Murano Specs

Release

OpenStack Murano Team

August 06, 2015

Contents

1 Environment Template Catalogue **3**
- 1.1 Problem description . . . 3
- 1.2 Proposed change . . . 3
- 1.3 Implementation . . . 9
- 1.4 Dependencies . . . 10
- 1.5 Testing . . . 10
- 1.6 Documentation Impact . . . 10
- 1.7 References . . . 10

2 Configuration Language Support **11**
- 2.1 Problem description . . . 11
- 2.2 Proposed change . . . 11
- 2.3 Implementation . . . 13
- 2.4 Dependencies . . . 14
- 2.5 Testing . . . 14
- 2.6 Documentation Impact . . . 14
- 2.7 References . . . 14

3 Policy Guided Fulfillment - Congress Support in Murano **15**
- 3.1 Problem description . . . 15
- 3.2 Proposed change . . . 15
- 3.3 Implementation . . . 17
- 3.4 Dependencies . . . 17
- 3.5 Testing . . . 17
- 3.6 Documentation Impact . . . 17
- 3.7 References . . . 17

4 Provide opportunity to manage application categories **19**
- 4.1 Problem description . . . 19
- 4.2 Proposed change . . . 19
- 4.3 Implementation . . . 22
- 4.4 Dependencies . . . 22
- 4.5 Testing . . . 22
- 4.6 Documentation Impact . . . 23
- 4.7 References . . . 23

5 Add timeouts to murano-agent calls **25**
- 5.1 Problem description . . . 25

5.2 Proposed change . . . 25
5.3 Implementation . . . 27
5.4 Dependencies . . . 27
5.5 Testing . . . 27
5.6 Documentation Impact . . . 27
5.7 References . . . 27

6 murano-mistral-integration **29**
6.1 Problem description . . . 29
6.2 Proposed change . . . 29
6.3 Implementation . . . 30
6.4 Dependencies . . . 30
6.5 Testing . . . 31
6.6 Documentation Impact . . . 31
6.7 References . . . 31

7 Murano Repository Support **33**
7.1 Problem description . . . 33
7.2 Proposed change . . . 33
7.3 Implementation . . . 35
7.4 Dependencies . . . 35
7.5 Testing . . . 35
7.6 Documentation Impact . . . 36
7.7 References . . . 36

8 Example Spec - The title of your blueprint **37**
8.1 Problem description . . . 37
8.2 Proposed change . . . 38
8.3 Implementation . . . 40
8.4 Dependencies . . . 40
8.5 Testing . . . 40
8.6 Documentation Impact . . . 40
8.7 References . . . 40

9 Plugable pythonic classes for Murano **43**
9.1 Problem description . . . 43
9.2 Proposed change . . . 43
9.3 Implementation . . . 45
9.4 Dependencies . . . 45
9.5 Testing . . . 46
9.6 Documentation Impact . . . 46
9.7 References . . . 46

10 Policy Guided Fulfillment - Policy Enforcement Point **47**
10.1 Problem description . . . 47
10.2 Proposed change . . . 48
10.3 Implementation . . . 49
10.4 Dependencies . . . 49
10.5 Testing . . . 50
10.6 Documentation Impact . . . 50
10.7 References . . . 50

11 Liberty specifications **51**
11.1 Murano . . . 51
11.2 Murano Project Resources . . . 51

11.3 License . . . 51

12 Add support for heat environments **53**
12.1 Problem description . . . 53
12.2 Proposed change . . . 53
12.3 Implementation . . . 55
12.4 Dependencies . . . 56
12.5 Testing . . . 56
12.6 Documentation Impact . . . 56
12.7 References . . . 56

13 Add support for heat environments and files **57**
13.1 Problem description . . . 57
13.2 Proposed change . . . 57
13.3 Implementation . . . 58
13.4 Dependencies . . . 59
13.5 Testing . . . 59
13.6 Documentation Impact . . . 59
13.7 References . . . 59

14 Download bundle of packages to local directory using muranoclient **61**
14.1 Problem description . . . 61
14.2 Proposed change . . . 61
14.3 Implementation . . . 62
14.4 Dependencies . . . 63
14.5 Testing . . . 63
14.6 Documentation Impact . . . 63
14.7 References . . . 63

15 Rework package class loader logic **65**
15.1 Problem description . . . 65
15.2 Proposed change . . . 65
15.3 Implementation . . . 66
15.4 Dependencies . . . 67
15.5 Testing . . . 67
15.6 Documentation Impact . . . 67
15.7 References . . . 67

16 Implement Cloud Foundry Service Broker API **69**
16.1 Problem description . . . 69
16.2 Proposed change . . . 69
16.3 Implementation . . . 73
16.4 Dependencies . . . 74
16.5 Testing . . . 74
16.6 Documentation Impact . . . 74
16.7 References . . . 74

17 Remove name field from fields and object model in dynamic UI **75**
17.1 Problem description . . . 75
17.2 Proposed change . . . 75
17.3 Implementation . . . 76
17.4 Dependencies . . . 77
17.5 Testing . . . 77
17.6 Documentation Impact . . . 77
17.7 References . . . 77

18 Configure environments from CLI **79**
18.1 Problem description . . . 79
18.2 Proposed change . . . 79
18.3 Implementation . . . 81
18.4 Dependencies . . . 82
18.5 Testing . . . 82
18.6 Documentation Impact . . . 82
18.7 References . . . 82

19 Environment abandoning support **83**
19.1 Problem description . . . 83
19.2 Proposed change . . . 83
19.3 Implementation . . . 85
19.4 Dependencies . . . 85
19.5 Testing . . . 85
19.6 Documentation Impact . . . 85
19.7 References . . . 85

20 Logging API for MuranoPL **87**
20.1 Problem description . . . 87
20.2 Proposed change . . . 87
20.3 Implementation . . . 89
20.4 Dependencies . . . 89
20.5 Testing . . . 89
20.6 Documentation Impact . . . 89
20.7 References . . . 89

21 Murano API - All Tenants Search **91**
21.1 Problem description . . . 91
21.2 Proposed change . . . 91
21.3 Implementation . . . 93
21.4 Dependencies . . . 94
21.5 Testing . . . 94
21.6 Documentation Impact . . . 94
21.7 References . . . 94

22 Murano API - Core Model Component Integration Improvement **95**
22.1 Problem description . . . 95
22.2 Proposed change . . . 95
22.3 Implementation . . . 97
22.4 Dependencies . . . 97
22.5 Testing . . . 97
22.6 Documentation Impact . . . 97
22.7 References . . . 97

23 Murano unified logging **99**
23.1 Problem description . . . 99
23.2 Proposed change . . . 99
23.3 Implementation . . . 101
23.4 Dependencies . . . 101
23.5 Testing . . . 101
23.6 Documentation Impact . . . 101
23.7 References . . . 101

24 Policy Based Modification of Environment **103**

24.1 Problem description . . . 103
24.2 Proposed change . . . 103
24.3 Implementation . . . 105
24.4 Dependencies . . . 105
24.5 Testing . . . 105
24.6 Documentation Impact . . . 105
24.7 References . . . 105

25 Simulated Execution Mode For Murano Engine **107**
25.1 Problem description . . . 107
25.2 Proposed change . . . 107
25.3 Implementation . . . 109
25.4 Dependencies . . . 110
25.5 Testing . . . 110
25.6 Documentation Impact . . . 111
25.7 References . . . 111

26 Add network selection element to UI form **113**
26.1 Problem description . . . 113
26.2 Proposed change . . . 114
26.3 Implementation . . . 115
26.4 Dependencies . . . 116
26.5 Testing . . . 116
26.6 Documentation Impact . . . 116
26.7 References . . . 116

27 Indices and tables **117**

Kilo specs:

CHAPTER 1

Environment Template Catalogue

https://blueprints.launchpad.net/murano/+spec/blueprint-template

1.1 Problem description

One of powerful Murano use cases is deploying compound applications, composed by set of different layers such as DB, web server and others, which implies the deployment of software not just in an unique VM but in several ones. The environment template is the specification of the set of VMs plus the applications to be installed on top of. The user can define environment templates from scratch or reuse and customize them (e.g. including keypairs). Environment templates not only cover instantiation, but also the specification of such templates, store them in a catalogue and clone them from the abstract template catalog. In this way, an abstract environment template catalogue as well as a environment template one will be stored in the database, storing templates shared among all users and individual ones.

1.2 Proposed change

In order to fullfill this functionality, a new entity can be introduced in the MURANO database. It is the Murano environment-template, which contains the specification about what is going to be deployed in terms of virtual resources and application information, and it can be deployed on top of Openstack by translating it into environment. This environment template can be created, deleted, modified and customized by the users. In fact, it can be instantiate as many times as the user wants. For instance, the user wants to have different deployments from the same environment template: one for testing and another for production.

The environment template is composed by a set of services/applications with the software to be installed together their properties to work with. This software can be instantiate over an virtual service.

In this case the workflow for the creation and the instantiation of the environment template will imply: 1.- Creation of the environment template (including application information) 2.- Transformation of the environment template into the environment (creation of the environment, session and adding applications to the environment 3.- Deploy the environment on top of Openstack.

The environment template structure and information will be similar to the environment, not including some hardware information, like default network, or virtual server name. Mainly, the environment template information will contain: - template name, the name for the template. In case, it is not provided, the request will not be valid. - services, the application information. For each service it will include information about the applications to be installed (like tomcat), including application properties like tomcat port. In addition, in case applied, the information about the virtual server (instance) will be incorporated like keyname, flavor, image and so on. The following lines show an environment template example.

```
{
        "name": "env_template_name",
        "services": [
        {
            "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",
                "port": "8080",
                "?": {
                    "type": "io.murano.apps.apache.Tomcat",
                    "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
                }
            }
        ]
    }
```

1.2.1 Alternatives

None

1.2.2 Data model impact

A environment template entity will be introduced in the MURANO object model. This implies its existence in the database, the inclusion of Template services and the extension of the API. The template entity can consist on:

template :- murano:property(temp_id, “created”, datetime) murano:property(temp_id, “updated”, datetime) murano:property(temp_id, “id”, ID) murano:property(temp_id, “name”, varchar) murano:property(temp_id, “tenant-id”, varchar) murano:property(temp_id, “version”, bigint) murano:property(temp_id, “description”, text) murano:property(temp_id, “networking”, text)

1.2.3 REST API impact

The inclusion of the environment-template entity will imply the extension of the API for the environment-template creation, deletion, updating and translate into the environment.

POST /templates

Request

Method	URI	Description
POST	/templates	Create a new environment template

Content-Type application/json

Example Payload This template description can be composed just by the environment template name, or it can include the description of all services/applications to be deployed. 1.- Just the template

```
{
    'name': 'env_template_name'
}
```

2.- Specification of all the services

```
{
    "name": "env_template_name",
    "services": [
    {
        "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

```
{
   "updated": "2015-01-26T09:12:51",
   "networking":
   {
   },
   "name": "template_name",
   "created": "2015-01-26T09:12:51",
   "tenant_id": "00000000000000000000000000000001",
   "version": 0,
   "id": "aa9033ca7ce245fca10e38e1c8c4bbf7",
}
```

+———————+———————————————————————+ | Code | Description | +=====================+===+ | 200 | OK. Environment Template created successfully | +———————-+———————————————————————+ | 401 | User is not authorized to access this session | +———————-+———————————————————————+ | 409 | The environment template already exists | +———————-+———————————————————————+

GET /templates/{env-temp-id}

Request

Method	URI	Description
GET	/templates/{env-temp-id}	Obtains the environment template information

Parameters:

- *env-temp-id* - environment template ID, required

Content-Type application/json

Response


```
{
   "updated": "2015-01-26T09:12:51",
   "networking":
   {
   },
   "name": "template_name",
   "created": "2015-01-26T09:12:51",
   "tenant_id": "00000000000000000000000000000001",
   "version": 0,
   "id": "aa9033ca7ce245fca10e38e1c8c4bbf7",
}
```


+———————+———————————————————————+ | 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 /templates/{env-temp-id}

Request

Method	URI	Description
DELETE	/templates/<env-temp-id>	Delete the template id

Parameters:

- *env-temp_id* - environment template ID, required

Response

Code	Description
200	OK. Environment Template deleted successfully
401	User is not authorized to access this session
404	Not found. Specified environment template doesn‘t exist

POST /templates/{template-id}/services

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"
}
```

Code	Description
200	OK. Application added successfully
401	User is not authorized to access this session
404	The environment template does not exit

GET /templates/{env-temp-id}/services* 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"
        }
    }
]
```

Code	Description
200	OK. Tier created successfully
401	User is not authorized to access this session
404	The environment template does not exit

POST /templates/{env-temp-id}/create-environment

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",
}
```

+——————+————————————————————+ | 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 | +——————-+——————————————————+

1.2.4 Versioning impact

Murano client will change to include this new functionality.

1.2.5 Other end user impact

As well as a change in the API to include this new entity, the python-muranoclient will be changed for including the environment template operations. * env-template-create Create an environment template. * env-template-delete Delete an environment template. * env-template-list List the environment templates. * env-template-rename Rename an environment template. * env-template-show Show the information of the environment template * env-template-add-app Add an application to the environment template * env-template-create-environment It creates an environment from the environment template description

1.2.6 Deployer impact

None

1.2.7 Developer impact

None

1.2.8 Murano-dashboard / Horizon impact

New views will be required for including the environment template functionality

1.3 Implementation

1.3.1 Assignee(s)

Primary assignee: hmunfru

Other contributors: jesuspg TBC

1.3.2 Work Items

1.- Including the environment template entity in database 2.- Extension of the API for environment template catalogue 3.- Generation of environment from template operation 4.- Implement the changes in murano CLI

1.4 Dependencies

1.5 Testing

TBD

1.6 Documentation Impact

Environment template documentation should be included.

1.7 References

https://etherpad.openstack.org/p/GLLAQ0m1H7

CHAPTER 2

Configuration Language Support

https://blueprints.launchpad.net/murano/+spec/conf-language-support

2.1 Problem description

There is a huge community of applications (opscode, puppet-labs) where deployment installation instructions are specified in configuration languages such as Puppet or Chef. In order to reuse these applications, adaptors like Chef or Puppet are required on the VM side. Both chef and puppet recipes will not be managed by centralized server (chef-server, puppet-master), but they will use the standalone version, specifically the usage of chef-solo and puppet apply.

2.2 Proposed change

Inclusion of new executors in the murano-agent project. These executors will be objects to be used by murano-agent. Specifically, two executors will be implemented: Puppet and Chef. Both executors will be in charge of:

- Obtaining the required modules or cookbooks in the virtual machine.

This task can be done by passing the information from murano-engine to murano-agent, obtaining the information from the package itself. It requires the user to upload the package information plus the cookbooks to be used. The second option implies the cookbooks are downloaded in the virtual machine, so that they only need the URL to be accessible.

- Generating the required files for the configuration language.

For instance, manifests and hiera data for puppet, and, node specifications for chef from the information stored in the execution plan.

- Executing the chef-solo or puppet-apply process.

Previously, some work has to be done to install Chef or Puppet inside the VM. This task can be done by using cloud-init from the murano engine. The following is an example on how these executors work:

```
## YAML Template.
---
FormatVersion: 2.0.0
Version: 1.0.0
Name: Deploy Tomcat
Parameters:
    port: $port
Body: |
```

```
    return deployTomcat(port=args.port).stdout
Scripts:
    deployTomcat:
        Type: Chef
        Version: 1.0.0
        EntryPoint: mycoockbook::myrecipe
        Files:
            tomcat:
                Name: tomcat
                URL: git://github.com/opscode-cookbooks/tomcat.git
                Type: Downloadable
            java:
                Name: java
                URL: git://github.com/opscode-cookbooks/java.git
                Type: Downloadable
            ssl:
                Name: openssl
                URL: https://github.com/opscode-cookbooks/ssl.git
                Type: Downloadable
Options:
    captureStdout: true
    captureStderr: true
```

In this case, a new script Type appears (instead of Application). It is Chef type, which will execute the Chef executor. The same happens with the Puppet Type. In addition, the EntryPoint contains the information about the cookbook and the recipe to be installed. The Files section is used for the cookbooks and its dependence information. The cookbooks properties are in the Parameter section.

All the required steps to be part of the executor can be summarized as follows.

For Chef,

1. Creating the node.json with the recipes and the configuration parameters:

   ```
   {
       orion::ports: 1026
       orion::dbname: oriondb
       "run_list": [
           "recipe[orion::0.13.0_install]"
       ]
   }
   ```

2. **Executing chef-solo:** chef-solo -j node.json

For puppet,

1. Generating the manifest (site.pp):

   ```
   node 'default'
   {
       class{
           'orion::install':
       }
   }
   ```

2. Creating the hiera data information: hiera.yaml:

   ```
   ## YAML Template.
   ---
   orion::port: 1026
   orion::dbname: oriondb
   ```

3. **Executing:** puppet apply –hiera_config=hiera.yaml –modulepath=/opt/puppet/modules/orion site.pp

2.2.1 Alternatives

None

2.2.2 Data model impact

None

2.2.3 REST API impact

None

2.2.4 Versioning impact

None

2.2.5 Other end user impact

None

2.2.6 Deployer impact

The solution proposed is valid for any VM which contains the configuration language implementation already installed. There are event chef-solo and puppet agents for Windows.

2.2.7 Developer impact

None

2.2.8 Murano-dashboard / Horizon impact

None

2.3 Implementation

2.3.1 Assignee(s)

Primary assignee: hmunfru

Other contributors: jesuspg

2.3.2 Work Items

1. Generate Chef executor
2. Generate Puppet executor
3. Work on configuration

2.4 Dependencies

None

2.5 Testing

Integration tests will be done

2.6 Documentation Impact

Information about how to defines application for Puppet and Chef will have to be documented, explaining the different fields.

2.7 References

- http://es.slideshare.net/hmunfru/fiware-and-murano-support-for-configuration-languages
- https://etherpad.openstack.org/p/conf-language-support-spec

CHAPTER 3

Policy Guided Fulfillment - Congress Support in Murano

URL of launchpad blueprint:

https://blueprints.launchpad.net/murano/+spec/congress-support-in-murano

3.1 Problem description

As a part of policy guided fulfillment we need to call Congress from Murano, to enforce the policy on Murano environment. For current release the enforcement will be done by Congress simulation API, where Murano will query table *predeploy_error* in *murano_system* policy for passed update sequence created by mapping of environment into congress schema.

3.2 Proposed change

We need to provide

- python congress client in Murano

  We will use simulation feature of congress API. Using this API Murano will send decomposed Murano environment to Congress tables, so simulation can evaluate *predeploy_error* rule without changing existing data in Congress.

- mapping of Murano Environment into *update sequence* used in simulation API

3.2.1 Murano To Congress Mapping Details

This section provides congress schema created for Murano environment mapping. It will be created by Congress datasource drivers - see murano driver spec .

- **Policies**
  - **murano**

    Dedicated policy for Murano data.
  - **murano_system** Dedicated policy for rules
- **Schema**
  - *murano:objects(obj_id, owner_id, type)*

    This table holds every MuranoPL object instance in an environment.

    - *obj_id* - uuid of the object as used in Murano
    - *owner_id* - uuid of the owner object as used in Murano
    - *type* - string with full type indentifier as used in Murano (e.g., io.murano.Environment,...)

- *murano:parent_types(obj_id, parent_type)*

  This table holds parent types of *obj_id* object. Note that Murano supports multiple inheritance, so there can be several parent types for one object

- *murano:properties(obj_id, name, value)*

  This table stores object's properties. For multiple cardinality properties, there can be number of records here. MuranoPL properties referencing class type (i.e., another object) are stored in *murano:relatinoship*. Properties with structured content will be stored component by component.

- *murano:relationships(src_id, trg_id, name)*

  This table stores relationship between objects (i.e., MuranoPL property to *class*). For multiple cardinality relationships several records should be stored.

- *murano:connected(src_id, trg_id)*

  This table stores tuples of objects connected directly and indirectly via relationship. It is necessary since Congress does not support recursive rules yet.

- *murano:states(end_id, state)*

  This table stores *EnvironmentStatus* of Murano environment (one of 'ready', 'pending', 'deploying', 'deploy failure', 'deleting', 'delete failure').

3.2.2 Alternatives

None

3.2.3 Data model impact

None

3.2.4 REST API impact

None

3.2.5 Versioning impact

None

3.2.6 Other end user impact

None

3.2.7 Deployer impact

None

3.2.8 Developer impact

None

3.2.9 Murano-dashboard / Horizon impact

None

3.3 Implementation

3.3.1 Assignee(s)

Primary assignee: filip-blaha

Other contributors: ondrej-vojta, radek-pospisil

3.3.2 Work Items

1. Introduce Congress python client into Murano
2. Implement mapping of Murano Environment into Congress simulation API update-sequence.
3. Provide tests

3.4 Dependencies

- Congress python client (GIT) will be added into Murano.
- *Murano datasource driver* in Congress murano driver spec

3.5 Testing

Testing will use predefined Congress policy rules in order to test client and mapping. See https://etherpad.openstack.org/p/policy-congress-murano-spec for an example mapping and test.

3.6 Documentation Impact

Documentation impact is specified in Policy Enforcement Point blueprint.

3.7 References

- *Murano datasource driver* in Congress https://blueprints.launchpad.net/congress/+spec/murano-driver
- https://blueprints.launchpad.net/murano/+spec/policy-enforcement-point
- https://etherpad.openstack.org/p/policy-congress-murano-spec

CHAPTER 4

Provide opportunity to manage application categories

https://blueprints.launchpad.net/murano/+spec/enable-category-management

Murano is an application catalog, where new applications can be easily added. Those applications may belong to a category, that is not on predefined list.

Also, some categories may not be needed, and user can delete such kind of categories. All operations should be available only for admin users.

4.1 Problem description

Category management should be provided during:

- Packages uploading in dashboard;
- Packages modifying in dashboard;
- Packages uploading or modifying via command line;

Adding new category:

- Category name may contain spaces and other special characters;
- Category name limit is equal to 80 characters;
- Only admin users can add new category;

Deleting category:

- Category may be deleted only if no packages belong to this category;
- Only admin users can delete category;

4.2 Proposed change

Changes required to support this feature:

- Add new panel named 'Categories' under 'Manage' section; It should be available only for admin users. This panel should represent table with categories. The table will contain 'name' column and 'assigned packages' column. "Add category" will be a table action and "Delete Category" will be row action. Delete button should be hidden for those categories connected to the packages.
- Provide corresponding methods in the python-muranoclient; Category manager will be added to v1 module;

4.2.1 Alternatives

Admin can edit a database to add or delete categories.

4.2.2 Data model impact

None

4.2.3 REST API impact

GET /catalog/categories

The previous call /catalog/packages/categories is will still be valid to support backward compatibility.

Request

Method	URI	Description
Get	/catalog/categories	Get list of existing categories

Response

```
{"categories": [
        "id": "3dd486b1e26f40ac8f35416b63f52042",
        "updated": "2014-12-26T13:57:04",
        "name": "Web",
        "created": "2014-12-26T13:57:04",
        "package_count": 0
        },
        {
        "id": "k67gf67654f095gf89hjj87y56g98965v",
        "updated": "2014-12-26T13:57:04",
        "name": "Databases",
        "created": "2014-12-26T13:57:04",
         "package_count": 0
        }]
}
```

GET /catalog/categories/<category_id>

Request

Method	URI	Description
Get	/catalog/categories/<category_id>	Get category detail

Parameters

- *category_id* - category ID, required

Response

```
{
    "id": "0420045dce7445fabae7e5e61fff9e2f",
    "updated": "2014-12-26T13:57:04",
    "packages": [
        "Apache HTTP Server",
        "Apache Tomcat",
        "PostgreSQL"
    ],
    "name": "Web",
```

```
    "created": "2014-12-26T13:57:04"
}
```

Code	Description
200	OK. Category retrieved successfully
401	User is not authorized to access this session
404	Not found. Specified category doesn‘t exist

POST /catalog/categories

Attribute	Type	Description
name	string	Category name

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",
}
```

Code	Description
200	OK. Category created successfully
401	User is not authorized to access this session
403	Forbidden. Category with specified name already exist

DELETE /catalog/categories

Request

Method	URI	Description
DELETE	/catalog/categories/<category_id>	Delete category with specified id

Parameters:

- *category_id* - 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. Only empty categories can be removed

4.2.4 Versioning impact

Murano dashboard will support only the version of the client, that includes corresponding changes in the client. ‘Categories’ panel will not work with the old murano version, but application catalog and package management will work fine.

4.2.5 Other end user impact

None

4.2.6 Murano-dashboard / Horizon impact

Category management will be available in dashboard. Areas to be changed: (described in sections above)

- Manage section will have new panel;

4.2.7 Deployer impact

None

4.2.8 Developer impact

None

4.3 Implementation

4.3.1 Assignee(s)

Ekaterina Chernova

Primary assignee: <efedorova@mirantis.com>

4.3.2 Work Items

- Introduce 2 additional calls in API
- Update API specification
- Provide these calls in python-muranoclient
- Implement changes in dashboard
- Enable CLI to manage categories

4.4 Dependencies

None

4.5 Testing

New tests should be added in dashboard integration tests

4.6 Documentation Impact

API specification should be updated. All changes are already represented here, just need to copy.

4.7 References

None

CHAPTER 5

Add timeouts to murano-agent calls

https://blueprints.launchpad.net/murano/+spec/murano-agent-timeouts

Now there is no way to be sure that the agent successfully started execution on a VM. Also there is no control of the execution time of scripts on agent. This process should be more controllable. It can be done by adding timeouts in Murano engine.

5.1 Problem description

- During the agent‘s work could be some problems with execution of scripts and VM may hang. In this case user will wait indefinitely without knowing what’s happend.
- Currently there is no feedback from agent so, it‘s impossible to determine whether agent is ready to accept execution plans or not.

It is proposed to provide mechanism of timeouts which solve these problems.

5.2 Proposed change

1. First of all add timeout to method *call* of agent on engine-side.

Optional parameter *timeout* will be added to method *call* of class *Agent* in agent.py. This parameter is the time in seconds with default value *600*. Developer can set up custom value during developing apps for example in this way:

```
- $.instance.agent.call($template, $resources, 300)
```

If the agent plan execution time exceeds the limit, it will be terminated.

2. Add method *waitReady* in Agent class.

Method will be added to *Agent* class in agent.py. It has optional parameter *timeout* with default value *100*. *waitReady* creates test plan with trivial body:

```
template = {'Body': 'return', 'FormatVersion': '2.0.0', 'Scripts': {}}
```

and sends this plan once to agent by method *call*. It can be uses by developer to stop deployment before sending template of app if agent is inaccessible as follows:

```
- $.instance.agent.waitReady()
- $.instance.agent.call($template, $resources)
```

If the agent test plan execution time exceeds the time limit, *TimeoutException* will be raised and deployment terminates. *TimeoutException* will be created in murano/common/exceptions.py.

3. Add new method *isReady* in Agent class.

This method will be simply call the *waitReady*. Method *isReady* returns:

- *True*, if test plan is executed on time;
- *False*, if the agent plan execution time exceeds the limit.
- and raise *PolicyViolationException*, which will be created in murano/coommon/exceptions,py, if the agent disabled by the server.

The method can be used during development Murano-applications. For example, developer can check if the agent is running and ready to execute templates before sending the execution plan of application:

```
- If: $.instance.agent.isReady()
  Then:
    - $._environment.reporter.report($this, 'Murano Agent is ready')
```

The message in above example will be reported to Murano Dashboard.

5.2.1 Alternatives

None

5.2.2 Data model impact

None

5.2.3 REST API impact

None

5.2.4 Versioning impact

None

5.2.5 Other end user impact

None

5.2.6 Deployer impact

None

5.2.7 Developer impact

None

5.2.8 Murano-dashboard / Horizon impact

None

5.3 Implementation

5.3.1 Assignee(s)

Primary assignee:lk ddovbii

5.3.2 Work Items

The change is simple and can be done in one Gerrit patch. Implementation is acutally completed.

5.4 Dependencies

None

5.5 Testing

Unit and integration tests must be done.

5.6 Documentation Impact

MuranoPL specification should be updated.

5.7 References

None

CHAPTER 6

murano-mistral-integration

https://blueprints.launchpad.net/murano/+spec/murano-mistral-integration-1

The purpose is to add integration between Murano and Mistral. We would like to allow invocation of a Mistral workflow from Murano. No prerequisite for uploaded Mistral workflow.

6.1 Problem description

A new capability in Murano modeling process should be added in order to answer several use cases, including:

1. The application modeler wishes to leverage existing workflow that deploys a specific component as part of a new application model creation process.
2. The application modeler wishes to add post-deployment logic to an application model.

   For example:

   (a) Check that the application has been successfully deployed.

   (b) Inject initial data to the deployed application.

6.2 Proposed change

Adding a new system class for Mistral Client that allows to call Mistral APIs from the Murano application model.

The system class will allow you to:

1. Upload a Mistral workflow to Mistral.
2. Trigger the already-deployed Mistral workflow, wait for completion and return the execution output.

6.2.1 Alternatives

None

6.2.2 Data model impact

None

6.2.3 REST API impact

None

6.2.4 Versioning impact

None

6.2.5 Other end user impact

None

6.2.6 Deployer impact

None

6.2.7 Developer impact

None

6.2.8 Murano-dashboard / Horizon impact

None

6.3 Implementation

6.3.1 Assignee(s)

Primary assignee: Natasha Beck

6.3.2 Work Items

1. Add mistral client system class that can trigger pre-deployed Mistral workflow.
2. Add ability to the client to upload the Mistral workflow.
3. Add test that deploys application which uploads the Mistral workflow from Resources, triggers it and gets output as a result.

6.4 Dependencies

Openstack Mistral component.

6.5 Testing

Functional test will be added to Murano. The test deploys application which uploads the Mistral workflow from Resources, triggers it and gets output as a result.

6.6 Documentation Impact

Murano-Mistral integration must be documented from the following perspectives:

- Setup configuration (for example with and without mistral)
- Include usage of Mistral workflow as a part of an application model

The following Murano documentation will be affected:

- **Murano Installation Guide** Add section on Mistral requirement
- **Murano Workflow** Add section on Murano-Mistral integration
- **Murano Article (new)** Article on Murano-Mistral integration

6.7 References

None

CHAPTER 7

Murano Repository Support

https://blueprints.launchpad.net/murano/+spec/muranoclient-marketplace-support

Murano applications provide a powerful and flexible way to move workloads from other cloud environments to OpenStack. In order to accelerate application migrating we need a way to deliver Murano packages and Glance images to customers incrementally, independent from major releases of OpenStack.

7.1 Problem description

Typical use cases:

- After the end users installed and configured Murano they would need to install murano-enabled applications. To do so they would use *murano* CLI client. Invoking a command like *murano install-package io.apps.application –version=2.0* would install the application in question alongside with all requirements (applications and glance images)
- A developer would want to provide murano applications to end users, by hosting them on a web server and providing http-access to application packages. In that case end users would be able to install the application by invoking a CLI command *murano install-package http://path/to/app*
- End user would want to install bundles of applications that are often required to work together, but not necessarily depend on each other. In that case end user would invoke a command *murano install-bundle bundle_name* and murano-client would download all the applications mentioned in the bundle alongside with their requirements.
- End users would want to perform same operations through murano-dashboard instead of CLI tools.

7.2 Proposed change

The proposition is to implement a set of features in python-muranoclient and murano-dashboard, that would allow end users to install applications from http application repository.

- Enable *package-import* command to support url as a parameter for import by modifying the way package creation of murano-client works.
- Enable *package-import* command to support package name as a parameter for import, introduce *–murano-repo-url* as a base path to the repository.
- Introduce *bundle-import* command and allow it to support local-files, urls and bundle names as parameters for import. The command should parse the bundle file and import all the packages, mentioned in the file. Bundle should be a simple json and/or yaml-structured file.

- Introduce *Require:* section to *manifest.yaml*, that would specify which packages are required by the appication.
- Enable package creation suite to automatically import all the packages mentioned in *Require:* section.
- Introduce *images.lst* file in the package structure, that would contain a list of images, required for the application to work.
- Enable package creation suite to automatically import required images into glance image service.
- Allow *bundle-import* command to import bundles from local-files. In that case first search for package files and image files in the local filesystem, relative to the location of the bundle file. If file is not found locally attempt to connect to the repository.
- Enable murano-dashboard to support changes made to client and introduce a way to upload packages via url/name to dashboard.
- Implement importing of bundles in murano-dashboard by url or by name. Since bundles are currently a simple json/yaml optionally we could optionally support direct input.
- Implement error handling both for CLI-tools and dashboard, that would inform end users about any errors that might have happened along the way of importing.
- Optionally implement a progress marker and a ETA-marker for CLI *import* commands.

7.2.1 Alternatives

Implementing a server, that would hold versions and paths to package files and implementing a client to that server might be a good alternative to a simple http-server, although it seems a bit excessive at the moment.

Another idea would be to implement a service, that would download packages asynchronously. This would allow client to download large package files and image files with possibly better error handling mechanisms. This also seems a bit excessive at the moment.

7.2.2 Data model impact

It might be a good idea to store information about required apps and required images, although it is not strictly required for the task.

7.2.3 REST API impact

It might be a good idea to warn the end user if the client installed a package, that depends on other packages, not present in app catalog, although it is not strictly required for the task.

7.2.4 Versioning impact

This proposition adds functionality both to python-muranoclient and to murano-dashboard. It should be fully backward compatible.

Minor version of python-muranoclient should be increased, because we add functionality.

7.2.5 Other end user impact

See *Proposed Change* section, as it describes all the ways users would interact with the feature

7.2.6 Deployer impact

None

7.2.7 Developer impact

None

7.2.8 Murano-dashboard / Horizon impact

Package import dialog should be changed to reflect different ways of importing an application. Additional bundle-import dialog should be implemented.

7.3 Implementation

7.3.1 Assignee(s)

Primary assignee: <kzaitsev@mirantis.com>

7.3.2 Work Items

- Add support for importing packages by url in client
- Add support for importing packages by name in client
- Add support for importing bundles by name/url in client
- Add support for recursive parsing of Require section of *manifest.yaml* file in client
- Add support for image download/upload from *images.lst* file in client
- Handle *package exists* error in CLI client
- Add support for different ways to import a package in murano-dashboard
- Add support for bundle-import in murano-dashboard
- Add support for image/package requirements while importing packages and bundles in murano-dashboard

7.4 Dependencies

None

7.5 Testing

Unit testing should be sufficient to cover most of the use cases. However an integration test, that would setup a simple repository is very desirable, since we add changes to both *python-muranoclient* and *muranodashboard*.

7.6 Documentation Impact

Additional capabilities of the CLI client should be documented, otherwise there should be no impact, since we do not change the API.

7.7 References

None

CHAPTER 8

Example Spec - The title of your blueprint

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/murano/+spec/example

Introduction paragraph – why are we doing anything? A single paragraph of prose that operators can understand. The title and this first paragraph should be used as the subject line and body of the commit message respectively.

Some notes about using this template:

- Your spec should be in ReSTructured text, like this template.
- Please wrap text at 79 columns.
- The filename in the git repository should match the launchpad URL, for example a URL of: https://blueprints.launchpad.net/murano/+spec/awesome-thing should be named awesome-thing.rst
- Please do not delete any of the sections in this template. If you have nothing to say for a whole section, just write: None
- For help with syntax, see http://sphinx-doc.org/rest.html
- To test out your formatting, build the docs using tox, or see: http://rst.ninjs.org
- If you would like to provide a diagram with your spec, ascii diagrams are required. http://asciiflow.com/ is a very nice tool to assist with making ascii diagrams. The reason for this is that the tool used to review specs is based purely on plain text. Plain text will allow review to proceed without having to look at additional files which can not be viewed in gerrit. It will also allow inline feedback on the diagram itself.
- If your specification proposes any changes to the Murano REST API such as changing parameters which can be returned or accepted, or even the semantics of what happens when a client calls into the API, then you should add the APIImpact flag to the commit message. Specifications with the APIImpact flag can be found with the following query:

  https://review.openstack.org/#/q/status:open+project:stackforge/murano-specs+message:apiimpact,n,z

8.1 Problem description

A detailed description of the problem:

- For a new feature this might be use cases. Ensure you are clear about the actors in each use case: End User vs Deployer
- For a major reworking of something existing it would describe the problems in that feature that are being addressed.

8.2 Proposed change

Here is where you cover the change you propose to make in detail. How do you propose to solve this problem?

If this is one part of a larger effort make it clear where this piece ends. In other words, what's the scope of this effort?

8.2.1 Alternatives

What other ways could we do this thing? Why aren't we using those? This doesn't have to be a full literature review, but it should demonstrate that thought has been put into why the proposed solution is an appropriate one.

8.2.2 Data model impact

Changes which require modifications to the data model often have a wider impact on the system. The community often has strong opinions on how the data model should be evolved, from both a functional and performance perspective. It is therefore important to capture and gain agreement as early as possible on any proposed changes to the data model.

Questions which need to be addressed by this section include:

- What new data objects and/or database schema changes is this going to require?
- What database migrations will accompany this change.
- How will the initial set of new data objects be generated, for example if you need to take into account existing instances, or modify other existing data describe how that will work.

8.2.3 REST API impact

Each API method which is either added or changed should have the following

- Specification for the method
  - A description of what the method does suitable for use in user documentation
  - Method type (POST/PUT/GET/DELETE)
  - Normal http response code(s)
  - Expected error http response code(s)
    - A description for each possible error code should be included describing semantic errors which can cause it such as inconsistent parameters supplied to the method, or when an instance is not in an appropriate state for the request to succeed. Errors caused by syntactic problems covered by the JSON schema definition do not need to be included.
  - URL for the resource
  - Parameters which can be passed via the url
- Example use case including typical API samples for both data supplied by the caller and the response
- Discuss any policy changes, and discuss what things a deployer needs to think about when defining their policy.

8.2.4 Versioning impact

Discuss how your change affects versioning and backward compatibility:

- Can it break any existing DSL code even in theory?
- Can it break API consumers that are using older python-muranoclient versions or non-python clients?
- If you make changes to Murano package please state how the version number should be incremented?
- Does your change require newer version of external or internal component?
- How to keep backward compatibility with code and consumers that were available prior to your change?

8.2.5 Other end user impact

Aside from the API, are there other ways a user will interact with this feature?

- Does this change have an impact on python-muranoclient? What does the user interface there look like?

8.2.6 Deployer impact

Discuss things that will affect how you deploy and configure OpenStack that have not already been mentioned, such as:

- What config options are being added? Should they be more generic than proposed (for example a flag that other hypervisor drivers might want to implement as well)? Are the default values ones which will work well in real deployments?
- Is this a change that takes immediate effect after its merged, or is it something that has to be explicitly enabled?
- If this change is a new binary, how would it be deployed?
- Please state anything that those doing continuous deployment, or those upgrading from the previous release, need to be aware of. Also describe any plans to deprecate configuration values or features. For example, if we change the directory name that instances are stored in, how do we handle instance directories created before the change landed? Do we move them? Do we have a special case in the code? Do we assume that the operator will recreate all the instances in their cloud?

8.2.7 Developer impact

Discuss things that will affect other developers working on OpenStack, such as:

- If the blueprint proposes a change to the driver API, discussion of how other hypervisors would implement the feature is required.

8.2.8 Murano-dashboard / Horizon impact

Does it require changes to the murano-dashboard / horizon? If so, changes should be described well. If it's about complex changes than probably a separate blueprint / spec should be created for it.

8.3 Implementation

8.3.1 Assignee(s)

Who is leading the writing of the code? Or is this a blueprint where you're throwing it out there to see who picks it up?

If more than one person is working on the implementation, please designate the primary author and contact.

Primary assignee: <launchpad-id or None>

Other contributors: <launchpad-id or None>

8.3.2 Work Items

Work items or tasks – break the feature up into the things that need to be done to implement it. Those parts might end up being done by different people, but we're mostly trying to understand the timeline for implementation.

8.4 Dependencies

- Include specific references to specs and/or blueprints in murano, or in other projects, that this one either depends on or is related to.
- If this requires functionality of another project that is not currently used by Murano, document that fact.
- Does this feature require any new library dependencies or code otherwise not included in OpenStack? Or does it depend on a specific version of library?

8.5 Testing

Please discuss how the change will be tested. We especially want to know what tempest tests will be added. It is assumed that unit test coverage will be added so that doesn't need to be mentioned explicitly, but discussion of why you think unit tests are sufficient and we don't need to add more tempest tests would need to be included.

Is this untestable in gate given current limitations (specific hardware / software configurations available)? Is this untestable in murano-ci? If so, are there mitigation plans (3rd party testing, gate enhancements, etc).

8.6 Documentation Impact

What is the impact on the docs team of this change? Some changes might require donating resources to the docs team to have the documentation updated. Don't repeat details discussed above, but please reference them here.

8.7 References

Please add any useful references here. You are not required to have any reference. Moreover, this specification should still make sense when your references are unavailable. Examples of what you could include are:

- Links to mailing list or IRC discussions

- Links to notes from a summit session
- Links to relevant research, if appropriate
- Related specifications as appropriate (e.g. if it's an EC2 thing, link the EC2 docs)
- Anything else you feel it is worthwhile to refer to

CHAPTER 9

Plugable pythonic classes for Murano

https://blueprints.launchpad.net/murano/+spec/plugable-classes

One of the key features of Murano is extensibility, and we need to push this feature even further and give our customers a way to extend Murano with new functions (e.g. support for F5 BigIP API) in a drag-n-drop manner. This spec proposes a solution which will add this extensibility option.

9.1 Problem description

Currently all the functionality which is available to user is limited to the features of MuranoPL language which just provides data transformation capabilities and flow control primitives. The language itself does not contain any functions for I/O operations, hardware access or interaction with host operating system, other OpenStack or third-party services. This is an intentional design feature: MuranoPL code is provided by users and cannot be always trusted. All the external communications and low-level interactions are done via python code which is bundled with Murano Engine and is accessible to MuranoPL code via MuranoPL wrappers. Any interactions which are not supported by that Python classes are impossible.

Some deployment scenarios may need to extend this set of allowed low-level interactions. They may include some customer-specific logic, custom software bindings etc, so trying to bundle all of them into the standard Murano Engine classes is not a good idea. Instead, there should be a way to dynamically add extra interactions to any existing deployment of Murano without modifying its core components but rather with installing some plugin-like components.

Installing these plugins is supposed to be a maintainer-only operation, requiring administrative access to nodes running Murano services. It is supposed that the maintainer is always aware about the contents of the plugins and is able to verify them from security, performance and other sensible points of view.

9.2 Proposed change

It is proposed to implement each extension as independent Python Package built using setuptools library. Each package should define one or more entry-point in a specific namespace (`io.murano.extensions` is suggested). Each of this entry- points should export a class, which may be registered as MuranoPL class when the engine loads.

Each package should be installed on Murano nodes into the same Python environment with Murano engine service.

Murano will get a PluginLoader class which will utilize stevedore library [1] to discover classes registered as entry-points in `io.murano.extensions` namespace.

Murano Engine will use PluginLoader to register all the loaded plugins in its class loader (i.e. will call `import_class` with a class imported from the plugin as a parameter). As the result, the classes will become available for the MuranoPL code being executed in the Engine.

To prevent potential name collisions, MuranoPL names for the loaded classes will be assigned automatically: the name will consist of the namespace (`io.murano.extensions` as suggested above) and the name of entry-point. To guarantee this naming rule the imported classes should not define their MuranoPL names on their own (i.e they should not have `@murano_class.classname` decorators or other code which modifies their `_murano_class_name` field). If they do, the PluginLoader will discard that information and will log a warning message.

As the entry-point name will eventually become a name of MuranoPL class, the PluginLoader will validate it accordingly.

As neither stevedore nor setuptools enforce any uniqueness constraints on the entry-point names (i.e. several packages may define entry-points with the same name within a same namespace, and all of them will be correctly loaded by stevedore), then this enforcement should also be done by the Murano's Plugin Loader. If two or more plugin packages attempt to register classes with the same endpoint name, then a warning will be logged and no classes from all the conflicting packages will be loaded.

PluginLoader will also ensure that objects being exported in these entry-points are indeed classes, and will check if they define a classmethod called `init_plugin`. If such method exists, the PluginLoader will execute it before loading it.

The plugins which are already installed in the environment may be prevented from being loaded by a configuration option. This new option called `enabled_plugins` will be added to `murano.conf`. If it has its default value `None` or is omitted from the config, there will be no restriction on the plugins which are being loaded (any plugin registered within the environment will be loaded). If it exists and is not None, then it is expected to contain a list of names of the packages from which the plugins will be loaded. If the package is not mentioned there, all its endpoints will be ignored and no classes from it will be imported. Empty value of `enabled_plugins` will mean that no plugins may be loaded and only bundled system classes are accessible from the MuranoPL code.

The `enabled_plugins` setting will be implemented using `EnabledExtensionManager` class of the stevedore library [2], so the disabled plugins will be excluded from entry-point name analysis. Thus if there are plugins which define conflicting entry-point names, then the conflict may be resolved with this setting instead of uninstalling the plugin from the environment.

Currently stevedore is unable to load packages which were installed after the start of the current process. So, in current proposal it is required to restart Murano services after plugin package is installed, removed or upgraded and after the changing of `enabled_plugins` value in configuration file. As the restart of the services is not a good thing for production solutions, it may be a good idea to design a "graceful restart" solution which will make the service to stop listening for incoming requests, finish its current tasks and then exit and restart, loading the updated configuration and plugins. However such solution is out of scope of the current spec and is left for future blueprints.

9.2.1 Alternatives

Instead of using stevedore to discover and load the plugins, some home-made solution may be invented to load Python modules from some directory. This solution may have its benefits (e.g. it does not require restarts to load new plugins), however stevedore is currently a de-facto standard for building plugable solutions in Openstack, so it is suggested to use it.

9.2.2 Data model impact

This proposal does not affect data model.

9.2.3 REST API impact

N/A

9.2.4 Versioning impact

N/A

9.2.5 Other end user impact

N/A

9.2.6 Deployer impact

The change itself does not have any immediate impact on deployer: a new configuration option is optional and has meaningful default. However registering new plugins will require to restart the Services, which may bring up some concerns in production environments.

9.2.7 Developer impact

Developers who build their own plugins should be aware about setuptools entry- points and should inherit their exported classes from *murano.dsl.murano_object.MuranoObject*.

9.2.8 Murano-dashboard / Horizon impact

No immediate changes required.

9.3 Implementation

9.3.1 Assignee(s)

Primary assignee: ativelkov

9.3.2 Work Items

- Implement the PluginLoader class
- Modify MuranoEngine to register plugin-imported classes in class loader.

9.4 Dependencies

This requires stevedore library as a dependency. It is already part of OpenStack Global Requirements, so no problems are expected.

9.5 Testing

The unit-tests have to cover PluginLoader class using the make_test_instance method of stevedore.

Separated tests should cover API method call.

Tempest tests are out of the scope of this spec.

9.6 Documentation Impact

There should be created a "Plugin developer's Manual" which will describe the process of plugin package creation.

9.7 References

[1] http://docs.openstack.org/developer/stevedore/ [2] http://docs.openstack.org/developer/stevedore/managers.html#enabledextensionm

CHAPTER 10

Policy Guided Fulfillment - Policy Enforcement Point

URL of launchpad blueprint:

https://blueprints.launchpad.net/murano/+spec/policy-enforcement-point

10.1 Problem description

As a part of policy guided fulfillment we need to implement *predeploy* policy enforcement point - i.e., Murano calls Congress to evaluate *predeploy* policy rules on data representing Murano environment being deployed. If evaluation returns *predeploy error* data (i.e., enforcement failed), then deployment of Murano environment fails.

Predeploy policy rules are represented by *predeploy_error(env_id, obj_id, message)* congress table. It means

- Congress administrator is responsible for creating predeploy rules, which has this table on the left side (see examples).
- Murano is not involved in policy rule evaluation, except that Murano provides data about Murano environments. Thus user can use any data (e.g., datasource tables, policy rules) available in Congress to define when environment can be deployed.

Table *predeploy_error(env_id, obj_id, message)* reports list of found errors:

- *env_id* environment id where error was detected
- *obj_id* object id (in environment id) on which error was detected
- *message* text message of error

Murano environment serialization/decomposition is described in Congress Support in Murano and Murano Congress Driver

Example (generic):

```
predeploy_error(env_id, obj_id, message) :-
    murano:state(env_id,"PENDING"),
    my-rule-table1(env_id, obj_id),
    concat("", "some error message 1", message)

predeploy_error(env_id, obj_id, message) :-
    murano:state(env_id,"PENDING"),
    my-rule-table2(env_id, obj_id),
    concat("", "some error message 2", message)

my-rule-table1(env_id, obj_id) :- ....
```

```
my-rule-table2(env_id, obj_id) :- ....
```

Example (allow only environments where all VM instances has flavor with max 2048MB RAM):

```
predeploy_error(eid, oid, msg) :-
    murano:object(oid, eid, type),
    checkIfError(oid),
    concat("", "Instance flavor has RAM size over 2048MB", msg)

checkIfError(oid) :-
    murano:parent_type(oid, "io.murano.resources.Instance"),
    murano:property(oid, "flavor", fname),
    nova:flavors(i,fname,v,r,d,e,rx),
    gt(r,2048)
```

Enforcement point will use Congress simulation api to evaluate rules on passed mapped environment into Congress schema.

10.2 Proposed change

When user executes *deploy* action on an environment following steps will be done

- environment will be mapped into *update sequence*
- Congress simulation API will be executed:

```
openstack congress  policy simulate murano_system 'predeploy-error(envId, objId, message)'  'env+(100
```

- if response will contain non empty result, then deployment will fails

10.2.1 Alternatives

None

10.2.2 Data model impact

None

10.2.3 REST API impact

None

10.2.4 Versioning impact

None

10.2.5 Other end user impact

Environment deployment may fail due to policy validation failure.

10.2.6 Deployer impact

Policy enforcement will be used only if

- Enforcement is enabled in *murano.conf*
- Congress is available in Keystone catalog (i.e., it is deployed in OpenStack)

10.2.7 Developer impact

None

10.2.8 Murano-dashboard / Horizon impact

None

10.3 Implementation

10.3.1 Assignee(s)

Primary assignee: ondrej-vojta

Other contributors: filip-blaha, radek-pospisil

10.3.2 Work Items

1. Use implementation of *Congress Support in Murano* in order to implement policy enforcement point as advised by Stan (see below). The Congress support must correctly deal with following setups

- Openstack with Congress installed
- Openstack without Congress

```
Stan: such approach makes PolicyEnforces (and thus dependency on Congress) be   mandatory for Murano
at https://github.com/stackforge/murano/blob/master/murano/common/engine.py#L112
something like

if config.CONF.enable_policy_enforcer:
     policyenforcer.validate(self.model)
```

2. Provide Developer and User Documentation (see Documentation section).

10.4 Dependencies

- *Congress Support in Murano* https://blueprints.launchpad.net/congress/+spec/murano-driver
- *Murano datasource driver* in Congress https://blueprints.launchpad.net/congress/+spec/murano-driver
- *Policy enforcement specification* https://etherpad.openstack.org/p/policy-congress-murano-spec

10.5 Testing

Unit and integration tests must be done.

Integration tests must cover following setups

- **Openstack has Congress installed**
  - situations when Congress is running (i.e., responding) and not running (i.e., not responding) must be tested
- Openstack has not Congress installed

10.6 Documentation Impact

Policy enforcement must be documented from following perspectives

- Setup configuration (e.g., with and without congress)
- Murano rules (i.e., Murano environment data decompositiion) in Murano policy
- How Murano policy affects environment deployment

Following Murano documentation will be affected

- **Murano Installation Guide** Add section on Congress requirement and section on enabling policy enforcement
- **Murano Workflow** Add section on Murano policy enforcement
- **Murano Article (new)** Article on Murano policy rules (e.g., Murano environment decomposition to Congress)

10.7 References

- *Congress Support in Murano* https://blueprints.launchpad.net/congress/+spec/murano-driver
- *Murano datasource driver* in Congress https://blueprints.launchpad.net/congress/+spec/murano-driver
- *Policy enforcement specification* https://etherpad.openstack.org/p/policy-congress-murano-spec

Liberty specs:

CHAPTER 11

Liberty specifications

This directory is supposed to hold approved specifications for the 'Liberty' release. You are welcome in contributing to Murano!

11.1 Murano

Murano Project introduces an application catalog, which allows application developers and cloud administrators to publish various cloud-ready applications in a browsable categorized catalog. Cloud users – including inexperienced ones – can then use the catalog to compose reliable application environments with the push of a button.

11.2 Murano Project Resources

- Murano Official Documentation
- Project status, bugs, and blueprints are tracked on Launchpad
- Additional resources are linked from the project Wiki page
- Python client

11.3 License

CHAPTER 12

Add support for heat environments

https://blueprints.launchpad.net/murano/+spec/add-support-for-heat-environments-and-files

Add support for using Heat environments, when saving and deploying Heat templates.

12.1 Problem description

Today there is no option to create stacks neither with an environment nor with nested stacks via Murano. Only basic stacks are supported.

Use cases for deploying a stack with an environment:

- Supporting resource registry
- Supporting saving 'test' and 'production' parameter profiles for the same template

12.2 Proposed change

The change described here will focus on supporting Heat environments. Another spec was submitted for Heat files.

In this spec no UI changes are included. A UI part that can be added in future specs, and in any case is out of the scope of this spec.

Allowing Heat environments to be saved and deployed as part of a Murano Heat package. Adding environments to a package should be optional. Heat environment files should be placed in the package under '/Resources/HotEnvironments' When a user request to add a Heat-generated package to a Murano environment, he should have an option to specify one of the hot-environment-files located at '/Resources/HotEnvironments' in the package, and this heat environment file should be sent to Heat with the template during Murano environment deployment. If no heat environment file was specified, the template will be deployed without a Heat environment. The stage where the Heat environment for the deployment should be specified is when adding a package to a Murano environment as part of a configuration session. The heat environment should be referenced by name.

In the future, there can be an API to list the hot environments of a package. This might be done by using the existing API of getting the UI of a package.

12.2.1 Alternatives

- The package structure can be different.

- Words that can replace the word environments to reference heat environments: configurations, profiles, settings. The problem is that in Heat they use the word environments.
- Specifying the heat environment to be deployed with the Heat template as part of a Murano environment when deploying the environment (instead of specifying it when adding a package to an environment). If we will wait to this point, we will have to give a map of packages and the environments to be deployed with them. this alternative requires more validations.

12.2.2 Data model impact

new data objects:

- hotEnvironment - This parameter will be passed as part of the /environments/{env-id}/services POST API request body. The value of this parameter will be an environment file name.
- templateParameters - All heat parameters that were passed in the root of the /environments/{env-id}/services POST API request body, will be moved under this property.

12.2.3 REST API impact

None

12.2.4 Versioning impact

None

12.2.5 Other end user impact

User will now have the option to add heat environments and additional files to the package, and have them deployed with the package as part of the Murano environment. The Heat environment will be deployed when it is requested in the relevant API, while the additional files will always be deployed.

At this point there will be no change in the python-muranoclient. If the user will wish to add environments or additional files to a heat generated package. He can edit the package and continue via API. If the user only added additional files, he can continue via python-muranoclient/UI as well.

12.2.6 Deployer impact

None

12.2.7 Developer impact

None

12.2.8 Murano-dashboard / Horizon impact

New features must not break UI. So since it was proposed to move the user defined parameters from the root of the request body to the property “templateParameters”, the UI must change the way it build the request to add a package to a Murano environment.

The new feature for sending a hot environment with the template will be exposed in the UI the following way: The user will be able to choose an environment file from a drop-down list during the same wizard actions he uses to enter heat template parameters.

12.3 Implementation

new data objects: * envs * env_name

envs will be generated when parsing the package, so if there is nothing in the package, the new data objects will be empty. If an API to list the environments will be implemented an empty list will return, same as in other Murano APIs.

A new object called envs of type dictionary will be added in HeatStack class in the init method. It can be referenced from inside HeatStack by using self._envs before/when sending the create request to Heat. This object will be initialized to be empty.

A new method called setEnvs will be added in HeatStack class. It will allow to enter a value into _envs just like setParameters allows to enter values into _parameters.

env_name will take the value passed by the user in the parameter heatEnvironment. It will be initialized and passed to class HeatStack the same way parameters are passed today.

When sending the create stack request to heat the selected environment content should be passed. If all is configured and passed correctly the environment file content can be accessed from class HeatStack by using the next command: self._envs[self._env_name]

A new method called _translate_envs will be add to HotPackage class. it will get a path to the envs directory and will return a dictionary of environments locations and files values. It will be in the next format: environmentRelativePathStartingHeatEnvironmentsDirectory -> stringFileContent For example if there is an environment with a full path of /Resources/HeatEnvironments/my_heat_env.yaml and content of: “heat_template_version: 2013-05-23nnparameters:n” and it is the only file in the folder, than this will be the dictionary returned: {“my_heat_env.yaml”: “heat_template_version: 2013-05-23nnparameters:n”}

A very similar function was proposed for the Heat files feature. There will be a reuse of the code there, if it will be implemented first.

12.3.1 Assignee(s)

Primary assignee: michal-gershenzon

Other contributors: noa-koffman

12.3.2 Work Items

- Add support for adding a package to a Murano environment with a Heat environment specified in the request.
- Add support for Heat environments when deploying a Murano environment. If a Heat environment is saved in the session for the package, it should be parsed and send with the template, when deploying a Murano environment.
- make sure that when ui generate a POST request for API: /environments/{env-id}/services the user defined parameters are located under templateParameters in the request body.

12.4 Dependencies

None

12.5 Testing

Unit tests should cover API calls changes:

- Test sending a Heat environment when adding a package to a Murano environment (positive and negative).
- Test that the request for Heat is build correctly with Heat environment

12.6 Documentation Impact

None

12.7 References

- http://docs.openstack.org/developer/heat/template_guide/environment.html

CHAPTER 13

Add support for heat environments and files

https://blueprints.launchpad.net/murano/+spec/add-support-for-heat-environments-and-files

Add support for using Heat additional files, when saving and deploying Heat templates.

13.1 Problem description

Today there is no option to create stacks neither with an environment nor with nested stacks via Murano. Only basic stacks are supported.

Use cases for deploying a stack with additional files:

- Supporting Heat nested stacks
- Supporting scripts

For more information see references section of this spec

13.2 Proposed change

The change described here will focus on supporting additional files. Another spec will be submitted for Heat environments.

Allowing Heat additional files to be saved and deployed as part of a Murano Heat applications. Adding additional files to a package should be optional. Such files should be placed in the package under '/Resources/HeatFiles'. When a Heat generated package is being deployed, if the package contains Heat files, they should be sent to Heat together with the template and params. If there are any Heat additional files located in the package under '/Resources/HeatFiles', they should all be sent with the template during stack creation in Heat.

There can be more nested directories under '/Resources/HeatFiles', and if they have files in them, those should be send to heat as well together with their relative path.

This part does not require any UI nor API changes.

13.2.1 Alternatives

- The package structure can be different.

13.2.2 Data model impact

None

13.2.3 REST API impact

None

13.2.4 Versioning impact

None

13.2.5 Other end user impact

User will have the option to add Heat files to the package, and have them deployed with the package as part of the Murano environment.

At this point there will be no change in the python-muranoclient. If the user will wish to add files to a heat generated package, he can edit the package and continue normally from there.

13.2.6 Deployer impact

None

13.2.7 Developer impact

None

13.2.8 Murano-dashboard / Horizon impact

None.

13.3 Implementation

new data objects: * files

This new data object will be generated when parsing the package, so if there is nothing in the package, the new data objects will be empty.

A new object called files of type dictionary will be added in HeatStack class in the init method. It can be referenced from inside HeatStack by using self._files when sending the create request to Heat. This object will be initialized to be empty.

A new method called setFiles will be added in HeatStack class. It will allow to enter a value into _files just like setParameters allows to enter values into _parameters.

A new method called _translate_files will be add to HotPackage class. it will get a path to the files directory and will return a dictionary of files locations and files values. It will be in the next format: fileRelativePathStartingHeatFilesDirectory -> stringFileContent For example if there is a file with a full path of /Resources/HeatFiles/my_heat_file.yaml

and content of “echo hello world” and it is the only file in the folder, than this will be the dictionary returned: {“my_heat_file.yaml”: “echo hello world”}

A new parameter of called files will be added to _generate_workflow method that can be found inside HotPackage class. It will be included in the deploy variable in the same way the template_parameters is included.

13.3.1 Assignee(s)

Primary assignee: michal-gershenzon

Other contributors: noa-koffman

13.3.2 Work Items

- Add support for Heat additional-files. If such files exist in the package, they should be parsed and send with the template, when deploying a Murano environment.

13.4 Dependencies

None

13.5 Testing

Unit tests should cover API calls changes:

- Test that the request for Heat is build correctly with Heat files

13.6 Documentation Impact

None

13.7 References

- http://docs.openstack.org/developer/heat/template_guide/hot_spec.html#get-file

CHAPTER 14

Download bundle of packages to local directory using muranoclient

https://blueprints.launchpad.net/murano/+spec/bundle-save

The purpose is to add command to muranoclient which allows to download the bundle from application catalog to local dir.

14.1 Problem description

There are cases when there is no Internet access cloud with murano installed. Then if user wants to add some bundle of packages into murano, he has to download all of them one-by-one from application catalog using local computer with access to the Internet. After that he saves them somewhere on data storage device and moves all files to cloud.

It is necessary to simplify process of saving packages to avoid manual downloading.

14.2 Proposed change

It's proposed to add new CLI command *bundle-save* to murano-client.

Method *do_bundle_save* will corresponds with new command. It will take three arguments:

- *filename* is a bundle name, bundle url or path to the bundle file;
- *–path* (optional) is a path to directory in which user wants save packages. If it is not specified, current directory will be used;
- *–no-images* (optional) is flag. If it is set, downloading of all required images will be skipped.

Method will build whole list of packages and its dependencies. This ability is already implemented and used in 'bundle-import'. Then method will save bundle file and each package to specified path. For this, method 'save' will be added to *FileWrapperMixin* – the parent class for *Bundle* and *Package* classes. This method will take one argument *dst* – destination for file. It will copy already downloaded file to the specified path.

Method *do_bundle_save* will also save images which packages require. *save_image_local* method will be used for that. All images will be downloaded if *–no-images* is not set.

After bundle saving directory with packages can be moved to lab with murano where all of them can be imported to murano application catalog in one command.

CLI command *package-save* also must be implemented. It will give to user the opportunity to download specific package or several packages he need. The implementation of command will be based on the methods described above.

14.2.1 Alternatives

None

14.2.2 Data model impact

None

14.2.3 REST API impact

None

14.2.4 Versioning impact

None

14.2.5 Other end user impact

User will have access to a new command.

14.2.6 Deployer impact

None

14.2.7 Developer impact

None

14.2.8 Murano-dashboard / Horizon impact

None

14.3 Implementation

14.3.1 Assignee(s)

Dmytro Dovbii

Primary assignee: ddovbii

14.3.2 Work Items

- Add method *save_image_local*
- Add method *save()* to *FileWrapperMixin* class
- Implement CLI command *bundle-save*
- Implement CLI commamd *package-save*

14.4 Dependencies

None

14.5 Testing

Unit tests for CLI client must be updated

14.6 Documentation Impact

CLI reference should be updated manually

14.7 References

None

CHAPTER 15

Rework package class loader logic

https://blueprints.launchpad.net/murano/+spec/change-murano-class-loader

This spec describes rework of murano package class loader as part of the another blueprint simulated-execution-mode-murano-engine.

The detailed logic regarding class modification would be described in this specification.

15.1 Problem description

Class loader should be able to load packages not only from the external repository, but from a local directory. It's needed not only for testing new packages, but for accepting packages on-the-fly. During the development phase the package gets frequently updated, and the need to upload it to the repository on each update complicates the development. Ability to load it from the local filesystem will streamline the process

Another case, when there is no connection to the external repository.

15.2 Proposed change

Need to create one more class, responsible for loading packages. It will look up at the provided directory for a package, that name was requested. It worth noting, that packages should not be zipped.

Class loader from repository (RCL) and new class loader from local dir (LCL) will provide the following logic:

- If local dir is not provided, LCL is not operating. Logic stays same as it's now. RCL do all the work.
- If directory path or several paths to load local packages from are provided, LCL will check all packages in dir and compare with requested name. If the desired class is found, package gets loaded. If not - next dir in the provided list will be scaned.

  If is not found in all the provided directories - RCL sends API call to find it in the repository as usual.

We do have both class loader implemented, we need an ability to combine two (or more) package loaders in one, with the ability to prioritize the queries. For example, a `CombinedPackageLoader` may be created with an instances of `DirectoryPackageLoader` and `ApiPackageLoader` underneath. When a request for a package or class comes in, it is first executed against loader with the highest priority (say, `DirectoryPackageLoader`), and if it is not found there, then goes to the next one.

15.2.1 Alternatives

Use separate class loader in test framework. But support two different class loaders is not good.

Also as an alternative we can also mention this approach. That is: introduce package_loaders config parameter. It can be a list of python-dot-notation class names, that murano would import and try to import pkg from each loader. This could further be modified to a list of lists, to allow params like this: *package_loaders: [pkg_loader1, [pkg_loader2, param1, param2]]* This would allow to easily add custom loaders without changing murano code itself.

15.2.2 Data model impact

None

15.2.3 REST API impact

None

15.2.4 Versioning impact

None

15.2.5 Other end user impact

None

15.2.6 Deployer impact

Enabling loading packages from local directory will be set up in config file.

New key in config file will be added under *[engine]* section and look like that:

Directory used as a way to load packages from. (string value) # local_packages_dir = <None>

15.2.7 Developer impact

None

15.2.8 Murano-dashboard / Horizon impact

None

15.3 Implementation

15.3.1 Assignee(s)

Primary assignee: <efedorova>

Other contributors: <ativelkov>, <slagun>

15.3.2 Work Items

- Implement `CombinedPackageLoader`:
- Perform testing

15.4 Dependencies

- Murano Simulation Mode: https://blueprints.launchpad.net/murano/+spec/simulated-execution-mode-murano-engine

15.5 Testing

Unit tests should be added/updated.

15.6 Documentation Impact

Opportunity to load apps from local directory should be described in the documentation.

15.7 References

None

CHAPTER 16

Implement Cloud Foundry Service Broker API

https://blueprints.launchpad.net/murano/+spec/cloudfoundry-api-support

Cloud Foundry is PaaS which supports full lifecycle from initial development, through all testing stages, to deployment. As far as Cloud Foundry comes in three flavours such as Cloud Foundry OSS, Pivotal Cloud Foundry and Pivotal Web Services.

If we implement Cloud Foundry Service Broker API in murano, murano apps will get an ability to be deployed through CloudFoundry itself. This will improve PaaS user experience as it will have an ability to integrate his app with existing apps in murano and deploy really complex configurations on top of OpenStack.

16.1 Problem description

Typical scenario of Cloud Foundry and murano collaboration will look like:

0. While configuring murano enable Service Broker. It will be deployed at the same nodes but on the different port.
1. Deploy Cloud Foundry on the node using murano or manually.
2. Configure Cloud Foundry to use murano Service Broker.
3. Deploy murano apps through Cloud Foundry API.

NOTE: as far as we want Cloud Foundry to use murano app catalog the end-user will use Cloud Foundry for provisioning.

16.2 Proposed change

We need to write Cloud Foundry Service Broker implementation for murano. One of the parts of this implementation should be some mapping function between Cloud Foundry and OpenStack resources. Now it's planned to map Cloud Foundry Organizations to Openstack tenants and Cloud Foundry spaces to murano environments. So, all tenant users will be granted with the privileges based on their existing roles in OpenStack tenant. And now it looks like we need to have to run a murano environment for every single service provisioned in Cloud Foundry. The parameters which is needed to murano for successful application deployment will store in service object section parameters. The Service Broker itself will parse them as soon as Cloud Foundry can't do it. It will be parsed during Provision request. The request body will look like that: .. code-block:: javascript

{ "service_id": "service-guid-here", "plan_id": "plan-guid-here", "organization_guid": "org-guid-here",
"space_guid": "space-guid-here", "parameters":{

“parameter1”: 1, “parameter2”: “value”

}

}

It’s planned to setup a Service Broker as a separate service which will be deployed on the same nodes as murano. Additional options should be added to the configs. Also we want to use Cloud Foundry experimental asynchronous operations[3].

16.2.1 Alternatives

None

16.2.2 Data model impact

None

16.2.3 REST API impact

No existing API of murano service is going to be changed. New service will be created implementing the standard Cloud Foundry Service Broker API. Its API methods are as follows:

GET /v2/catalog

Request

Method	URI	Description
GET	/v2/catalog	List all available apps

Parameters:

- None

Response

Code	Description
200	OK. The expected resource body as below

```
{"services": [{
    "id": "service-guid-here",
    "name": "mysql",
    "description": "A MySQL-compatible relational database",
    "bindable": true,
    "plans": [{
      "id": "plan1-guid-here",
      "name": "small",
      "description": "A small shared database with 100mb storage quota and 10 connections"
    },{
      "id": "plan2-guid-here",
      "name": "large",
      "description": "A large dedicated database with 10GB storage quota, 512MB of RAM, and 100 conne
      "free": false
    }],
    "dashboard_client": {
      "id": "client-id-1",
      "secret": "secret-1",
      "redirect_uri": "https://dashboard.service.com"
```

```
    }
  }]
}
```

PUT /v2/service_instances/:instance_id?accepts_incomplete=true

Request

Method	URI	Description
PUT	/v2/service_instances/:instance_id?accepts_incomplete=true	Create new service resources for developer

```
{
  "service_id":        "service-guid-here",
  "plan_id":           "plan-guid-here",
  "organization_guid": "org-guid-here",
  "space_guid":        "space-guid-here"
}
```

Response

Code	Description
200	OK. May be returned if the service instance already exists and the requested parameters are identical to the existing service instance.
202	Accepted. Service instance creation is in progress.
409	Conflict. Should be returned if the requested service instance already exists. The expected response body is "{}"
422	Should be returned if the request did not include ?accepts_incomplete=true

```
{
 "dashboard_url": "http://example-dashboard.com/9189kdfsk0vfnku"
}
```

PATCH /v2/service_instances/:instance_id?accepts_incomplete=true

Request

Method	URI	Description
PATCH	/v2/service_instances/:instance_id?accepts_incomplete=true	Update existing service instance

```
{
 "plan_id": "plan_guid_here"
}
```

Response

Code	Description
200	Return if only the new plan matches the old one completely
202	Accepted. Service instance update is in progress.
422	Should be returned if the request did not include ?accepts_incomplete=true

DELETE /v2/service_instances/:instance_id?accepts_incomplete=true

Method	URI	Description
DELETE	/v2/service_instances/:instance_id?accepts_incomplete=true	Delete all resources create during the provision.

Response

Code	Description
202	Accepted. Service instance deletion in progress.
410 | Returned if service does not exist	

422 | Should be returned if the request did not include ?accepts_incomplete=true

---

PUT /v2/service_instances/:instance_id/service_bindings/:binding_id

Request

Method	URI	Description
PUT	/v2/service_instances/:instance_id/service_bindings/:binding_id	Bind service

```
{"plan_id": "plan_guid_here",
 "service_id": "service_guid_here",
 "app_guid": "app_guid_here"
}
```

Response

Code	Description
201	Binding has been created. The expected response body is below.
200	May be returned if the binding already exists and the requested parameters are identical to the existing binding. The expected response body is below.
409	Should be returned if the requested binding already exists. The expected response. body is *{}*, though the description field can be used to return a user-factorin error message.

```
{
  "credentials": {
    "uri": "mysql://mysqluser:pass@mysqlhost:3306/dbname",
    "username": "mysqluser",
    "password": "pass",
    "host": "mysqlhost",
    "port": 3306,
    "database": "dbname"
  }
}
```

DELETE /v2/service_instances/:instance_id/service_bindings/:binding_id

Request

Method	URI	Description
DELETE	/v2/service_instances/:instance_id/service_bindings/:binding_id	Unbind service

Response

Code	Description
200	Binding was deleted
410	Returned if binding does not exist

GET /v2/service_instances/:instance_id/last_operation

Request

Method	URI	Description
GET	/v2/service_instances/:instance_id/last_operation	Polling status of the last 202 operation

Response

Code	Description
200	OK
410	GONE. Appropriate only for asynchronous delete requests Cloud Foundry will consider this response a success and remove the resource from its database.

::

{ “state”: “in progress”, “description”: “Creating service (10% complete).”

}

16.2.4 Versioning impact

None

16.2.5 Other end user impact

None

16.2.6 Deployer impact

Service Broker should be deployed and enabled in the murano config.

16.2.7 Developer impact

None

16.2.8 Murano-dashboard / Horizon impact

None

16.3 Implementation

16.3.1 Assignee(s)

Primary assignee: starodubcevna

16.3.2 Work Items

Changes can be split to this parts:

- Implement the stub of Service Broker itself. Add needed config opts and starting point.
- Implement basic Cloud Foundry API calls such as list and provision. Also on this step we should add murano specific API calls.
- Series of extensions for Cloud Foundry API support: * Add update and deprovision API calls * Add bind/unbind API calls

16.4 Dependencies

None

16.5 Testing

Unit tests should cover new API calls.

16.6 Documentation Impact

Document "Murano and Cloud Foundry HowTo". It should be step by step guide for Cloud Foundry and murano cooperation.

16.7 References

[1] https://youtu.be/ezq9P1WN2LY [2] http://docs.cloudfoundry.org/services/api.html [3] https://docs.cloudfoundry.org/services/asynchronous-operations.html

CHAPTER 17

Remove name field from fields and object model in dynamic UI

https://blueprints.launchpad.net/murano/+spec/dynamic-ui-specify-no-explicit-name-field

Now name is a required parameter in every single form definition. But murano-engine doesn't know anything about this parameter. It mostly used for dymanic UI purposes. So we can insert this field automatically in every form.

17.1 Problem description

- 'name' property has no built-in predefined meaning for MuranoPL classes or applications.

  Now all unknown parameters are ignored, but the error will be spawned and all applications will be invalid in the near future. To prevent global failure this change is suggested.

17.2 Proposed change

Add automatic field inserting into the first form and store 'name' field value in object header:

- Dynamic UI version will be increased to 2.1;
- 'name' property is not required in MuranoPL class definition anymore. If user still have 'name' property in class definition, he should supply corresponding field in UI definition. It will have no special meaning for murano dashboard.
- New field will be inserted to the first form if Dymanic UI version is higher or equal to 2.1;
- If Dymanic UI version is higher or equal to 2.1 'name' field value will be placed to the object header ('?'), in "done" method of application creation wizard;
- In get-mode, application 'name' parameter will be checked in:
  - Primary under '?' parameter;
  - Secondary in the object model root;
- Add new YAQL function to dashboard's YAQL to be used by dynamic UI. The function will be called 'name' and will allow to use automatically-inserted name field in object model description.

17.2.1 Alternatives

None

17.2.2 Data model impact

None

17.2.3 REST API impact

None

17.2.4 Versioning impact

This change introduces new version of dynamic UI form definition. Only backward compatibility will be supported: new dashboard can work with old engine.

17.2.5 Other end user impact

None

17.2.6 Deployer impact

None

17.2.7 Developer impact

None

17.2.8 Murano-dashboard / Horizon impact

This spec is supposed to be implemented in murano-dashboard only. User will see new field for adding app name.

17.3 Implementation

17.3.1 Assignee(s)

Primary assignee: <efedorova@mirantis.com>

17.3.2 Work Items

- Implement automatic 'name' field insertion if version satisfies the requirements;
- Put 'name' field value to the object header area;
- Use 'name' attribute from object header or inside object modal root in environments table;
- Implement new YAQL function, which returns applications name.

17.4 Dependencies

This change is a dependency for a future changes in engine

17.5 Testing

CI tests should be updated and catch all errors.

17.6 Documentation Impact

Dynamic UI form definition examples should be updated

17.7 References

None

CHAPTER 18

Configure environments from CLI

Blueprint for this specification:

https://blueprints.launchpad.net/python-muranoclient/+spec/env-configuration-from-cli

Currently there is no way to configure and/or deploy a murano environment from the command line. This specification describes how this issue can be addressed and what steps have to be taken in order to implement this abilities.

18.1 Problem description

Currently the only possible way to deploy a murano environment is through interaction with horizon. CLI client currently lacks commands to add apps to environment or deploy an environment. This makes murano useless without horizon and murano-dashboard. It also means, that scripting and/or automating deployment of murano apps/environments is currently problematic to say the least.

18.2 Proposed change

This spec proposes to add several new CLI commands and modify existing to allow user to add apps to the env, configure it and send it to deploy.

Command `environment-create` currently only supports adding the name of the environment, while the api call supports setting aditional params, such as defaultNetworks. `environment-create` should be updated to allow setting custom keys to environment model.

`environment-edit` command should perform all the editing (adding, deleting, modifying of existing apps) This command should be non-interctive, adding an interactive mode is a good idea as of itself, but is out of scope of this spec.

`environment-edit` would use jsonpatch format for input (as described by RFC 6902). This is a well known and powerfull instrument to manipulate json, and object model is currently stored in json. This would allow easy addition, modification and deletion of apps. Editing of environment attributes is not allowed by API, so it's beyond the scope of this spec.

During command execution, env configuration parameters should be read from stdin or a file. Supporting stdin as input source means it should be possible to issue a command like `murano environment-edit env_id < app_patch.json`

An example `app_patch.json` might look like:

```
[
  { "op": "add", "path": "/services/-",
      "value":
        {'?':
          {'id': '2fc3005d-b4f8-420d-9e15-f961b41f49ee',
           'type': 'io.murano.apps.App1'
          },
         'instance':
              {'?': {'id': '53366263-04e9-49d1-829d-50e27158749c',
                     'type': 'io.murano.resources.LinuxMuranoInstance'},
               'assignFloatingIp': True,
               'availabilityZone': 'nova',
               'flavor': 'm1.medium',
               'image': 'afc1aa61-f623-4c66-bbd8-5359261c5272',
               'name': 'ilphvibwqyu6h1'  },
         'name': 'App1'
         }
  },
]
```

All operations would only apply to services field, so the */services/* part of the path can be ommited.

note:: This means, that a user has to generate unique ids. This can be a difficult task and may hinder scripting capabilities. We might think of a way to mitigate this issue, by introducing a way to generate ids. This can be done by introducing an optional parameter `--id-format`. It would accept a template for strings to be replaced with generated ids. For example: `--id-format=##id##` would replace all occurrences of ##id##-1 with fist generated id, all occurrences of ##id##-2 with second generated id and all occurrences of ##id## with a unique id. uuids should be used for id generation.

Command should perform validation of its input. This can be done using MuranoPL classes and property contracts. If an error happens during non-interactive mode (for example validation fails for some field) `murano` should return a non-zero status, to facilitate scipting and error-detecting.

Command `environment-edit` should support optional `--session` parameter, that allows user to specify a session for editing an environment. It should also warn user in case more than one session is open. If the session parameter is omitted: last open session for given environment is used. If there are no open sessions — a new session would be opened.

note:: This means that to deploy a simple application one has to know jsonpatch syntax. We might think about optional *syntax sugar* commands, that would allow adding a single app to the environment. These would use predefined jsonpatch patch documents and would accept only value to be added to /Objects/services/- path.

We should implement session controlling commands during implementation of this spec, for example:

1. `session-list` Show list of open sessions for a given environment.
2. `session-show` Show details of currently configured session..
3. `session-delete` Discard an open session for a given environment.

In case an error occurs during environment deployment this would allow to rollback the changes to the previous version of environment. This should also allow to handle cases, when a single environment is edited simultaneously by different users.

`environment-action` command should be implemented to allow performing actions against an environment. One example of actions is a deploy action, so `murano environment-action deploy` should deploy the environment.

18.2.1 Alternatives

Alternatively we can implement an interactive mode, that would mimic current dashboard behaviour. This would not allow scripting, but would allow us to build a more user-friendly CLI interface. The dowside is that current UI definitions are scheduled to be changed and would most likely be replaced in the forseable future. This means, that most of the work would go to waste.

18.2.2 Data model impact

None

18.2.3 REST API impact

None, CLI should reuse API calls, already used by dashboard

18.2.4 Versioning impact

Since we're adding functionality — None

18.2.5 Other end user impact

None

18.2.6 Deployer impact

It is possible that implementation of this spec would require setting and reading intermediate environment variables to work correctly with sessions, during app addition, env deployment.

Overall deployment of murano would not be affected.

18.2.7 Developer impact

None

18.2.8 Murano-dashboard / Horizon impact

None

18.3 Implementation

18.3.1 Assignee(s)

Primary assignee: <kzaitsev>

18.3.2 Work Items

1. `environment-edit` command (should create session, check permissions, add patch resulting model into).
2. Input validation for `envieonment-edit` (request packages from API, check ID uniqueness, check input field adequacy).
3. (optional) `--format-id` option support
4. (optional) syntax sugar command, that would allow easy addition of a package to an environment.
5. Session controlling commands.
6. `environment-action` command.
7. Shell unit tests.
8. Integration tests.

18.4 Dependencies

None

18.5 Testing

We shall need Unit tests for new commands introduced.

Also, since this change introduces a way to deploy an env from CLI. This means that integration tests for murano client should be implemented. A typical test should upload and app, configure a simple environment with 1-2 apps and set some custom parameters, like access port and optionally deploy the env in question. This tests should probably take place in murano-ci.

18.6 Documentation Impact

New python-muranoclient commands would have to get a proper documentation. It's also possible, that we would want to document the whole process of deploying an app or scripting of such a deployment as a separate article in murano documentattion.

18.7 References

- http://jsonpatch.com
- https://tools.ietf.org/html/rfc6902
- https://tools.ietf.org/html/rfc7159
- https://pypi.python.org/pypi/jsonpatch

CHAPTER 19

Environment abandoning support

https://blueprints.launchpad.net/murano/+spec/environment-abandon

The purpose is to add an ability to abandon the environment, which hangs during deployment or deleting in order to hide it from the list of other environments.

19.1 Problem description

There are cases when the process of deleting or deploying environment may hang. Then the user has to release all used resources and manually clean the database in order to get rid of the list of failed environments.

Manual cleaning of the database is unsafe and inconvenient operation. That's why *abandon* feature has to be provided. It should improve usability and allows to user to avoid direct database editing. But it is necessary to notify the user that all resources used by abandoned environment must be also released manually as previously.

19.2 Proposed change

The implementation of this feature consists of three stages:

1. Modify environment-delete API endopoint to have *abandon* feature

The method *delete* must be changed. Depending on the parameter "abandon", obtained from request data, the method should work differently. This parameter has a boolean type. If it equals *True* enviroment will be directly removed from the database without object model cleanup.

2. Provide corresponding changes in the python-muranoclient

Method *delete* of environment manager should be modified. New boolean parameter *abandon* with default value *False* should be added. The value of parameter affects the building of url, which is sent to murano-api.

3. Add new button *Abandon* to murano-dashboard

This button should be available with any environment state.

Proposed change doesn't solve problem of deployment process hanging. Murano-engine may continue to deploy abandoned environment. It is necessary to find a way how to stop murano-engine in this case.

19.2.1 Alternatives

None

19.2.2 Data model impact

None

19.2.3 REST API impact

DELETE /environments/<environment_id>?abandon

Request

Method	URI	Description
DELETE	/environments/{id}?abandon	Remove specified environment.

Parameters:

- *abandon* - boolean, indicates how to delete environment. *False* is used if all resources used by environment must be destroyed; *True* is used when just database must be cleaned

Response

Code	Description
200	OK. Environment deleted successfully
403	User is not allowed to delete this resource
404	Not found. Specified environment doesn‘t exist

19.2.4 Versioning impact

Murano dashboard will support only the version of the client, that includes corresponding changes.

19.2.5 Other end user impact

None

19.2.6 Deployer impact

None

19.2.7 Developer impact

None

19.2.8 Murano-dashboard / Horizon impact

New action *Abandon* will be added to murano-dashboard. It will be always available in the row of other action.

Dialog with warning should appear when user executes action

19.3 Implementation

19.3.1 Assignee(s)

Dmytro Dovbii

Primary assignee: <ddovbii@mirantis.com>

19.3.2 Work Items

- Modify method 'delete' of environment API to support two delete modes
- Implement adding 'abandon' parameter to url in 'delete' method of environment manager in muranoclient
- Add flag '–abandon' to CLI command 'environment-delete'
- Add new class 'AbandonEnvironment' which provide new button 'Abandon' in murano-dashboard

Implementation is acutally completed.

19.4 Dependencies

None

19.5 Testing

Functional tests for murano-dashboard must be updated. Unit tests should cover API call and CLI client Tempest tests are out of the scope of this spec.

19.6 Documentation Impact

API specification should be updated

19.7 References

None

CHAPTER 20

Logging API for MuranoPL

https://blueprints.launchpad.net/murano/+spec/logging-api-for-muranopl

The purpose is to add an ability to log actions while developing MuranoPL applications

20.1 Problem description

It is good practice to log basic stages during application execution. Logging API should make debugging and troubleshooting processes easier.

20.2 Proposed change

Implemention key points

1. New MuranoPL class *io.murano.system.Logger*

   Class *Logger* will be part of the MuranoPL core library.

   This class should contain basic logging functionality. Usage example in MuranoPL:

   ```
   $.log: logger('logger_name')
   $.log.info('message: {0}', 'checkpoint')
   ```

   that code should be equivalent of python code:

   ```
   from oslo_log import log as logging
   LOG = logging.getLogger('logger_name')
   LOG.info(_LI('message: {0}').format('checkpoint'))
   ```

   Others logging methods are *Logger.debug*, *Logger.info*, *Logger.warning*, *Logger.error*, *Logger.critical*. Each method corresponds to the oslo.log logging level.

   There is also separate method for exception stack trace output described below.

2. Exceptions logging

   Method *Logger.exception* intended to log a stack trace:

   ```
   $.log.exception(exc, 'Something bad happened: {0}', 'Oops!')
   ```

   exception method uses the same log level as *Logger.error*.

3. New global MuranoPL function *logger()*

Call of the function *logger('logger_name')* returns new instance of the *io.murano.system.Logger*. If function with the same *logger_name* was called previously then the same logger instance should be returned instead of building new one.

4. Logging configuration

   Logging should use standard Murano Engine logging subsystem.

   Application itself cannot set logging settings at runtime.

   All of appenders, formatters an etc. may be configured via standard way as others loggers in Murano Engine.

   Prefix *applications* should be added to each logger name which created by application. As example, logger named *active_directory* in the MuranoPL should be identified as *applications.active_directory* at the python side and in the system config. This is for separation loggers used by applications and engine system loggers. Also it will allow us to specify settings for the application loggers separately from others loggers.

5. Default configuration

   All logs written in the one file by default. Log rotation should be used, so maximum size of logs is limited.

6. Logging naming conventions

   A note about the logging naming conventions should be added to the MuranoPL tutorial.

20.2.1 Alternatives

None

20.2.2 Data model impact

None

20.2.3 REST API impact

None

20.2.4 Versioning impact

None

20.2.5 Other end user impact

None

20.2.6 Deployer impact

Deployer will get more information about application execution stages and additional tool for more precise troubleshooting

20.2.7 Developer impact

Logging API will allow developer to debug application in a more effective manner getting information from logs.

20.2.8 Murano-dashboard / Horizon impact

None

20.3 Implementation

20.3.1 Assignee(s)

Alexey Khivin

Primary assignee: <akhivin@mirantis.com>

20.3.2 Work Items

- Create new MuranoPL class *io.murano.system.Logger*
- Create new global MuranoPL function *logger()*
- Create method *Logger.exception*
- Add new section for logging parameters into the Murano Engine config
- Describe naming conventions for loggers in the Murano docs

20.4 Dependencies

None

20.5 Testing

Functional tests for MuranoPL must be updated.

20.6 Documentation Impact

MuranoPL

20.7 References

None

CHAPTER 21

Murano API - All Tenants Search

https://blueprints.launchpad.net/murano/+spec/murano-api-all-tenants-search

Congress Murano datasource driver pulls environments from one tenant only. The goal is to pull all environments from all tenants (as nova driver does for servers).

21.1 Problem description

Murano - Congress integration is part of a part of Policy Guided Fulfillment. It uses Congress policy framework to define and evaluate restrictions on Murano environments. So Murano environments are pulled by Congress Murano datasource driver, so Congress policy rules can be evaluated.

The problem is that Murano REST API returns environments of one tenant of authenticated user's token only. Thus Congress policy rules evaluation is run on data from one tenant only.

Other Congress datasource drivers are dealing with similar requirements also - for example Nova datasource driver pulls data about all servers across all tenants in its *nova* policy. It is possible because Nova REST API supports search option *all_tenants*.

Note that *Congress policy* is a place of both rules and data related to one *service*. If *policy* is defined by datasource driver, then its configuration have *user*, *password* and *tenant*, which are used to get token to access the service.

21.2 Proposed change

Search option *all_tenants* will be added to operation *List Environments* of Murano REST API. When set, the returned list will contain all environments accessible by the user (specified by token) regardless of tenant. Listing environments from all tenants can only admin user.

21.2.1 Alternatives

The requested behavior can be also achieved by iterating operation *List Environment* over all tenants available to the configuration user. This solution has following performance issues:

- each pull cycle executes the REST operation for every tenant where user is member, instead of one execution in case of *all_tenants*
- user's tenant assignment has to be periodically updated, so it leads to another requests to keystone each such period

21.2.2 Data model impact

None

21.2.3 REST API impact

- List Environments
  - *all_tenants* parameter is added. When set to *true*, then search over all tenants is executed, otherwise search on token's tenant is done

Example (without *all_tenants*):

```
GET http://<server-name>:8082/v1/environments

{
  "environments": [
    {
      "status": "deploying",
      "updated": "2015-05-06T08:14:06",
      "networking": {},
      "name": "test",
      "created": "2015-05-06T08:08:40",
      "tenant_id": "cd9e218f9b894ebdb421e9906fbec15e",
      "version": 1,
      "id": "8cc3187c763f4ca9bc58cdaf89f926d3"
    }
  ]
}
```

Example (with *all_tenants* - note different *tenant_id*):

```
GET http://<server-name>:8082/v1/environments?all_tenants=true

{
  "environments": [
    {
      "status": "deploying",
      "updated": "2015-05-06T08:14:06",
      "networking": {},
      "name": "test",
      "created": "2015-05-06T08:08:40",
      "tenant_id": "cd9e218f9b894ebdb421e9906fbec15e",
      "version": 1,
      "id": "8cc3187c763f4ca9bc58cdaf89f926d3"
    },
    {
      "status": "deploying",
      "updated": "2015-05-08T09:34:16",
      "networking": {},
      "name": "test 2",
      "created": "2015-05-08T08:18:20",
      "tenant_id": "8908989abbeec239023489023ccc1234f",
      "version": 1,
      "id": "abecbf88328932bbecbefe82348238b"
    }
  ]
}
```

21.2.4 Versioning impact

None

21.2.5 Other end user impact

python-muranoclient will be changed as follows:

- *–all-tenants* on CLI

Example:

```
$ murano environment-list --all-tenants
```

- *search options* will be supported on API level

Example:

```
class EnvironmentManager(base.ManagerWithFind):
   def list(self):
   ...

   def list(self, search_opts):
   ...
```

21.2.6 Deployer impact

None

21.2.7 Developer impact

None

21.2.8 Murano-dashboard / Horizon impact

None

21.3 Implementation

21.3.1 Assignee(s)

Primary assignee: filip-blaha

21.3.2 Work Items

- Introduce *all_tenants* search option in
  - file *murano/api/v1/environments.py*
- Modify *policy.json* file with rules
  - file *etc/murano/policy.json*

- Add support for search options in *python-muranoclient*
  - file *muranoclient/v1/environments.py*
- Add support for *–all-tenants* in *python-muranoclient* CLI
  - file *muranoclient/shell.py*

21.4 Dependencies

None

21.5 Testing

Unit tests should cover server API side also client and shell should be covered.

21.6 Documentation Impact

REST API documentation will be modified to mention *all_tenants* search option.

21.7 References

- https://wiki.openstack.org/wiki/PolicyGuidedFulfillmentLibertyPlanning
- https://wiki.openstack.org/wiki/PolicyGuidedFulfillmentLibertyPlanning_MuranoAPI

CHAPTER 22

Murano API - Core Model Component Integration Improvement

https://blueprints.launchpad.net/murano/+spec/murano-core-model-integration-improvements

Core Model can be seen as API, because user is using it when writing Datalog queries in Congress, or integrating with Mistral workflows. Current Murano Core model does not provide means to easy link them with realized OpenStack entities (for example Murano Instance does not provide UUID of provisioned Nova Server).

22.1 Problem description

Congress datalog queries are one of core features used by Policy Guided Fulfillment. These queries are used to express validity of Murano environment either in pre-deployment and/or runtime. In order to evaluate environment validity it is necessary to work with realized entities by core Murano components - for example

- *I want to check if Murano Instance's Nova server exists and is running*
  - In kilo I have to use IP address of the Murano Instance and do multiple joins of neutron tables to identify Nova server.
- *I want to check if owner of network used by Murano Environment is from given group of users*
  - In kilo it is impossible because Murano network object contains only name of the network (*a-network*), while realized network (via Heat) contains name with Murano object id (*a-network-bed7a70ed791434c8acdd53a52a8d4ca*)

22.2 Proposed change

Changes of core Murano model:

- **io.murano.resources.Instance**
  - add property *openstackId* and fill the property with value once Heat provisioned Instance

```
openstackId:
  Contract: $.string()
  Usage: Out
```

- **io.murano.resources.Network**
  - add property *openstackId* and fill the property with value once Heat provisioned network (as part of Instance provisioning)

```
openstackId:
  Contract: $.string()
  Usage: Out
```

22.2.1 Alternatives

Instead of adding the same property to each class aware of openstack ID we can create mixin class (e.g. **OpenstackIdMixin**) with this property and all classes aware of openstack ID will extend that mixin.

```
Name: OpenstackIdMixin
Properties:
  openstackId:
    Contract: $.string()
    Usage: Out
```

22.2.2 Data model impact

None

22.2.3 REST API impact

None

22.2.4 Versioning impact

Murano package versioning is currently analyzed and it is planned for Liberty.

It makes sense to introduce it (modifications of the core Murano packages) as new versions of core Murano packages.

On the other hand, proposed changes are backward compatible, so they can be done prior versioning.

22.2.5 Other end user impact

None

22.2.6 Deployer impact

If proposed changes will be done prior Murano package versioning, then after upgrade the Murano objects won't have initialized introduced properties (*openstackId*).

22.2.7 Developer impact

None

22.2.8 Murano-dashboard / Horizon impact

The change will simplify implementation of Horizon UI (instance detail)

22.3 Implementation

22.3.1 Assignee(s)

Primary assignee: filip-blaha

22.3.2 Work Items

- introduce *openstackId* properties to
  - *meta/io.murano/Classes/resources/Instance.yaml*
  - *meta/io.murano/Classes/resources/Network.yaml*
- implemented instance and network *openstackId* property population
  - *meta/io.murano/Classes/resources/Instance.yaml* , *deploy* method
  - *meta/io.murano/Classes/resources/NeutronNetwork.yaml* , *deploy* method
  - *meta/io.murano/Classes/resources/NovaNetwork.yaml* won't be modified, as nova networking is deprecated

22.4 Dependencies

https://blueprints.launchpad.net/murano/+spec/murano-versioning

22.5 Testing

Both unit and tempest tests of policy guided fulfillment will be enhanced to test properties *openstackId*.

22.6 Documentation Impact

None

22.7 References

- https://wiki.openstack.org/wiki/PolicyGuidedFulfillmentLibertyPlanning
- https://wiki.openstack.org/wiki/PolicyGuidedFulfillmentLibertyPlanning_MuranoAPI

CHAPTER 23

Murano unified logging

https://blueprints.launchpad.net/murano/+spec/unified-style-logging

Rewrite murano logging in unified OpenStack style proposed by https://blueprints.launchpad.net/nova/+spec/log-guidelines

23.1 Problem description

Now log levels and messages in murano are mixed and don't match the OpenStack logging guideliness.

23.2 Proposed change

The good way to unify our log system would be to follow the major guidelines. Here is a brief description of log levels:

- Debug: Shows everything and is likely not suitable for normal production operation due to the sheer size of logs generated (e.g. scripts executions, process execution, etc.).
- Info: Usually indicates successful service start/stop, versions and such non-error related data. This should include largely positive units of work that are accomplished (e.g. service setup, environment create, successful application deployment).
- Warning: Indicates that there might be a systemic issue; potential predictive failure notice (e.g. package execution problems, problems with categories listing).
- Error: An error has occurred and an administrator should research the event (e.g. deployment failed, app add failed).
- Critical: An error has occurred and the system might be unstable, anything that eliminates part of murano's intended functionality; immediately get administrator assistance (e.g. failed to access keystone/database, plugin load failed).

As far as murano-dashboard has it own notification system all notifications should be duplicated at log messages and should follow this spec in the selection of log level.

Here are examples of log levels depending on environment execution:

- Action execution:

```
LOG.debug('Action:Execute <ActionId: {0}>'.format(action_id))
```

- Environment creation:

```
LOG.info(_LI('Environments:Create {id} succeed>'.format(id=environment.id)))
```

- Package execution problems

```
LOG.warning(_LW("Class is defined in multiple packages!"))
```

- Environment is not found

```
LOG.error(_LE('Environment {id} is not found').format(id=environment_id))
```

Additional step for our logging system should be usage of pep3101 as unified format for all our logging messages. As soon as we try to make our code more readable please use {<smthg>} instead of {0} in log messages.

23.2.1 Alternatives

We need to follow OpenStack guidelines, but if needed we can move plugin logs to DEBUG level instead of INFO. It should be discussed separately in each case.

23.2.2 Data model impact

None

23.2.3 REST API impact

None

23.2.4 Other end user impact

None

23.2.5 Deployer impact

None

23.2.6 Developer impact

None

23.2.7 murano-image-elements impact

None

23.2.8 murano-dashboard / Horizon impact

None

23.3 Implementation

23.3.1 Assignee(s)

Primary assignee: starodubcevna

23.3.2 Work Items

- Unify existing logging system
- Unify logging messages
- Synchronize dasboard notifications and log entries
- Add additional logs if needed

23.4 Dependencies

None

23.5 Testing

None

23.6 Documentation Impact

None

23.7 References

https://blueprints.launchpad.net/nova/+spec/log-guidelines https://www.python.org/dev/peps/pep-3101/

CHAPTER 24

Policy Based Modification of Environment

https://blueprints.launchpad.net/murano/+spec/policy-based-env-modification

Goal is to be able to define modification of an environment by Congress policies prior deployment. This allows to add components (for example monitoring), change/set properties (for example to enforce given zone, flavors, ...) and relationships into environment, so modified environment is after that deployed.

24.1 Problem description

Currently it is possible to reject deployment of an environment if it does not follows set of so called pre-deployment policies set by admin. Administrator wants to also modify environment prior it is deployed:

- add/set/remove component properties
- add/remove relationships
- add/remove objects

Example Use Case: Policy Based Monitoring

Admin wants to monitor an environment, so he wants to

- install monitoring agent on each Instance
- it is done by adding component with the agent and creating relationship between agent and Instance. It is done at pre-deploy time
- register monitoring agent on Monitoring server
- it is done by calling monitoring server API during deployment of monitoring agent.

24.2 Proposed change

Changes

- Introduce new Congress policy rule *predeploy_modify(eid,oid,modify-action-id,priority, [key-val]*)*

  predeploy_modify policy rule is queried on all actions. Simulation Congress API is used like in case of *predeploy_errors* policy rule.

  If it returns non empty list of *modifications* for given environment, then

  - *deploy* action is temporarily paused, until all modifications are processed

    - if any of modification fails, then environment *deploy* fails
- Pluggable modification actions Modification actions can be plug using setup *entry_points*.

  Out of box, there will be following modification actions
- add_property(name=name, value=value)
- remove_property(name=name)
- set_property(name=name, value=value)
- add_relationship(name=name, source=source-uuid, target=target-uuid)
- remove_relationship(name=name, object=object-uuid)
- add_object(type=type, owner=owner-uuid, owner-rel-name=name, [name=val]*)
- remove_object(object=object-uuid)

24.2.1 Alternatives

Alternative can be usage of *executes[]* of Congress policy, which executes modify actions. In this approach

- modify action has to be implemented as Congress datasource action
- triggering of executes[] has to be solved
- it is not possible to order modify action ordering
- Murano session-id of REST API must be passed to Congress
- actions can be executed only as asynchronous, so it is not possible to postpone *deploy* environment action until all modify actions are finished

Thus it is not alternative.

24.2.2 Data model impact

None

24.2.3 REST API impact

None

24.2.4 Versioning impact

None

24.2.5 Other end user impact

User (admin) can control modification by creating *predeploy_modify* Congress policy rules.

24.2.6 Deployer impact

None

24.2.7 Developer impact

None

24.2.8 Murano-dashboard / Horizon impact

None

24.3 Implementation

24.3.1 Assignee(s)

Primary assignee: filip-blaha

24.3.2 Work Items

- design API of modify actions
- framework for pluggable modify actions - registering and managing available actions
- implement out-of-box actions
- add point to engine where congress called and returned action list is processed on given environment

24.4 Dependencies

None

24.5 Testing

We need to cover by unit tests: * framework for registering/managing modify actions * applying modify actions on environment * processing action list returned by congress

We need to create functional tests covering end-to-end scenario.

24.6 Documentation Impact

It is documented as part of policy guided fulfillment.

24.7 References

CHAPTER 25

Simulated Execution Mode For Murano Engine

https://blueprints.launchpad.net/murano/+spec/simulated-execution-mode-murano-engine

25.1 Problem description

As an Application Developer I'd like to execute my workflows without actual deployment and interaction with murano-dashboard in order to verify my workflow before actual deployment by those increasing my application development speed.

25.2 Proposed change

Verifying application packages should be simple and fast. User doesn't have to re-upload package and add app to the environment on every change.

- Allow application author to validate his application using unit-tests
- Those tests will be put to the application package to allow anyone to test this app at any time
- Tests will look like regular unit tests. Testing framework, witch will run unit-tests will support commands, that will allow to load test package from directory, to call class methods and configure deployments parameters. That will make deployment test run easier. Also, test writer may run deployment several times, examine and compare results with different parameters
- Tests should be able to produce complete object model with some parameters of deployment:
  - environment attributes (such as tokens) (or overwriting values, defined in config);
  - mock the methods of system classes which include various kinds of external communications. Dependent applications, system resources and various API clients and also be mocked. It should be allowed to specify a returned value. There would be separate specification for mocking, where the details will be described.

Test-case prototype may look like that:

```
Namespaces:
   =: io.murano.apps.foo.tests
   sys: io.murano.system
   pckg: io.murano.apps.foo

Extends: io.murano.tests.TestFixture

Name: FooTest
```

```
Methods:
   initialize:
       Body:
           # - $.appJson: new(sys:Resources).json('foo-test-object-model.json')
           - $.appJson:
               - ?:
                   id: 123
                   type: io.murano.apps.foo.FooApp
                 name: my-foo-obj
                 instance:
                     ?:
                         type: io.murano.resources.Instance
                         id: 42
                     ...

   setUp:
       Body:
           - $.env: $.createEnvironment($.appJson)   # creates an instance of std:Environment
           - $.myApp: $.env.applications.where($.name='my-foo-obj').first()
           - mock($.myApp.instance, "deploy", "mockInstanceDeploy", $this)
           - mock(res:Instance, deploy, "mockInstanceDeploy", $this)


   testFooApp:
       Body:
           - $.env.deploy()
           - $.assertEqual(true, $.myApp.getAttr('deployed'))

   tearDown:
       Body:

   mockInstanceDeploy:
       Arguments:
           - mockContext
       Body:
           - Return:
              # heat template
```

25.2.1 Alternatives

Provide one CLI command, that will mock creation of VMs and other things and returns the deployment result.

Cons: Impossible to verify deployments, where execution plan returns a value, which is used in future app workflow. Compare results of several deployments would be inconvenient Real VM's can't be

25.2.2 Data model impact

None

25.2.3 REST API impact

None

25.2.4 Versioning impact

Tests will be placed to a package, so manifest version need to be updated. This functionality should be described in a separate spec. For now, there will be no impact on project itself.

25.2.5 Other end user impact

None

25.2.6 Deployer impact

None

25.2.7 Developer impact

None

25.2.8 Murano-dashboard / Horizon impact

None

25.3 Implementation

25.3.1 Assignee(s)

Primary assignee: <efedorova@mirantis.com>

25.3.2 Work Items

#. Add 'simulation' mode (new entry-point) to Murano Engine, where packages would be uploaded from the provided path there would be no interconnection with RabbitMQ

1. Make changes to the class-loader, located in engine, to not use API. Separate spec is provided for this change (https://review.openstack.org/#/c/198745/).
2. Implement testing framework, written in MuranoPL that will include the classes, described below. The structure would be taken from python unittest module. Framework will include test-runner
3. Implement mock support.
4. Define what is needed to change in MuranoPL itself

Testing framework may contain the following classes and methods. This are base classes for simple testing framework.

- `TestCase` class

Method	Description
setUp()	Method called immediately before calling the test method.
tearDown()	Method called immediately after the test method has been called and the result recorded.
run(result=None)	Run the test, collecting the result into the test result object passed as result.
assert...	Different asserts (assertEqual, assertNotEqual, assertTrue, assertFalse).

- `TestResult` class: This class is used to compile information about which tests have succeeded and which have failed.

Attribute	Description
errors	A list containing 2-tuples of TestCase instances and strings holding formatted tracebacks. Each tuple represents a test which raised an unexpected exception.
failures	A list containing 2-tuples of TestCase instances and strings holding formatted tracebacks. Each tuple represents a test where a failure was explicitly signalled using the TestCase.assert*() methods.
testsRun	The total number of tests run so far.

- `TestRunner(stream=sys.stderr, descriptions=True, verbosity=1)` A basic test runner implementation which prints results on standard error. Has *run* method, witch executes the given test case. Also stores the execution result.

For the fist time test may be run only one by one. Later we can add `TestSuite` class and `TestLoader` class: * `TestLoader` class is responsible for loading tests according to various criteria and returning them wrapped in a TestSuite (or TestSuite if will add this class).

Methods	Description
loadTestsFromTestCase(testCaseClass)	Return a suite of all tests cases contained in the TestCase-derived testCaseClass.

1. Implement simple mocking machinery

All mockes are separated into NonCallable and Callable mocks

`Mock` class

Public methods

Methods	Description
start()	Activate a patch, returning any created mock.
stop()	Stop an active patch.
patch(target)	The *target* is patched with a *new* object. *target* should be a string in the form *package.module.ClassName*.
attach_mock(mock, attribute)	Attach a mock as an attribute of this one, replacing its name and parent
configure_mock(kwargs)	Set attributes on the mock through keyword arguments

Private methods:

initialize, __call__, _patch, __enter__, __exit__

25.4 Dependencies

None

25.5 Testing

None

25.6 Documentation Impact

New testing framework will be documented from scratch.

25.7 References

Discussions in IRC will be provided

CHAPTER 26

Add network selection element to UI form

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/murano/+spec/ui-network-selection

Sometime a VM should be placed to existing network rather then to a new network created during deployment. While our workflows support this, there is no way for the end user to select this network in the easy way in the UI. It will be great if there is a special form element which will pre-populate a list of available networks and provide an easy option to select desired network for the application.

26.1 Problem description

Currently murano supports only so called "Default Scenario" when it comes to networking: it creates a single network per environment, picks a router, allocates an IP range and creates a subnet with this range within a created network. This behavior is fine for most of the cases, however it may be insufficient in complicated scenarios and topologies.

For example, an application may require some pre-configured networks to exist so it may manipulate with the resources associated to them, allocate floating IPs from the specific net etc. In this case, the existing net becomes a valid input property for the application, so its developer may ask the users to specify it before the deployment.

Another scenario is the need to have all the VMs of the environment to join some pre-configured network - regardless of their configuration and the applications they run for. This may be caused by some specific networking requirements of a particular cloud (a frequent example is a custom proxy to access the internet which is reachable only from a specific network segment).

When the Network Management scenarios were initially introduced in murano [1] we planed to have a so-called "Advanced scenario", i.e to provide the users with an ability to use existing networks, subnets, routers etc, or configure some sophisticated combination of them.

This scenario was properly supported at engine and at the level of the Core Library: the *io.murano.resources.Instance* class has a *networks* field which allows to specify *customNetworks* as a collection of objects inheriting from *io.murano.resources.Network*, which may include the existing networks or new networks with non-default configuration.

However, there is no support of this functionality at the UI level: the object model being generated by the Dynamic UI contains the default networking definitions only: i.e. the directive to join Environment's default network, which is - in its turn - is hardcoded to be a newly created network and a subnet in it.

26.2 Proposed change

A new field will be added to the Dynamic UI framework which will allow to pick the network and a subnet from the ones available to the current user. This will be a drop-down list populated when the form is rendered. The value selected by the user in this field will be a tuple, consisting of the network id and a subnet id. This ids may be passed to the application either as plain strings or as part of a more complicated Object Model, for example as the properties of *io.murano.resources.ExistingNeutronNetwork* objects. It is up to application developer to properly interpret and use these values. The existing applications will not be affected by this change, as their "configure instance" step of the UI dialog will not include any networking settings. In future some of our standard apps may be updated to utilize this new field, but those updates are out of the scope of this spec.

A similar field (but a static one rather then defined as part of Dynamic UI framework) should be added to the *New Environment* dialog form, so the user may choose an existing network to be the default network of a given environment. The default value in that choice should lead to the creation of a new environment's network (i.e. to replicate the existing behavior), while any other choice should lead to a generation of a new object of type *io.murano.resources.ExistingNeutronNetwork* which will be passed to murano-api as part of *defaultNetworks* dictionary as the default environment's network.

Both these fields should share the same logic for value population. Additionally, the dynamic UI field should have the following options, defined as constructor arguments and exposed to dynamic ui as yaml attributes:

- *include_subnets* - *True* by default. If *True*, the dropdown includes all the possible combinations of network and subnet. E.g. if there are two available networks X and Y, and X has two subnets A and B, while Y has a single subnet C, then the dropdown will include 3 item: (X, A), (X, B), (Y, C). If set to *False* the subnet info will not be retrieved, and *None* values will be returned as second items in output tuples, so only network ids are available.
- *filter* - *None* by default. If set to a regexp string, will be used to display only the networks with names matching the given regexp.
- *murano_networks* - *None* by default. May have values *None*, *exclude* or *translate*. Defines the handling of networks which are created by murano. Such networks usually have very long randomly generated names, and thus look ugly when displayed in dropdown. If this value is set to *exclude* then these networks are not shown in the dropdown at all. If set to *translate* the names of such networks are replaced by a string *Network of %env_name%*. Note that this functionality is based on the simple string matching of the network name prefix and the names of all the accessible murano environments. If the environment is renamed after the initial deployment this feature will not be able to properly translate or exclude its network name.
- *allow_auto* - *True* by default. Defines if the default value of the dropdown (labeled "Auto") should be present in the list. The default value is a tuple consisting of two *None* values. The logic on how to treat this value is up to application developer. It is suggested to use this field to indicate that the instance should join default environment network. For use-cases where such behavior is not desired, this parameter should be set to *False*.

The string representation of the dropdown items should look like *%NetworkName%: %cidr% (%SubnetName%)*, where *%SubnetName%* part is optional and may be not present if the subnet's name is not set.

If neutron is not available (so murano falls back to nova-network support) the dropdown (both the static and dynamic ones) are not populated and appropriate hints are available in the *New Environment* dialog.

26.2.1 Alternatives

Currently the only way to change the default networking behavior is the usage of *networking.yaml* file which allows to override the networking setting at the environment level, for all the murano environments of all the tenants. This is not flexible enough and does not provide the desired user experience.

However this method will remain, as it allows to override the network setting globally.

26.2.2 Data model impact

No impact. The existing data structures will be used.

26.2.3 REST API impact

No impact. The existing API methods will be used.

26.2.4 Versioning impact

As this feature adds a new type of Dynamic UI fields this will bump the minor version of Dynamic UI format version. The version will change from 2.0 to 2.1

26.2.5 Other end user impact

The user will see the new field in the "Create Environment" dialog. It will also be shown when the environment is created inline in the environments grid. The default value of this new field will follow the old behavior.

The user experience with the existing applications will not be changed.

26.2.6 Deployer impact

The dropdown in Dashboard will be calling public neutron APIs. If they are not accessible due to some reason, the UI will guess that neutron is not installed at all so nova network usage will be assumed. However, the actual decision on the fallback to nova-network is done at the murano-api. So, if the dashboard is unable to connect to neutron while the api is then the behavior is inconsistent: the UI tells user that nova-network is used, while this is not true. No error occur in this case though.

26.2.7 Developer impact

The application developers may need to modify their apps to use the new feature. Patch [2] may be used as an example. Existing apps will not be affected, they will just have the old default behavior.

26.2.8 Murano-dashboard / Horizon impact

The whole change proposed in this spec is a change of murano-dashboard. No other components are modified.

26.3 Implementation

26.3.1 Assignee(s)

Primary assignee: ativelkov

Other contributors: ddovbii

26.3.2 Work Items

- Implement the shared logic to retrieve and filter the list of networks
- Implement the DynamicUI control to select networks in apps
- Add a dropdown field to a static Create Environment form to select the default network of the environment.
- Add the support of the new control in the existing murano apps in murano-apps repository.

26.4 Dependencies

- Include specific references to specs and/or blueprints in murano, or in other projects, that this one either depends on or is related to.
- If this requires functionality of another project that is not currently used by Murano, document that fact.
- Does this feature require any new library dependencies or code otherwise not included in OpenStack? Or does it depend on a specific version of library?

26.5 Testing

There should be an acceptance testing implemented on this feature:

- We should test deploying the apps with existing network selected and with the default option.
- Modified application (for example [2]) should be deployed both with "Auto" as instance network or with some existing network selected.
- The test cases above should verify the ability to assign floating IPs to the VMs
- The networks being used as an options for the manual selection should be connected to a router uplinked to the external network (otherwise app deployment will fail). Also the DNS nameservers has to be manually assigned on those networks.

26.6 Documentation Impact

A new Dynamic UI field type has to be documented at *Dynamic UI definition specification* guide at [3]

26.7 References

- [1] https://wiki.openstack.org/wiki/Murano/Specifications/Network_Management
- [2] https://review.openstack.org/#/c/201659/
- [3] https://murano.readthedocs.org/en/latest/draft/appdev-guide/muranopackages/dynamic_ui.html#dynamicuispec

CHAPTER 27

Indices and tables

- *search*