Murano Developers Guide

Murano Developers Guide

v0.3

Table of Contents

List of Figures

List of Tables

Chapter 1. Overview

Welcome to Murano Project. Full information about Murano in Openstack wiki page [https://wiki.openstack.org/wiki/Murano].

Murano Project introduces an application catalog, which allows application developers and cloud administrators to publish various cloud-ready applications in a browsable categorised catalog, which may be used by the cloud users (including the unexperienced ones) to pick-up the needed applications and services and composes the reliable environments out of them in a “push-the-button” manner. Key goal is to provide UI and API which allows to compose and deploy composite environments on the Application abstraction level and then manage their lifecycle

Intended Audience

This guide is intended to individuals who want to contribute to our project.

Document Change History

This version of the Murano Manual replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	**Summary of Changes**
April. 4, 2013	• Initial document creation
September. 4, 2013	• Update for Release-0.2
December. 9, 2013	• Update for Release-0.4
February. 5, 2013	• Update for Release-0.4.1

Chapter 2. Install Murano

This chapter describes Murano services installation in virtual environment.

Note that all Murano modules can be downloaded from our page [https://launchpad.net/murano/] on launchpad.

Pre-Requisites

Murano supports the following operating systems:

1. Ubuntu 12.04
2. RHEL/CentOS 6.4

These system packages are required for Murano:

Ubuntu

1. gcc
2. python-pip
3. python-dev
4. libxml2-dev
5. libxslt-dev
6. libffi-dev

CentOS

1. gcc
2. python-pip
3. python-devel
4. libxml2-devel
5. libxslt-devel
6. libffi-devel

Note

Before installing any packages make sure that your system is updated and upgraded.

Note

To deploy web farm services Neutron LBaaS (Neutron extension package) should be installed in Openstack

Installing with virtual environment

For a local development, all Murano components can be installed in a virtual environment. First thing you need to do is to install check that prerequisites are installed, system is updated and upgraded. Next, install virtualenv package if you don't have one:

```
sudo pip install virtualenv
```

Murano Api

- Check out git repository with murano component:

```
git clone https://github.com/stackforge/murano-api
```

- Execute a script located at the `murano-api/tools` directory to create virtual environment automatically:

```
cd murano-api && python ./tools/install_venv.py
```

- Copy murano-api sample config file located at `etc/murano/murano-api.conf.sample` to `etc/murano/murano-api.conf`. In config file set up keystone endpoint in Openstack installation. Other settings are described in 'Murano Admin Guide'. Config `etc/murano/murano-api-paste.conf` doesn't require any modification.
- And finally run Murano API:

```
./tools/with_venv.sh python muranoapi/cmd/api.py --config-file=./etc/murano/murar
```

Murano Conductor

- Check out git repository with murano component:

```
git clone https://github.com/stackforge/murano-conductor
```

- Execute the following script located at the `murano-conductor/tools` directory to create virtual environment automatically:

```
cd murano-conductor && python ./tools/install_venv.py
```

- Copy Murano Conductor sample config file located at `etc/murano/conductor.conf.sample` to `etc/murano/conductor.conf`. In config file set up keystone endpoint in Openstack installation. All other possible configuration described in the Murano Admin Guide.
- Run Murano Conductor:

```
./tools/with_venv.sh python muranoconductor/cmd/run.py --config-file=./etc/murano
```

Murano Repository

- Check out git repository with murano component:

```
git clone https://github.com/stackforge/murano-repository
```

- Execute a script located at the `murano-repository/tools` directory to create virtual environment automatically:

```
cd murano-repository && python ./tools/install_venv.py
```

- Copy a config file `etc/murano/murano-repository.conf.sample` to `etc/murano/murano-repository.conf` and point out Keystone parameters.
- And finally run Murano Repository:

```
./tools/with_venv.sh python muranorepository/cmd/run.py --config-file=./etc/murano
```

Murano Dashboard

- Check out git repository with murano component:

```
git clone https://github.com/stackforge/murano-dashboard
```

- Execute a script located at the `murano-dashboard/tools` directory to create virtual environment automatically:

```
cd murano-dashboard && python ./tools/install_venv.py
```

- Murano is a plugin for a Openstack dashboard. So need to install it and it's dependency:
  -
    ```
    # ./tools/with_venv.sh pip install https://github.com/openstack/horizon/archive
    ```

    Ubuntu

    ```
    # apt-get install nodejs
    ```

    CentOS

```
# yum install nodejs
```

- To configure Murano Dashboard copy example config file:

```
# cp muranodashboard/local/local_settings.py.example muranodashboard/local/local_
```

and set in just copied file the following parameters:

- *MURANO_API_URL* Required if murano-api service is not registered in keystone. Provide valid URL where Murano API is running
- *MURANO_METADATA_URL* Required if murano-repository service is not registered in keystone. Provide valid URL where Murano Repository service is running
- *NETWORK_TOPOLOGY*
  - network configuration. Duplicates murano-conductor setting (routed - default, flat, nova).

Since all UI forms are described in Metadata Repository and kept in Django sessions - default Memcached as a session backend is not suitable for a dynamic UI and we need to use Database backend. (It's already set up in murano settings file). But this requires to synchronize database with the specified settings:

```
./tools/with_venv.sh ./manage.py syncdb
```

- *Run Murano Dashboard:* To start the Murano development server use the Django manage.py utility with the context of the virtual environment:

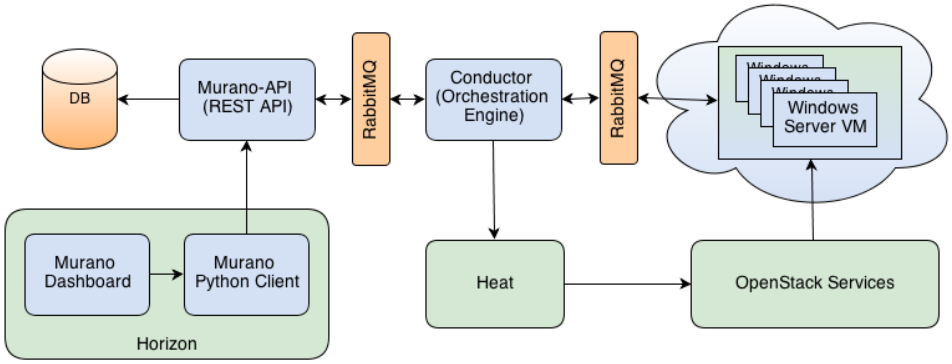
```
./tools/with_venv.sh ./manage.py runserver 0.0.0.0:8080
```

Chapter 3. Architecture

The Murano Service communicates with the following OpenStack components:

- Horizon - provides a GUI with ability to use all Murano features;
- Keystone - authenticates users and provides the security token that is used to work with OpenStack, hence limiting the user abilities in Murano based on OpenStack privileges;
- Heat - is used to provision VMs and other OpenStack resources
- Glance - stores VM images, with preconfigured OS and a containing sets of initial scripts
- Quantum - provides the network configuration API
- Agent - provides agent functionality to communicate with the Orchestration Engine and executes tasks on VMs

Figure 3.1. Architecture



REST API

Murano exposes a service endpoint for communication with a client. It exposes API functions to manipulate objects such as environment and service. This component is responsible for translating API function parameters to Object Model attributes and propagating the deployment status from the Orchestration Engine.

Object Model

An internal representation of Services and Environments. All attributes and entities are described in the API specification.

Orchestration Engine

This is the core component which evaluates Object Model changes and creates a plan for implementing these changes on the instances or in the cloud. This component will support extensions via plug-ins. Plugins can add new services and extend existing services for integration.

Metadata Repository

Murano Repository operates with metadata objects: store, display, group and use for deployment. Custom objects can be uploaded to the Murano Metadata Server and these data will be accessible to all Murano components.

Integration with Heat

Heat is a cloud resource management engine that allows you to manipulate resources that represent OpenStack entities (Security Groups, Instances, Floating IPs, Volumes, etc.) and some entities such as AutoScaling groups from a single point of control.

OpenStack resource provisioning is one of the steps required for environment deployment and Heat will be used for that purpose. Heat allows you to define all OpenStack resources in a single document that will be easy to maintain and will not require resorting to multiple OpenStack APIs while keeping the software configuration separate.

Chapter 4. API Specification

Revision Date	Summary of Changes
February 4, 2013	• Initial
February 22, 2013	• Enhance API with latest architecture changes
March 06, 2013	• Fix specification according to remarks from Dmitry Teselkin
Jun 06, 2013	• ASP.NET Application, Web Server Farm and ASP.NET Application Farm Services Added, uri/address/endpoint corrections, hostname assignment section added

Introduction

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

Glossary

For detailed information about entities and terms used in this document, please refer first to architecture.

Environment	Environment is a set of logically related Services managed by a single tenant. Environment defines Windows environment boundaries. Services within single Environment may comprise some complex configuration while Services in different Environments are always independent from one another. Each Environment is associated with single OpenStack project (tenant).
Service	Service is building block of Windows environment. Service is a set of one or more Virtual Machines sharing a common purpose and configured together. Each service belongs to a single Environment and single Service Type. Services are comprised from one or more Service Units.
Service Type	Service type is definition for describing set of features exposed by service.
Service Unit	Service Units are the actual Windows Server VMs instantiated by OpenStack and then configured according to its Service Type (this may also correspond to one of predefined Windows Server roles).
Service Metadata	Service Metadata is a JSON-encoded definition of Environment, its Services and their Service Units along with all their attributes. Service Metadata may describe both current and the intended state of the Environment.
Session	All changes to environment done in scope of Session. After all changes to Environment state are accumulated, changes actually are applied only after session is deployed.

Return codes and errors

All REST API calls return the natural HTTP response codes for the operations, e.g. a successful GET returns a HTTP 200, a successful PUT returns a HTTP 201, a GET for a non-existent entity returns HTTP 404, unauthorized operations return HTTP 401 or HTTP 403, internal errors return HTTP 500.

Response of POSTs and PUTs

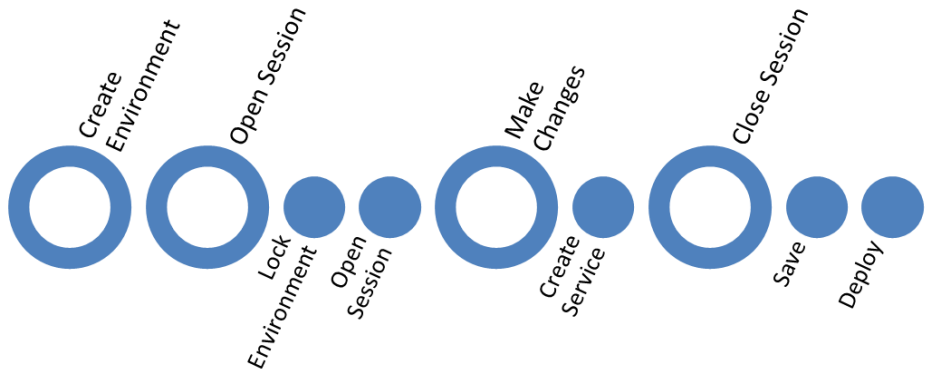
All POST and PUT requests by convention should return the created object (in the case of POST, with a generated ID) as if it was requested by GET.

Authentication

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

Workflow

Figure 4.1. Sample Workflow



Let’s review a sample workflow (series of API calls) for creating new Environment with Active Directory Service deployment:

1. POST /environments/ - Creating new Environment
2. POST /environments/id/configure – Creating new configuration session for Environment
3. POST /environments/id/services – Creating new ActiveDirectory service
4. POST /environments/id/sessions/session_id/deploy – Saving and deploying changes

Hostname assignment

Each Service Object definition may have an attribute "unitNamingPattern" that is used to control hostnames that will be assigned to spawned VM instances of the service.

Hostname pattern has the form of "name#" where "#" character is replaced with sequential number of unit within the service (starting with 1) and all other characters remain intact. For example Service with unitNamingPatter equal to "ad#-loc" will have units with hostnames "ad1-loc", "ad2-loc" etc.

"unitNamingPattern" attribute is optional. If it is omitted then a unique random hostname would be assigned.

API

Environment API

This section describes API calls for Environment management.

Get a List of existing Environments

Table 4.1. Environment Object

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

Call

Table 4.2. GET /environments Call

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

Payload

None

Returns

This call returns list of environments. Only the basic properties are returned. For details see "Get Environment Instance Detailed Information":

```
{
   "environments": [
       {
           "id": "0ce373a477f211e187a55404a662f968",
           "name": "dc1",
           "created": "2010-11-30T03:23:42Z",
           "updated": "2010-11-30T03:23:44Z",
           "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
```

```
            "version": 1,
            "status": "ready"
        },
        {
            "id": "c697bd2429304820a928d145aa42af59",
            "name": "dc2",
            "created": "2010-11-30T03:23:42Z",
            "updated": "2010-11-30T03:23:44Z",
            "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
            "version": 2,
            "status": "deploying"
        }
    ]
}
```

Create Environment instance

Table 4.3. Environment Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name

Call

Table 4.4. POST /environments Call

Method	URI	Description
POST	/environments	Create new Environment

Payload

```
{
  "name": "env1"
}
```

Returns

This call returns created environment:

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "env1",
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:44Z",
```

```
    "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
    "version": 0
}
```

Update Environment Instance

Table 4.5. Environment Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name

Call

Table 4.6. PUT /environments/<id> Call

Method	URI	Description
PUT	/environments/<id>	Update properties of Environment instance

Table 4.7. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources

Payload

```
{
  "name": "env1-changed"
}
```

Returns

This call returns modified environment object:

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "env1-changed",
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:44Z",
    "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
    "version": 0
}
```

Get Environment Instance Detailed Information

Call

Table 4.8. GET /environments/<id> Call

Method	URI	Description
GET	/environments/<id>	Returns detailed information about Environment including child entities

Table 4.9. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources

Payload

None

Returns

This call returns environment object with underlying services:

```
{
    "environments": [{
         "id": "0ce373a477f211e187a55404a662f968",
         "name": "dc1",
         "created": "2010-11-30T03:23:42Z",
         "updated": "2010-11-30T03:23:44Z",
         "tenant_id": "0849006f7ce94961b3aab4e46d6f229a",
         "version": 1,
         "status": "deploying",
         "services": [
             "activeDirectories": [{
                 "id": "96365940588b479294fe8e6dc073db04",
                 "name": "acme.dc",
                 "created": "2010-11-30T03:23:42Z",
                 "updated": "2010-11-30T03:23:44Z",
                 "status": "deploying",
                 "units": [{
                     "id": "d08887df15b94178b244904b506fe85b",
                     "isMaster": true,
                     "location": "west-dc"
                 }, {
                     "id": "dcf0de317e7046bea555539f19b8ea84",
                     "isMaster": false,
                     "location": "west-dc"
                 }]
             }]
         ]
    }]
```

```
}
```

Remove Environment

Call

Table 4.10. DELETE /environments/<id> Call

Method	URI	Description
DELETE	/environments/<id>	Remove specified Environment.

Table 4.11. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources

Payload

None

Returns

None

Environment Configuration API

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

Table 4.12. Configuration Session Object

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

Configure Environment / Open session

During this call new working session is created, and session ID should be sent in header (X-Configuration-Session) to all next API calls.

Call

Table 4.13. POST /environments/<id>/configure Call

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

Table 4.14. Error Response Codes

Code	Description
403	Could not open session for environment, environment has deploying status

Payload

None

Returns

This call returns created session:

```
{
    "id": "4aecdc2178b9430cbbb8db44fb7ac384",
    "environment_id": "4dc8a2e8986fa8fa5bf24dc8a2e8986fa8",
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:54Z",
    "user_id": "d7b501094caf4daab08469663a9e1a2b",
    "version": 12,
    "state": "open"
}
```

Deploy changes from Session

Call

Table 4.15. POST /environments/<id>/sessions/<sessionId>/deploy Call

Method	URI	Description
POST	/environments/<id>/sessions/<sessionId>/deploy	Deploying changes made in session with specified <sessionId>

Table 4.16. Error Response Codes

Code	Description
403	Session is invalid
403	Session is already deployed or deployment is in progress

Payload

None

Returns

None

Get session information

Call

Table 4.17. GET /environments/<id>/sessions/<sessionId> Call

Method	URI	Description
GET	/environments/<id>/sessions/<sessionId>	Getting details about session with specified <sessionId>

Table 4.18. Error Response Codes

Code	Description
401	User is not authorized to access this session
403	Session is invalid

Payload

None

Returns

This call returns session information:

```
{
    "id": "4aecdc2178b9430cbbb8db44fb7ac384",
    "environment_id": "4dc8a2e8986fa8fa5bf24dc8a2e8986fa8",
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:54Z",
    "user_id": "d7b501094caf4daab08469663a9e1a2b",
    "version": 0,
    "state": "deploying"
}
```

Delete Session

Call

Table 4.19. DELETE /environments/<id>/sessions/<sessionId> Call

Method	URI	Description
DELETE	/environments/<id>/sessions/<sessionId>	Delete session with specified <sessionId>

Table 4.20. Error Response Codes

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

Payload

None

Returns

None

Services API

This section describes API calls for managing all types of services.

Calls and Endpoints

GET

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

Call

Table 4.21. GET /environments/<id>/services Call

Method	URI
GET	/environments/<id>/services

Table 4.22. Headers

Name	Type	Required	Description
X-Configuration-Session	string	no	ID of valid configuration session

Payload

None

Returns

Json Object

PUT

Using PUT calls user can update or add any attribute on any service.

Call

Table 4.23. PUT /environments/<id>/services Call

Method	URI
PUT	/environments/<id>/services

Table 4.24. Headers

Name	Type	Required	Description
X-Configuration-Session	string	yes	ID of valid configuration session

Payload

Json Object

Returns

Json Object

POST

POST calls used for adding new elements to lists with objectes.

Call

Table 4.25. POST /environments/<id>/services Call

Method	URI
POST	/environments/<id>/services

Table 4.26. Headers

Name	Type	Required	Description
X-Configuration-Session	string	yes	ID of valid configuration session

Payload

Json Object

Returns

Json Object

DELETE

User can remove any attribute or list item using DELETE calls.

Call

Table 4.27. DELETE /environments/<id>/services Call

Method	URI
DELETE	/environments/<id>/services

Table 4.28. Headers

Name	Type	Required	Description
X-Configuration-Session	string	yes	ID of valid configuration session

Payload

None

Returns

None

Example Calls using Web Server as example

Please use this example for constructing general CRUD calls to services.

Get a List of existing services

Call

Table 4.29. GET /environments/<id>/services Call

Method	URI	Description
GET	/environments/<id>/services	Get a list of existing WebServers

Table 4.30. Headers

Name	Type	Required	Description
X-Configuration-Session	string	no	ID of valid configuration session

Payload

None

Returns

This call returns list of services:

```
[
  {
    "id": "0ce373a477f211e187a55404a662f968",
    "name": "frontend",
    "type": "webServer".
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:44Z",
    "domain": "ACME",
    "uri": "http://10.0.0.2",
    "units": [{
        "id": "1bf3491c409b4541b6f18ea5988a6437",
        "address": "10.0.0.2",
```

```
        "location": "west-dc"
    }]
  },
  {
    "id": "c697bd2429304820a928d145aa42af59",
    "name": "backend",
    "type": "webServer".
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:44Z",
    "domain": "ACME",
    "uri": "http://10.0.0.3",
    "units": [{
        "id": "eb32f97866d24001baa430cb34e4049f",
        "address": "10.0.0.3",
        "location": "west-dc"
    }]
  }
```

Create Web Server instance

Call

Table 4.31. POST /environments/<id>/services Call

Method	URI	Description
POST	/environments/<id>/services	Create new Web Server

Table 4.32. Headers

Name	Type	Required	Description
X-Configuration-Session	string	yes	ID of valid configuration session

Payload

```
{
  "name": "frontend",
  "type": "webServer",
  "adminPassword": "password",
  "domain": "acme.dc",
  "units": [{
    "location": "west-dc"
  }]
}
```

Returns

This call returns created web server:

```
{
    "id": "ce373a477f211e187a55404a662f968",
    "name": "frontend",
    "type": "webServer",
    "created": "2010-11-30T03:23:42Z",
    "updated": "2010-11-30T03:23:44Z",
    "domain": "ACME",
    "units": [{
        "id": "1bf3491c409b4541b6f18ea5988a6437",
        "location": "west-dc"
    }]
}
```

Remove Web Server

Call

Table 4.33. DELETE /environments/<id>/services/<service_id> Call

Method	URI	Description
DELETE	/environments/<id>/services/<service_id>	Remove specified service.

Table 4.34. Headers

Name	Type	Required	Description
X-Configuration-Session	string	yes	ID of valid configuration session

Payload

None

Returns

None

Shared Attributes

This section describes attributes common to all services.

Request Object Specs

Table 4.35. Service Object Specs

Attribute	Type	Description
id	string	Unique ID
type	string	Service Type
created	datetime	Creation date and time in ISO format

Attribute	Type	Description
updated	datetime	Modification date and time in ISO format
unitNamingPattern	string	Unit instances naming pattern
availabilityZone	string	Availability where service is located
flavor	string	Active Directory Unit object
osImage	string	Name of image used to power instance

Create Object Specs

Table 4.36. Service Object Specs

Attribute	Type	Description
type	string	Service Type
unitNamingPattern	string	Unit instances naming pattern
availabilityZone	string	Availability where service is located
flavor	string	Active Directory Unit object
osImage	string	Name of image used to power instance

Active Directory Specs

This section describes objects specs for Active Directory service.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: activeDirectory

Request Object Specs

Table 4.37. Active Directory Object

Attribute	Type	Description
name	string	User-friendly name
domain	string	Domain name
units	object	Active Directory Unit object

Table 4.38. Active Directory Unit Object

Attribute	Type	Description
id	string	Unique ID
isMaster	boolean	true for primary domain controller, false otherwise

Create Object Specs

Table 4.39. Active Directory Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name

Attribute	Type	Required	Description
adminPassword	string	yes	Password from domain administrator account
domain	string	yes	Domain name
units	object	yes	Active Directory Unit object

Table 4.40. Active Directory Unit Object

Attribute	Type	Required	Description
isMaster	boolean	yes	true for primary domain controller, false otherwise
recoveryPassword	string	yes	Recovery password

Web Server Specs

This section describes API calls for managing Windows web-server software – IIS.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: webServer

Request Object Specs

Table 4.41. Web Server Object

Attribute	Type	Description
name	string	User-friendly name
uri	string	URI of the Service
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member
units	object	Web Server Unit object

Table 4.42. Web Server Unit Object

Attribute	Type	Description
id	string	Unique ID

Create Object Specs

Table 4.43. Web Server Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
domain	string	no	Domain name
units	object	yes	Web Server Unit object

Table 4.44. Web Server Unit Object

Attribute	Type	Required	Description
-	-	-	-

ASP.NET Application Specs

This section describes API calls for managing ASP.NET Applications

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: aspNetApp

Request Object Specs

Table 4.45. ASP.NET Application Object

Attribute	Type	Description
name	string	User-friendly name
repository	string	URL of git repository containing the application source files
uri	string	URI of the Service
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member
units	object	ASP.NET Application Unit object

Table 4.46. ASP.NET Application Unit Object

Attribute	Type	Description
id	string	Unique ID

Create Object Specs

Table 4.47. ASP.NET Application Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
repository	string	yes	URL of git repository containing the application source files
domain	string	no	Domain name
units	object	yes	ASP.NET Application Unit object

Table 4.48. ASP.NET Application Unit Object

Attribute	Type	Required	Description
-	-	-	-

Web Server Farm Specs

This section describes API calls for managing Web Server (IIS) Web Farm services

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: webServerFarm

Request Object Specs

Table 4.49. Web Server Farm Object

Attribute	Type	Description
name	string	User-friendly name
uri	string	URI of the Service
loadBalancerPort	integer	Port number of the Farm
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member
units	object	Web Server Farm Unit object

Table 4.50. Web Server Farm Unit Object

Attribute	Type	Description
id	string	Unique ID

Create Object Specs

Table 4.51. Web Server Farm Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
loadBalancerPort	integer	yes	Port number for the Farm
domain	string	no	Domain name
units	object	yes	Web Server Farm Unit object

Table 4.52. Web Server Farm Unit Object

Attribute	Type	Required	Description
-	-	-	-

ASP.NET Application Farm Specs

This section describes API calls for managing ASP.NET Web Farm Application Services

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: aspNetAppFarm

Request Object Specs

Table 4.53. ASP.NET Application Farm Object

Attribute	Type	Description
name	string	User-friendly name
uri	string	URI of the Service

Attribute	Type	Description
repository	string	URL of git repository containing the application source files
loadBalancerPort	integer	Port number of the Farm
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member
units	object	ASP.NET Application Farm Unit object

Table 4.54. ASP.NET Application Farm Unit Object

Attribute	Type	Description
id	string	Unique ID

Create Object Specs

Table 4.55. ASP.NET Application Farm Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
repository	string	yes	URL of git repository containing the application source files
loadBalancerPort	integer	yes	Port number for the Farm
domain	string	no	Domain name
units	object	yes	ASP.NET Application Farm Unit object

Table 4.56. ASP.NET Application Farm Unit Object

Attribute	Type	Required	Description
-	-	-	-

MS SQL Server Specs

This section describes API calls for managing MS SQL Server Service.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: msSqlServer

Request Object Specs

Table 4.57. MS SQL Server Object

Attribute	Type	Description
name	string	User-friendly name
mixedModeAuth	bool	Use LDAP to access SQL Server
saPassword	string	SQL Server admin password
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member

Attribute	Type	Description
adminPassword	string	Domain password
units	object	SQL Server Unit object

Table 4.58. SQL Server Unit object

Attribute	Type	Description
id	string	Unique ID

Create Object Specs

Table 4.59. SQL Server Unit object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
mixedModeAuth	bool	yes	Use LDAP to access SQL Server
saPassword	string	no	SQL Server admin password
domain	string	no	Domain name
adminPassword	string	no	Domain admin password
units	object	yes	SQL Server Unit object

Table 4.60. MS SQL Server Unit Object

Attribute	Type	Required	Description
-	-	-	-

MS SQL Server Cluster Specs

This section describes API calls for managing MS SQL Server Cluster Service.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: msSqlClusterServer

Request Object Specs

Table 4.61. MS SQL Server Object

Attribute	Type	Description
name	string	User-friendly name
externalAD	boolean	Is Active Directory is configured by the System Administrator
domainAdminUserName	string	Username for Active Directory user with admin role, if Active Directory is configured by the System Administrator
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member

Attribute	Type	Description
mixedModeAuth	bool	Use LDAP to access SQL Server
clusterIp	string	A valid IPv4 fixed IP
clusterName	string	Service name for new SQL Cluster service
agGroupName	string	Availability Group Name
agListenerName	string	FQDN name of a new DNS entry for AG Listener endpoint
agListenerIP	string	Availability Group Listener IP
units	object	MS SQL Server Cluster Unit object

Table 4.62. MS SQL Server Cluster Unit object

Attribute	Type	Description
id	string	Unique ID
isMaster	boolean	Is this node master?
isSync	boolean	Is this node in sync mode?

Create Object Specs

Table 4.63. MS SQL Server Cluster Unit object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
adminPassword	string	yes	Domain password
externalAD	boolean	yes	Is Active Directory is configured by the System Administrator
domainAdminUserName	string	no	Username for Active Directory user with admin role, if Active Directory is configured by the System Administrator
domainAdminPassword	string	no	Password for Active Directory user with admin role, if Active Directory is configured by the System Administrator
domain	string	no	Domain name. This attribute may be empty/null/ omitted if machine is not a domain member
mixedModeAuth	bool	yes	Use LDAP to access SQL Server
saPassword	string	no	SQL Server admin password
clusterIp	string	yes	A valid IPv4 fixed IP
clusterName	string	yes	Service name for new SQL Cluster service
agGroupName	string	yes	Availability Group Name
agListenerName	string	yes	FQDN name of a new DNS entry for AG Listener endpoint
agListenerIP	string	yes	Availability Group Listener IP
sqlServicePassword	string	yes	User password that will be created to manage cluster instances

Attribute	Type	Required	Description
units	object	yes	SQL Server Cluster Unit object

Table 4.64. MS SQL Server Cluster Unit Object

Attribute	Type	Required	Description
isMaster	boolean	yes	Is this node master?
isSync	boolean	yes	Is this node in sync mode?

Linux Apache Specs

This section describes objects specs for Linux Apache Service.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: linuxApacheService

Request Object Specs

Table 4.65. Linux Apache Object

Attribute	Type	Description
name	string	User-friendly name
deployApachePHP	boolean	Enable PHP support on installed service
units	object	Linux Apache Unit object

Table 4.66. Linux Apache Unit Object

Attribute	Type	Description
-	-	-

Create Object Specs

Table 4.67. Linux Apache Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
deployApachePHP	boolean	no	Enable PHP support on installed service
units	object	yes	Linux Apache Unit object

Table 4.68. Linux Apache Unit Object

Attribute	Type	Required	Description
-	-	-	-

Linux Telnet Specs

This section describes objects specs for Linux Telnet service.

Please, refer to Shared Attributes article for shared/common attributes specification.

Type: linuxTelnetService

Request Object Specs

Table 4.69. Linux Telnet Object

Attribute	Type	Description
name	string	User-friendly name
units	object	Linux Telnet Unit object

Table 4.70. Linux Telnet Unit Object

Attribute	Type	Description
-	-	-

Create Object Specs

Table 4.71. Linux Telnet Object

Attribute	Type	Required	Description
name	string	yes	User-friendly name
units	object	yes	Linux Telnet Unit object

Table 4.72. Linux Telnet Unit Object

Attribute	Type	Required	Description
-	-	-	-

Chapter 5. Metadata-Repository API Specification

Revision Date	Summary of Changes
December, 9th, 2013.	• Initial

Introduction

Murano Metadata Repository is used to operate (store, display, group and use for deployment) the metadata objects. It is the simplest possible storage for metadata objects, without any significant modifications in the existing Murano components.

Glossary

Metadata object type	Name of supported metadata objects. For now there are 6 of them: • Service Manifest - *manifests* • Workflow - *workflow* • Heat templates - *heat* • Agent templates - *agent* • Scripts - *scripts* • UI definitions - *ui*
Service Manifest	Defines basic properties of the services. It binds together metadata objects by defining the names of the files required for a given service. The manifest should be defined as a yaml file.
Workflow	Definition of workflow rules and actions needed to deploy a service
Heat templates	A parametrized Heat template
Agent templates	A template for Agent execution plan (contains script imports and commands to execute)
Scripts	A script which is referenced in execution plan and is executed on the Agent. Conductor loads the content of this file, base-64 encodes it and injects it into the execution plan
UI definitions	A definition of the dynamic User Interface for service configuration in the Dashboard.

v1

This chapter defines all available api calls for v1 version. All definitions are listed without version prefix. Just add "v1" to the beginning of any call. Example: v1/client/ui

Return codes and errors

All REST API calls return the natural HTTP response codes for the operations, e.g. a successful GET returns a HTTP 200, a successful PUT returns a HTTP 201, a GET for a non-existent entity returns HTTP 404, unauthorized operations return HTTP 401 or HTTP 403, internal errors return HTTP 500.

Authentication

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

Client API

This section describes API calls for Metadata Repository clients.

Get archive for a specified client

Call

Table 5.1. Get archive for a specified client

Method	URI	Description
GET	/client/<client_type>	Returns archive with all metadata objects for specified client for all active services

Table 5.2. Parameters

Parameter	Description	Required
client_type	One of supported Murano Repository client: ui - Murano Dashboard; conductor - Murano Conductor	true

Table 5.3. Response Codes

Code	Description
200	Archive with metadata objects successfully composed and sent
401	User is not authorized to access this tenant resources
404	Client_type is not supported
304	Returns empty body if the hash matches the hashsum of current version of the metadata objects archive

Payload

None

Returns

This call returns an array of bytes represents tar.gz archive

Download the service definition

Table 5.4. Download the service definition

Method	URI	Description
GET	/client/services/<full_service_name>	Returns a tar.gz archive containing service definition along with all referenced metadata

Method	URI	Description
		objects, directory structure of metadata objects is preserved

Table 5.5. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.6. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	Service with specified full_service_name does not exist(there is no manifest file for this service)

Payload

None

Returns

This call returns an array of bytes represents tar.gz archive with all files required for given service

Admin API

List Metadata Objects

Table 5.7. List Metadata Objects

Method	URI	Description
GET	/admin/<object_type>/<path>	Returns a json-array containing the filenames of the metadata objects of a given type in a given subdirectory. Includes all subdirectories and their content

Table 5.8. Parameters

Parameter	Description	Required
object_type	One of supported Murano Repository objects: workflow, scripts, ui, agent, heat, manifests	true
path	May define a directory (which is present under the root directory of the appropriate type of metadata). If specified, the	false

Parameter	Description	Required
	output will include only files and subdirectories of this directory (the path will be relative to it).	

Table 5.9. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	Metadata object type or directory specified in path parameter are not exist

Payload

None

Returns

This call returns list all files for specified object type and path

```
{
    "ui": [
        "MsSqlClusterServer.yaml",
        "AspNetAppFarm.yaml",
        "LinuxApache.yaml",
        "WebServerFarm.yaml",
        "Demo.yaml",
        "WebServer.yaml",
        "ActiveDirectory.yaml",
        "MsSqlServer.yaml",
        "AspNetApp.yaml",
        "LinuxTelnet.yaml"
    ]
}
```

Get Metadata Object

Table 5.10. Get Metadata Object

Method	URI	Description
GET	/admin/<object_type>/<file_name>	Returns a file containing specified metadata object.

Table 5.11. Parameters

Parameter	Description	Required
object_type	One of supported Murano Repository objects: workflow, scripts, ui, agent, heat, manifests	true

Parameter	Description	Required
file_name	Defines a filename of the file to be downloaded. Should be relative to the root folder of the given type	true

Table 5.12. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	Metadata object type or directory specified in path parameter are not exist

Payload

None

Returns

This call returns an array of bytes represents a file

Upload a metadata object

Table 5.13. Upload a metadata object

Method	URI	Content-Type	Description
POST	/admin/<object_type>/<path>	multipart/form-data	Submits a new metadata file of a given type. Attribute "file" that points to the uploading file should be in request.FILES section
POST	/admin/<object_type>/<path>?filename=test.yaml	application/octet-stream	Submits a new metadata file of a given type. Filename parameter is required

Table 5.14. Parameters

Parameter	Description	Required
object_type	identifies the type of the new metadata object. May be one of the following: workflow, scripts, ui, agent, heat, manifests	true
path	May define a directory (which is present under the root directory of the appropriate type of metadata). If specified, the file will be placed into that dir. If omitted, the file will be placed into the root directory of the given metadata type. This parameter may not be defined for files of type “manifests”	false

Table 5.15. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The object_type is unknown or the path is set to an non-existing directory
400	The is no parameter 'file' in request.FILES(for multipart/form-data Content type) or there is no filename parameter in URL(for application/octet-stream)

Payload

File will be stored on Metadata Repository Server in a directory corresponds to given object type or in it's nested directory

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Create directory

Table 5.16. Create directory

Method	URI	Description
PUT	/admin/<object_type>/ <full_directory_name>	Creates a new directory in the given path

Table 5.17. Parameters

Parameter	Description	Required
object_type	Identifies the type of the new metadata object. May be one of the following: workflow, scripts, ui, agent, heat.	true
full_directory_name	Defines a name of a new directory. If any components of the path are missing, they will be created. If the directory already exists, nothing will happen.	true

Table 5.18. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The object_type is unknown
403	The object_type is "manifest"

Payload

None

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Delete metadata object or directory

Table 5.19. Delete metadata object or directory

Method	URI	Description
DELETE	/admin/ <object_type>/ <full_path_to_object>	Deletes a metadata object or a directory. Directory should be empty to be deleted

Table 5.20. Parameters

Parameter	Description	Required
object_type	Identifies the type of the new metadata object. May be one of the following: workflow, scripts, ui, agent, heat, manifests.” typed cannot be specified, as subdirectories for the service manifests cannot be created	true
full_path_to_object	Specifies the full path a file or a directory .	true

Table 5.21. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The object_type is unknown or the full_path_to_object is set to an non-existing directory or object
403	The object specified by full_path_to_object is a not empty directory

Payload

File or directory with the given name permanently deletes from Murano Metadata Repository Server

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

List all the services

Table 5.22. List all the services

Method	URI	Description
GET	/admin/services	Returns a json-array containing all the info (version, display name, author, etc) about each service except their references to other metadata objects

Payload

None

Returns

In case of successful result this call returns a json:

```
{
    "services": [
        {
            "author": "Mirantis Inc.",
            "description": "<strong> Demo Service </strong>
                            shows how Murano is working.",
            "enabled": true,
            "full_service_name": "demoService",
            "service_display_name": "Demo Service",
            "service_version": "",
            "valid": true,
            "version": "0.1"
        },
        {
            "author": "Mirantis Inc.",
            "description": "<strong> The  Internet Information Service </strong>
                           sets up an IIS server and joins it into an existing dom
            "enabled": true,
            "full_service_name": "webServer",
            "service_display_name": "Internet Information Services",
            "service_version": 1,
            "valid": true,
            "version": 0.1
        }
   ]
}
```

List metadata objects of a given service

Table 5.23. List metadata objects of a given service

Method	URI	Description
GET	/admin/services/<full_service_name>	Returns a json-dictionary containing the filenames of the metadata objects referenced

Method	URI	Description
		by a given service. Filenames include all parent directories, all objects belonging to a certain %metadata_type% are grouped into an array value for of a %metadata_type% key.

Table 5.24. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.25. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The full_service_name is unknown

Payload

None

Returns

This call returns a json:

```
{
    "agent": [
        "Demo.template"
    ],
    "heat": [
        "Demo.template",
        "RouterInterface.template",
        "Network.template",
        "NNSecurity.template",
        "Param.template",
        "Subnet.template",
        "InstancePortWSubnet.template",
        "InstancePort.template"
    ],
    "scripts": [],
    "ui": [
        "Demo.yaml"
    ],
    "workflows": [
        "Networking.xml",
        "Common.xml",
        "Demo.xml"
    ]
```

```
}
```

List meta information of a given service

Table 5.26. List meta information of a given service

Method	URI	Description
GET	/admin/services/<full_service_name>/info	Returns a json-dictionary containing information of a given service

Table 5.27. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.28. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The full_service_name is unknown

Payload

None

Returns

This call returns a json:

```
{
    "author": "Mirantis Inc.",
    "description": "Demo Service shows how Murano is working.",
    "enabled": true,
    "full_service_name": "demoService",
    "service_display_name": "Demo Service",
    "service_version": "",
    "version": "0.1"
}
```

Switch service parameter: enabled/disabled

Table 5.29. Switch service parameter: enabled/disabled

Method	URI	Description
POST	/admin/services/<full_service_name>/toggle_enabled	Toggle current state of 'Enabled' parameter in service manifest file

Table 5.30. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.31. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The full_service_name is unknown

Payload

None

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Upload the complete service definition

Table 5.32. Upload the complete service definition

Method	URI	Content-Type	Description
POST	/admin/services	application/octet-stream	Uploads a tar.gz archive containing service definition to be added to catalog along with the referenced metadata objects. Metadata objects could be located in nested directories, the directory structure will be preserved on server-side
POST	/admin/services	multipart/form-data	Uploads a tar.gz archive containing service definition to be added to catalog along with the referenced metadata objects. Attribute "file" that points to the uploading file should be in request.FILES section

Table 5.33. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.34. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The full_service_name is unknown
400	Uploading file is not tar.gz or archive is invalid (archive structure not correspond to defined in config)

Payload

All files from archive are copied to corresponds location in Metadata Service

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Create or modify service definition

Table 5.35. Create or modify service definition

Method	URI	Description
PUT	/admin/services/<full_service_name>	Create or modify manifest file. Request data should be a json object If full_service_name is not specified new manifest file will be created. Otherwise existent service definition (manifest file) will be rewritten with the given data.

Table 5.36. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.37. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
400	If specified full_service_name is not equivalent to that parameter in json or there is no json data in request

Payload

New manifest created in manifests directory. Metadata Service

Example

Here are json that will create new service manifest in Metadata Repository (json for service update looks the same - all data will be overwritten with the new one). Note that service_display_name is a required parameter and full_service_name should correspond to`full_service_name` given in URL or skipped.

```
{
  "full_service_name": "new_service",
  "service_display_name" : "Test Service",
  "version": "0.1"
}
```

All other parameters will be set to default parameters.

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Delete service from repository

Table 5.38. Delete service from repository

Method	URI	Description
DELETE	/admin/services/<full_service_name>	Deletes a service from repository

Table 5.39. Parameters

Parameter	Description	Required
full_service_name	Identifies the type service (corresponds to full_service_name in manifest)	true

Table 5.40. Error Response Codes

Code	Description
401	User is not authorized to access this tenant resources
404	The full_service_name is unknown

Payload

Manifest for specified service deleted from

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Reset cache

Table 5.41. Reset cache

Method	URI	Description
POST	/admin/reset_caches	Clear server cache in case of manual data modification

Payload

None

Returns

In case of successful result this call returns a json:

```
{"result": "success"}
```

Chapter 6. Workflows XML DSL

XML DSL

Workflows are written using XML markup language. This XML has no fixed structure but instead XML tags are translated into Python function calls. Thus such XML can be considered as a simplified programming language.

Consider the following XML fragment:

```
<func arg1="value1" arg2="value2" />
```

This is an equivalent to `func(arg1='value1', arg2='value2')` in Python. Tag name is mapped to a function name, one that Murano Conductor knows. Each attribute corresponds to function argument.

XML functions may also have a body. It is evaluated to "body" argument. Thus

```
<func arg1="value1" arg2="value2">some text</func>
```

is translated to `func(arg1='value1', arg2='value2', body='some text')`

In example above all function arguments were constant values. But they also can be evaluated:

```
<foo arg="value">
    <bar/>
</foo>
```

turns to

```
body_value = bar()
foo(arg='value', body=body_value)
```

Tag body may consist of several function invocations. In such case all values would be concatenated. For example if

```
def foo(**kwargs):
  return "foo"

def bar(**kwargs):
  return "bar"
```

then `<func><foo/> - <bar/></func>` will result in `func(body = 'foo - bar')`

Function parameters can also be function calls using <parameter> tag

```
<foo arg="value"/>
```

can be rewritten as

```
<foo>
    <parameter name="arg">value</parameter>
</foo>
```

while that later form is more verbose it allows having dynamically evaluated values as function arguments:

```
<foo>
    <parameter name="arg"><bar/></parameter>
</foo>
```

Functions may also have other custom tags in their body that interpreted in a way different from plain function invocation.

DSL functions

There are a number of functions in Murano Conductor that can be used in XML DSL:

- `<true/>` - returns True
- `<false/>` - False
- `<null/>` - None
- `<text><foo/></text>` - converts body to string `(str(foo()))`
- list - form list (array) object -

  ```
  <list>
      <item>item1</item>
      <item>item2</item>
      <item><true/></item>
  </list>
  ```

  equals to `["item1", "item2", True]`

- map - form dictionary (map) object

  ```
  <map>
      <item name="key1">value1</item>
      <item name="key2">value2</item>
  </map>
  ```

  equals to `{ "key1": "value1", "key2": "value2" }` For both list and map functions names of item nodes ("item" in examples above) is irrelevant and can be changed to better match structure usage. For example

  ```
  <map>
      <set name="key"><null/></set>
  ```

```
</map>
```

Structures can be nested:

```
<list>
    <item>
        <map>
            <item name="name 1">value 1</item>
            <item name="name 2">value 2</item>
        </map>
    </item>
    <item>
        <map>
            <item name="name 1">value 3</item>
            <item name="name 2">value 4</item>
        </map>
    </item>
</list>
```

equals to

```
[
    { 'name 1': 'value 1', 'name 2': 'value 2' },
    { 'name 1': 'value 3', 'name 2': 'value 4' }
]
```

Workflows

Workflows are XML DSL scripts that describe the steps that conductor need to perform in order to deploy specified environment. All workflow constructs are just ordinary DSL functions similar to those described above.

Usually workflows extract some data from input environment definition (Object Model) and then invoke one of the actions with these data as a input arguments.

Actions are:

- Heat commands that update or delete Heat Stack: <update-cf-stack>, <delete-cf-stack>
- Send command to Murano Agent: <send-command>
- Report state to API: <report>

Workflow logic can be described in 6 steps:

1. Choose a node (set of nodes) to update. If none of the nodes can be updated we are done.
2. According to the current state of node (that is a part of input Object Model) choose appropriate command to execute (update Heat stack or issue PowerShell command)
3. Select appropriate information from Object Model and substitute it into chosen template
4. Execute command

5. Update Object Model according to command execution result

6. Go to step 1

Accessing Object Model

Object Model is a definition of environment that Murano Conductor was asked to deploy.

Lets take the following Object Model that describes Active Directory service with single controller for our further examples:

```
{
    "name": "MyDataCenter",
    "id": "adc6d143f9584d10808c7ef4d07e4802",
    "services": [ {
        "name": "TestAD",
            "type": "activeDirectory",
            "osImage": {"name": "ws-2012-std", "type": "ws-2012-std"},
            "availabilityZone": "nova",
            "flavor": "m1.medium",
            "id": "9571747991184642B95F430A014616F9",
        "domain": "acme.loc",
        "adminPassword": "SuperP@ssw0rd",
    "units": [ {
            "id": "273c9183b6e74c9c9db7fdd532c5eb25",
            "name": "dc01",
            "isMaster": true,
            "recoveryPassword": "SuperP@ssw0rd!2"
        } ]
     } ]
}
```

There are several functions to select values from Object Model and function to modify it so that the Workflow can persist the changes made during deployment.

All reads and writes to Object Model are relative to some cursor position which is managed by workflow rule. Lets call it current cursor position.

The simplest method of accessing Object Model is a select function. Suppose current cursor position points to a single unit of our single service. Then `<select path="name"/>` is "dc01" and `<select path="isMaster"/>` is True.

Path parameter may start with colon to indicate navigation to one level up or slash to go to Model root:

```
 <select path=":"/>
```

is

```
[
    {
        "id": "273c9183b6e74c9c9db7fdd532c5eb25",
        "name": "dc01",
        "isMaster": true,
```

```
        "recoveryPassword": "SuperP@ssw0rd!2"
    }
]
```

`<select path="::domain">` is "acme.loc" and `<select path="/name" />` is "MyDataCenter"

The path also supports drill-down notation: `<select path="::osImage.name">` is "ws-2012-std" `<select path="::units.0.name">` equals to `<select path="/services.0.units.0.name">` that is "dc01"

If the path does not exist then result of a select would be None: `<select path="::noSuchProperty"/>` = None

`<select/>` without path results in object pointed by current cursor position.

It also possible to select multiple values using JSONPath selection language:

```
 <select-all path="/$.services[?(@.type == 'activeDirectory')].units[*]"/>
```

- returns array of all units of all services of type 'activeDirectory'

JSONPath expressions by default select data relative to current cursor position and has no way for navigating up the Model tree. But Conductor has several improvements to JSONPath language:

- JSONPath expression may start with one or more colon characters to perform query relative to current cursor parent (grandparent etc.)
- JSONPath expression may also start with slash as in example above to query the whole Model from the tree root
- Expressions may reference nonexistent Model attributes in the same way as `<select/>` function does. Such attributes considered to have None values

`<select-single path="JSONPath expression"/>` is similar to `<select-all/>` , but returns only the first value matched by a JSONPath query or None.

Object Model can also be modified using `<set/>` function. `<set/>` is very similar to `<select/>` with the only difference in that `<select/>` writes values while `<select/>` reads them: `<set path="name">dc02</set>` changes unit name (eg. the object pointed by current cursor position) to "dc02". `<set>` can also introduce new attributes, event nested ones: `<set path="newProperty">value</set>` creates new attribute "newProperty" with given value, `<set path="state.property">value</set>` makes current unit have property `"state": { "property": "value" }`

In this case "state" is just a convention - a reserved property that would never be used for service our units input properties. There is a also a special property called "temp". The contents of this property (on all services and units and environment itself) if wiped after each deployment.

Function context

Function context is an equivalent to Python stack frame. This allows functions to have local variables that are visible only to the function XML body.

Function context may be considered as a key-value storage. If a key does not exist in current context then parent context is searched for the key.

Data may be published to function context using ordinary `<set/>` function by using path in a form of "#key": `<set path="#myKey">value</set>` sets "myKey" function context value to "value". This value may be later accessed in inner block using select: `<select path="#myKey"/>`

Values is function context may be not just scalars but a whole objects: `<set path="#myKey"><select/><set/>` sets myKey local variable to an object pointed by current cursor position. Because this is a reference to a model part rather than its copy if ones edits its content the changes will be also present in original Model. To edit content of local variable (eg. value in function context) "target" argument of `<set/>` function is used: `<set path="property" target="myKey">value</set>` . Because target always denotes a key in function context there is no need for # sign.

Function context values may also be the source for the `<select/>` and other selection functions: `<select path="property" source="myKey"/>`

Workflow rules

Workflow consists of rules. Rule is a function with the following parameters:

- **match** - JSONPath expression to be executed relative to current cursor position
- **desc** - optional human-readable free-form rule description
- **id** - optional rule ID (auto-generated if not provided)

for example

```
<rule match="$.services[?(@.type == 'msSqlClusterServer' and @.domain)].units[
    <set path="domain">
         <select path="::domain"/>
    </set>
</rule>
```

The logic of rule is simple:

1. Execute given JSONPath expression
2. For each of matched objects make current cursor position point to it and then execute function XML body
3. If JSONPath hasn't matched any object execute `<empty>...</empty>` block if present

Rules can be nested. In this case nested rule's JSONPath expression would be executed relative to parent's rule current cursor position. For outermost rules current cursor position is the Object Model itself.

Rules are grouped into workflows:

```
<workflow>
    <rule id="rule1" match="...">
        …
    </rule>

    <rule id="rule2" match="...">
        …
```

```
    </rule>
</workflow>
```

Workflow which is happen to be a XML DSL function and also a root element of workflow XML files. Workflow function executes rules one by one. If one of the rules has modified some part of Object Model then all rules executed once again. This repeats until there are no more actions that can be performed by workflow. At this point all pending actions (e.g. all the invocations of update-cf-stack, send-command etc.) got executed and the workflow loop is repeated. When there no more actions that can be performed by workflow and also no pending commands then we are done and Object Model with all modifications made by workflows (except for "temp" attributes) returned back to Murano API service.

Workflow actions

There are several actions (functions) that can be invoked by rules to do the actual deployment. When action is invoked it is not executed immediately but enqueued and executed later when there are no more actions that can be performed by workflow.

The following actions are available for workflow rules:

- **update-cf-stack** - updates Heat stack by substituting values into Heat template and merging it into Heat stack definition. It has the following parameters:
  - **template** - Heat template filename without extension
  - **error** - function context variable to be populated with command error info in case of command failure
  - **mappings** - dictionary to be used for values substitution into template. All values in JSON template file in the form of "$myKey" are replaced with a value under key "myKey" in this parameter
  - **arguments** - optional dictionary of Heat template arguments ("Parameters" section of Heat templates)

  update-cf-stack function also searches for 2 predefined tags in its body:
  - <**success**> - a block to be executed after successfull stack update
  - <**failure**> - block that would be executed in case of function failure

  Templates are located in data/cf directory
- **send-command** - sends an execution plan to Murano Agent on specific VM. It has the following parameters:
  - **template** - execution plan template filename without extension
  - **error** - function context variable to be populated with command error info in case of command failure
  - **service** - ID of a service that target units belongs to
  - **unit** - ID of target unit (VM)
  - **mappings** - dictionary to be used for values substitution into template. All values in JSON template file in the form of "$myKey" are replaced with a value under key "myKey" in this parameter

  send-command function also searches for 2 predefined tags in its body:
  - <**success**> - a block to be executed after successfull stack update
  - <**failure**> - block that would be executed in case of function failure

Templates are located in data/agent directory

- **report** - sends status report back to REST API service. It has the following parameters:
  - **entity** - entity type ("unit", "service", "environment")
  - **level** - log level
  - **id** - ID of unit/service/environment
  - **text** - reported status text

Workflow control

There are several functions that affect workflow execution. `<stop/>` stops execution of workflows causing Conductor returning current result back to REST API service and stop deployment activities.

`<mute/>` excludes current object from being processed by the current rule during next workflow rounds. Mutes table tracks items using pairs or rule id (that is rule function argument, auto-generated if not provided) and object ID ("id" attribute of current object. If object has no such attribute it cannot be muted). It is possible to explicitly specify rule id via "rule" argument and object id via "id" argument. Missing parameters are auto-guest from current context - current cule ID and current object ID.

Note that objects are automatically muted for every object that matched by the rule but the mutes get reseted after each workflow loop round. `<mute/>` places a permanent mute on the object

Mutes can be removed by `<unmute/>` function which has exactly the same arguments as `<mute/>` command but is only usable with explicit specification of rule and id because otherwise if the current object is already muted the rule body would not be executed for the object and thus `<unmute/>` would not be executed either.

When referencing id of a nested rule IDs of all rule chain must be joined by dot. E.g.

```
<rule id="id1">
    <rule id="id2">
        <mute rule="id1.id2"/>
    </rule>
</rule>
```

Execution Plans

Execution plans are a description of what should be executed on Murano Agent at VM to perform some deployment step.

Execution plans is a JSON document that has 3 keys:

- **Commands** array - list of functions to be executed. Each function has a "Name" property and "Arguments" dictionary which maps function argument names to parameter values.
- **Scripts** - list of PowerShell script file names to be included into execution plans. The scripts contain function implementations that can be referenced in Command array.Script files need to be located in data/templates/agent/scripts directory.
- **RebootOnCompletion** - 0 = do not reboot, 1 = reboot only upon successful plan execution, 2 = reboot always. Murano Agent send execution result after system reboot.

As for other Murano JSON templates keys and values starting with $ sign will be replaced with a values provided in Workflow.

Example of execution plan:

```
{
    "Scripts":
    [
        "ImportCoreFunctions.ps1",
        "DeployWebApp.ps1"
    ],
    "Commands":
    [
        {
            "Name": "Deploy-WebAppFromGit",
            "Arguments":
            {
                "URL": "$repository"
            }
        }
    ],
    "RebootOnCompletion": 0
}
```

Result of execution plan has the following format:

```
{
    "IsException": false,
    "Result":
    [
        {
            "IsException": false,
            "Result": [ "result value" ]
        }
    ]
}
```

There are result entry for each source command. Each result can have many values - all the outputs of PowerShell function. If IsException is set to true then Result is an array in form of `["Exception type", "Error message"]`. IsException at command level means exception during function invocation while root IsException means that execution plan cannot be even started (corrupted data, PowerShell engine failure etc.)

Chapter 7. Known Issues

Actual bug state can be found in Murano launchpad page. [https://bugs.launchpad.net/murano]

- Due to current Heat limitation services that involve load-balancer creation (farms) can be deployed only by tenant administrators.
- When Heat creates different clients for Nova, Cinder and others it doesn't pass SSL-related options to clients' constructor. If Nova is configured to have SSL endpoints and self-signed certificates Heat will fail to create instances because there is no way to disable server certificate validation as there is no "insecure" flag passed etc.
- Farm services can't be deployed without KeyPair. If KeyPair is not set load balancer won't be created and these messages will show up in logs:

```
2013-08-06 09:10:07 - Unable to deploy instance ipkrmhk0vzq4b6 (asp-farm_instance
2013-08-06 09:10:07 - Unable to create a Server Farm load balancer on unit ipkrmh
```

  And deploy will hang up.

- MS SQL Cluster service relies on Quantum for 'virtual-ip' (address-pair) functionality, and to work on Nova Network or Quantum configured as "Mixed Flat and Private Network", anti-spoofing rules on compute nodes should be disabled.

  Internet Information Services Web Farm & ASP.NET Application Web Farm services are based on Heat, particularly on resource called AWS::ElasticLoadBalancing::LoadBalancer, that currently does not support specification of any network related parameters. Without support for network configuration specification LoadBalancer does not work in Murano over OpenStack deployments with Quantum configured as "Per-tenant Routers with Private Networks".

Chapter 8. Release History

Table 8.1. Murano Releases

Release	Tag	Date
Release-0.1	0.1	2013-05-30
Release-0.2	0.2	2013-09-05
Release-0.2.11	0.2.11	2013-10-01
Release-0.2.12	0.2.12	2013-11-18
Release-0.3	0.3	2013-11-25
Release-0.4	0.4	2013-12-20
Release-0.4.1	0.4.1	2014-02-05

Chapter 9. How To Participate

If you would like to ask some questions or make proposals, feel free to reach us on #murano irc channel at FreeNode. Typically somebody from our team will be online at irc from 6:00 to 20:00 UTC. You can also contact Murano community directly by openstack-dev@lists.openstack.org [mailto:openstack-dev@lists.openstack.org]

We’re going to hold public weekly meetings on Tuesdays at 17:00 UTC on #openstack-meeting-alt irc channel.

If you want to contribute either to docs or to code, simply send us change request via review.openstack.org [http://review.openstack.org] (gerrit). You can file bugs and register blueprints at Murano launchpad page [https://launchpad.net/murano].