations:

```
[
    { "op": "add", "path": "/tags", "value": [ "foo", "bar" ] },
    { "op": "add", "path": "/categories", "value": [ "foo", "bar" ] },
    { "op": "remove", "path": "/tags", ["foo"] },
    { "op": "remove", "path": "/categories", ["foo"] },
    { "op": "replace", "path": "/tags", "value": [] },
    { "op": "replace", "path": "/categories", "value": ["bar"] },
    { "op": "replace", "path": "/is_public", "value": true },
    { "op": "replace", "path": "/enabled", "value": true },
    { "op": "replace", "path": "/description", "value":"New description" },
    { "op": "replace", "path": "/name", "value": "New name" }
]
```

Request 200 (application/murano-packages-json-patch)

```
[
 { "op": "add", "path": "/tags", "value": [ "windows", "directory"] },
 { "op": "add", "path": "/categories", "value": [ "Directory" ] }
]
```

Response 200 (application/json)

```
{
    "updated": "2014-04-03T13:00:13",
    "description": "A domain service hosted in Windows environment by using Active Directory Role",
    "tags": ["windows", "directory"],
    "is_public": true,
    "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
    "categories": ["test1"],
    "name": "Active Directory",
    "author": "Mirantis, Inc",
    "created": "2014-04-03T13:00:13",
    "enabled": true,
    "class_definition": [
        "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
        "com.mirantis.murano.windows.activeDirectory.SecondaryController",
        "com.mirantis.murano.windows.activeDirectory.Controller",
        "com.mirantis.murano.windows.activeDirectory.PrimaryController"
    ],
    "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "type": "Application",
    "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```

Response 403

- An attempt to update immutable fields
- An attempt to perform operation that is not allowed on the specified path
- An attempt to update non-public package by user whose tenant is not an owner of this package

Response 404

- An attempt to update package that doesn't exist

Delete application definition from the catalog

/v1/catalog/packages/{id} [DELETE]

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package to delete

Response 404

- An attempt to delete package that doesn't exist

Get application package

/v1/catalog/packages/{id}/download [GET]

Get application definition package

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/octetstream)

The sequence of bytes representing package content

Response 404

Specified package id doesn't exist

Get UI definition

/v1/catalog/packages/{id}/ui [GET]

Retrieve UI definition for a application which described in a package with provided id

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/octet-stream)

The sequence of bytes representing UI definition

Response 404

Specified package id doesn't exist

Response 403

Specified package is not public and not owned by user tenant, performing the request

Response 404

- Specified package id doesn't exist

Get logo

Retrieve application logo which described in a package with provided id

/v1/catalog/packages/{id}/logo [GET]

Parameters

`id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/octet-stream)

The sequence of bytes representing application logo

Response 403

Specified package is not public and not owned by user tenant, performing the request

Response 404

Specified package is not public and not owned by user tenant, performing the request

1.19.11 Categories

Provides category management. Categories are used in the Application Catalog to group application for easy browsing and search.

List categories

- */v1/catalog/packages/categories [GET]*

  !DEPRECATED (Plan to remove in L release) Retrieve list of all available application categories

  Response 200 (application/json)

  A list, containing category names

  Content-Type application/json

```
{
    "categories": ["Web service", "Directory", "Database", "Storage"]
}
```

- */v1/catalog/categories [GET]*

Method	URI	Description
GET	/catalog/categories	Get list of existing categories

  Retrieve list of all available application categories

  Response 200 (application/json)

  A list, containing detailed information about each category

  Content-Type application/json

```
{"categories": [
    {
        "id": "0420045dce7445fabae7e5e61fff9e2f",
        "updated": "2014-12-26T13:57:04",
        "name": "Web",
        "created": "2014-12-26T13:57:04",
        "package_count": 1
    },
    {
        "id": "3dd486b1e26f40ac8f35416b63f52042",
        "updated": "2014-12-26T13:57:04",
        "name": "Databases",
        "created": "2014-12-26T13:57:04",
        "package_count": 0
```

```
        }]
    }
```

Get category details

/catalog/categories/<category_id> [GET]

Return detailed information for a provided category

Request

Method	URI	Description
GET	/catalog/categories/<category_id>	Get category detail

Parameters

- `category_id` - required, category ID, required

Response

Content-Type application/json

```
{
    "id": "b308f7fa8a2f4a5eb419970c827f4466",
    "updated": "2015-01-28T17:00:19",
    "packages": [
        {
            "fully_qualified_name": "io.murano.apps.ZabbixServer",
            "id": "4dfb566e69e6445fbd4aea5099fe95e9",
            "name": "Zabbix Server"
        }
    ],
    "name": "Web",
    "created": "2015-01-28T17:00:19",
    "package_count": 1
}
```

Code	Description
200	OK. Category deleted successfully
401	User is not authorized to access this session
404	Not found. Specified category doesn‘t exist

Add new category

/catalog/categories [POST]

Add new category to the Application Catalog

Parameters

Attribute	Type	Description
name	string	Environment name; only alphanumeric characters and ‘-‘

Request

Method	URI	Description
POST	/catalog/categories	Create new category

Content-Type application/json

Example {“name”: “category_name”}

Response

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "category_name",
    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:23:44Z",
    "package_count": 0
}
```

Code	Description
200	OK. Category created successfully
401	User is not authorized to access this session
409	Conflict. Category with specified name already exist

Delete category

/catalog/categories [DELETE]

Request

Method	URI	Description
DELETE	/catalog/categories/<category_id>	Delete category with specified id

Parameters:

- `category_id` - required, category ID, required

Response

Code	Description
200	OK. Category deleted successfully
401	User is not authorized to access this session
404	Not found. Specified category doesn‘t exist
403	Forbidden. Category with specified name is assigned to the package, presented in the catalog

1.19.12 Environment Template API

Manage environment template definitions in Murano. It is possible to create, update, delete and deploy into Openstack by translating it into an environment. In addition, applications can be added or delete to the environment template.

Environment Template Properties

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name
created	datetime	Creation date and time in ISO format
updated	datetime	Modification date and time in ISO format
tenant_id	string	OpenStack tenant ID
version	int	Current version
networking	string	Network settings
description	string	The environment template specification

Common response codes

Code	Description
200	Operation completed successfully
401	User is not authorized to perform the operation

Methods for Environment Template API

List Environments Templates

Request

Method	URI	Description
GET	/templates	Get a list of existing environment templates

Response

This call returns list of environment templates. Only the basic properties are returned.

```
{
    "templates": [
        {
            "updated": "2014-05-14T13:02:54",
            "networking": {},
            "name": "test1",
            "created": "2014-05-14T13:02:46",
            "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
            "version": 0,
            "id": "2fa5ab704749444bbeafe7991b412c33"
        },
        {
            "updated": "2014-05-14T13:02:55",
            "networking": {},
            "name": "test2",
            "created": "2014-05-14T13:02:51",
            "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
            "version": 0,
            "id": "744e44812da84e858946f5d817de4f72"
        }
    ]
}
```

Create Environment Template

Request

Method	URI	Description
POST	/templates	Create a new environment template

Content-Type application/json

Example {“name”: “env_temp_name”}

Response

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "env_temp_name",
    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:23:44Z",
    "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
}
```

Error code

Code	Description
200	Operation completed successfully
401	User is not authorized to perform the operation
409	The environment template already exists

Get Environment Templates Details

Request

Return information about environment template itself and about applications, including to this environment template.

Method	URI	Description
GET	/templates/{env-temp-id}	Obtains the enviroment template information

- *env-temp-id* - environment template ID, required

Response

Content-Type application/json

```
{
  "updated": "2015-01-26T09:12:51",
  "networking":
  {
  },
  "name": "template_name",
  "created": "2015-01-26T09:12:51",
  "tenant_id": "00000000000000000000000000000001",
  "version": 0,
  "id": "aa9033ca7ce245fca10e38e1c8c4bbf7",
}
```

Error code

Code	Description
200	OK. Environment Template created successfully
401	User is not authorized to access this session
404	The environment template does not exit

Delete Environment Template

Request

Method	URI	Description
DELETE	/templates/<env-temp-id>	Delete the template id

Parameters:

- *env-temp_id* - environment template ID, required

Error code

Code	Description
200	OK. Environment Template created successfully
401	User is not authorized to access this session
404	The environment template does not exit

Adding application to environment template

Request

Method	URI	Description
POST	/templates/{env-temp-id}/services	Create a new application

Parameters:

- *env-temp-id* - The environment-template id, required
- payload - the service description

Content-Type application/json

Example

```
{
    "instance": {
        "assignFloatingIp": "true",
        "keyname": "mykeyname",
        "image": "cloud-fedora-v3",
        "flavor": "m1.medium",
        "?": {
            "type": "io.murano.resources.LinuxMuranoInstance",
            "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
        }
    },
    "name": "orion",
    "port": "8080",
    "?": {
        "type": "io.murano.apps.apache.Tomcat",
        "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
    }
}
```

Response

```
{
   "instance":
   {
       "assignFloatingIp": "true",
       "keyname": "mykeyname",
       "image": "cloud-fedora-v3",
       "flavor": "m1.medium",
       "?":
       {
           "type": "io.murano.resources.LinuxMuranoInstance",
           "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
       }
   },
   "name": "orion",
   "?":
   {
       "type": "io.murano.apps.apache.Tomcat",
       "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
   },
   "port": "8080"
}
```

Error code

Code	Description
200	OK. Environment Template created successfully
401	User is not authorized to access this session
404	The environment template does not exit

Get applications information from an environment template

Request

Method	URI | Description
GET	/templates/{env-temp-id}/services | It obtains the service description

Parameters:

- *env-temp-id* - The environment template ID, required

Content-Type application/json

Response

```
[
    {
        "instance":
        {
            "assignFloatingIp": "true",
            "keyname": "mykeyname",
            "image": "cloud-fedora-v3",
            "flavor": "m1.medium",
            "?":
            {
                "type": "io.murano.resources.LinuxMuranoInstance",
                "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
            }
        },
        "name": "tomcat",
        "?":
        {
            "type": "io.murano.apps.apache.Tomcat",
            "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
        },
        "port": "8080"
    },
    {
        "instance": "ef984a74-29a4-45c0-b1dc-2ab9f075732e",
        "password": "XXX",
        "name": "mysql",
        "?":
        {
            "type": "io.murano.apps.database.MySQL",
            "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
        }
    }
]
```

Error code

Code	Description
200	OK. Environment Template created successfully
401	User is not authorized to access this session
404	The environment template does not exit

Create an environment from an environment template

Request

Method	URI | Description
POST	/templates/{env-temp-id}/create-environment| Create an environment

Parameters:

- *env-temp-id* - The environment template ID, required

Payload:

- 'environment name': The environment name to be created.

Content-Type application/json

Example

```
{
    "name": "environment_name"
}
```

Response

```
{
    "environment_id": "aa90fadfafca10e38e1c8c4bbf7",
    "name": "environment_name",
    "created": "2015-01-26T09:12:51",
    "tenant_id": "00000000000000000000000000000001",
    "version": 0,
    "session_id": "adf4dadfaa9033ca7ce245fca10e38e1c8c4bbf7",
}
```

Error code

Code	Description
200	OK. Environment template created successfully
401	User is not authorized to access this session
404	The environment template does not exit
409	The environment already exists

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*