Murano Developers Guide

Murano Developers Guide

v0.2

Publication date 2013-09-09

Abstract

This document is intended for individuals who wish to use our product or intend to contribute.

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 is intended to get opportunity for non-experienced users to deploy reliable Windows-based environments with 1-Click. Key goal is to provide UI and API which allows to deploy and operate Windows environments on the "Windows Services" abstraction level. The Service should be able to orchestrate complex circular dependent cases in order to setup complete Windows environments with many dependant services.

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

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

All these packages will be installed in murano-installation scripts. In addition to these packages some repositories are required. Please follow the instructions in the appendix to admin guide to prepare your environment for murano installation.

Installing with virtual environment

For a local development, all Murano components can be installed in a virtual environment.

- Install virtualenv package if you don't have one:

```
sudo pip install virtualenv
```

- Check out git repository with murano component:
  - ```
    git clone https://github.com/stackforge/murano-api
    ```
  - ```
    git clone https://github.com/stackforge/murano-conductor
    ```
  - ```
    git clone https://github.com/stackforge/murano-dashboard
    ```
- Make sure that required system packages are installed. Check list from prerequisites page
- Execute a script in the `tools` directory to create virtual environment automatically:

```
$ python murano-api/tools/install_venv.py
$ python murano-conductor/tools/install_venv.py
$ python murano-dashboard/tools/install_venv.py
```

- For **Murano Dashboard** additional installation need to be done:
  - Openstack dashboard (horizon).:

    ```
    ./tools/with_venv.sh pip install https://github.com/openstack/horizon/archive/g
    ```
  - Install openstack-dashboard dependency:
    - *Ubuntu*

      ```
      # apt-get install nodejs
      ```
    - *CentOS*

      ```
      # yum install nodejs
      ```
  - Customized Djblets package to support datagrid element (installation by pip is not supported):

    ```
    ./tools/with_venv.sh easy_install https://github.com/tsufiev/djblets/archive/ma
    ```
- Config files for the development infrastructure of murano-api and murano-conductor can be found at `etc` under folder with component repository. In murano-api and murano-conductor config file (located
    ```

under ./etc directory) just point out IP address where your RabbitMQ is running. All other possible configuration described in the Murano Admin Guide. To configure Murano Dashboard copy

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

and set in just copied file the the actual IP address of the OpenStack end-point. If you haven't register murano-api service in the keystone catalog you can set MURANO_API_URL in the same settings file. Note that local murano-api service will be using by default.

- Run Murano API:

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

- Run Murano Conductor:

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

- *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
```

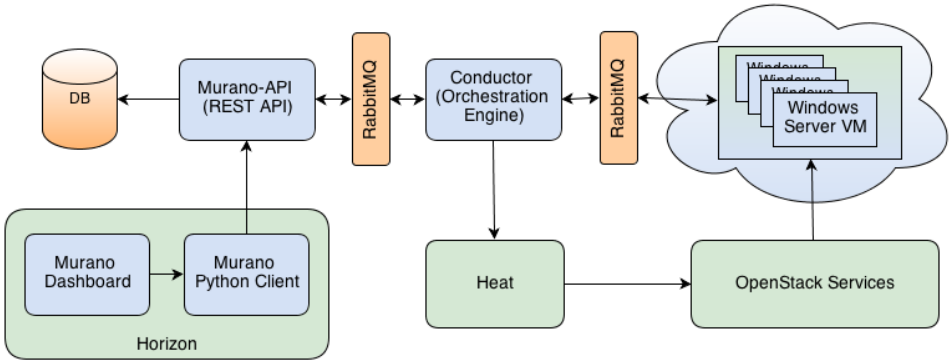
Chapter 3. Architecture

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 for Windows Environments;
- Glance - stores Windows Server VM images, with each image containing an installed OS and a set of 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 Windows 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.

Releases

- *Release-0.1* has 0.1 tag in all Murano repositories. Released 2013-05-30.
- *Release-0.2* has 0.2 tag in all Murano repositories. Released 2013-09-05.

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.

Windows on OpenStack

Windows works on KVM pretty smoothly, and with the RedHat-created open-source VirtIO drivers for Windows, it’s possible to work efficiently with KVM exposed devices.

In OpenStack’s Grizzly release, Microsoft’s hypervisor Hyper-V will be supported. The Hyper-V virtual switch will be also supported as a Quantum plug-in. From the performance viewpoint, Hyper-V Server 2012 compares very favorably with bare metal, processing just over 6% fewer transactions per second compared to the same workload running on a similarly configured physical server.

Also, unlike the current OpenStack, Hyper-V also natively supports Windows Clusters.

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

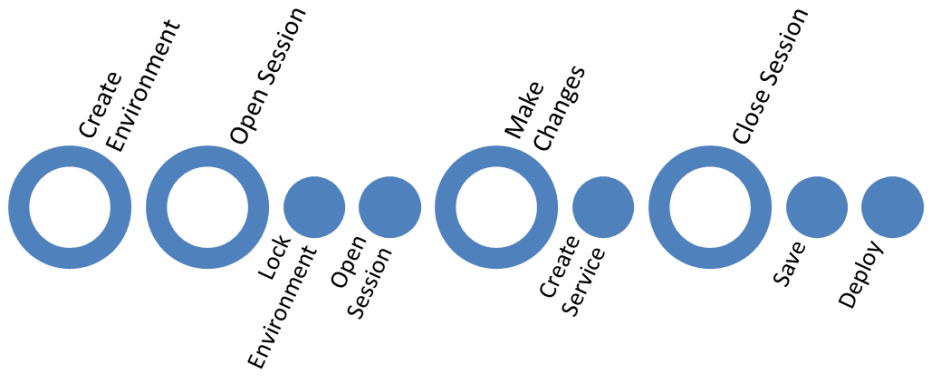
All POST and PUT requests by convention should return the created object (in the case of POST, with a generated ID) as if it was requested by GET.

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
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.

Request Object Specs

Table 4.37. Active Directory Object

Attribute	Type	Description
id	string	Unique ID
name	string	Domain 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	Domain 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.

Request Object Specs

Table 4.41. Web Server Object

Attribute	Type	Description
id	string	Unique ID
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.

Request Object Specs

Table 4.45. ASP.NET Application Object

Attribute	Type	Description
id	string	Unique ID
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.

Request Object Specs

Table 4.49. Web Server Farm Object

Attribute	Type	Description
id	string	Unique ID
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.

Request Object Specs

Table 4.53. ASP.NET Application Farm Object

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name

Attribute	Type	Description
uri	string	URI of the Service
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.

Request Object Specs

Table 4.57. MS SQL Server Object

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name
mixedModeAuth	bool	Use LDAP to access SQL Server
saPassword	string	SQL Server admin password

Attribute	Type	Description
domain	string	Domain name. This attribute may be empty/null/omitted if machine is not a domain member
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
-	-	-	-

Chapter 5. 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 (http://goessner.net/articles/JsonPath/):

```
<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 6. 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_instanc
2013-08-06 09:10:07 - Unable to create a Server Farm load balancer on unit ipkrm
```

  And deploy will hang up.

Chapter 7. 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 Mondays at 15:00 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].