
Murano

Release 2.0.0.0b3.dev156

February 26, 2016

1	Introduction to Murano	3
1.1	Key features	3
1.2	Target Users	4
1.3	Architecture	5
1.4	Use cases	6
2	Using Murano	7
2.1	Quickstart	7
2.2	Managing environments	11
2.3	Managing applications	14
2.4	Log into murano-spawned instance	36
2.5	Deploying environments using CLI	37
3	Developing Applications	41
3.1	Step-by-Step	41
3.2	Execution plan template	51
3.3	HOT packages	53
3.4	MuranoPL Reference	56
3.5	Murano actions	70
3.6	Murano packages	71
3.7	Migrating applications between releases	79
3.8	Application unit tests	85
3.9	Examples	89
3.10	Use-cases	90
3.11	FAQ	93
4	Miscellaneous	95
4.1	Murano Installation Guide	95
4.2	Murano workflow	108
4.3	Murano Policy Enforcement	110
4.4	Building Murano Image	118
4.5	Murano automated tests description	122
4.6	Murano client	125
4.7	Contributing to Murano	128
4.8	Development Guidelines	129
4.9	Murano TroubleShooting and Debug Tips	129
4.10	Murano API v1 specification	131

Murano is an open source OpenStack project that combines an application catalog with versatile tooling to simplify and accelerate packaging and deployment. It can be used with almost any application and service in OpenStack.

Murano project consists of several source code repositories:

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

This documentation guides application developers through the process of composing an application package to get it ready for uploading to Murano.

Besides the deployment rules and requirements, it contains information on how to manage images, categories, and repositories using the murano client that will surely be helpful for cloud administrators.

It also explains to end users how they can use the catalog directly from the dashboard. These include guidance on how to manage applications and environments.

And it provides information on how to contribute to the project.

Note: *Deploying Murano* and *Contributing* guides are under development at the moment. The most recently updated information is published as the *BETA version of the Murano documentation*.

Introduction to Murano

1.1 Key features

Murano has a number of features designed to interact with the application catalog, for instance managing what's in the catalog, and determining how apps in the catalog are deployed.

1.1.1 Application catalog

1. Easy browsing:

- Icons display applications for point-and-click and drag-and-drop selection and deployment.
- Each application provides configuration information required for deploying it to a cloud.
- An application topology of your environment is available in a separate tab, and shows the number of instances spawned by each application.
- The presence of the *Quick Deploy* button on the applications page saves the time.

2. Quick filtering by:

- Tags and words included in application name and description.
- Recent activity.
- Predefined category.

3. Dependency tracking:

- Automatic detection of dependent applications that minimizes the possibility of an application deployment with incorrect configuration.
- No underlying IaaS configuration knowledge is required.

1.1.2 Application catalog management

1. Easy application uploading using UI or CLI from:

- Local zip file.
- URL.
- Package name, using an application repository.

2. Managing applications include:

- Application organization in categories or transfer between them.
 - Application name, description and tags update.
 - Predefined application categories list setting.
3. Deployment tracking includes the availability of:
- Logs for deployments via UI.
 - Deployment modification history to track the recent changes.

1.1.3 Application lifecycle management

1. Simplified configuration and integration:
 - It is up to an application developer to decide what their application will be able to do.
 - Dependencies between applications are easily configured.
 - New applications can be connected with already existing ones.
 - Well specified application actions are available.
2. HA-mode and auto-scaling:
 - Application authors can set up any available monitoring system to track application events and call corresponding actions, such as failover, starting additional instances, and others.
3. Isolation:
 - Applications in the same environments can easily interact with each other, though applications between different tenants are isolated.

1.2 Target Users

Cloud end users want to simply use applications as opposed to installing and managing them. Cloud administrators, in turn, would like to offer a well tested set of on demand self-service applications to dramatically reduce their support burden.

Murano solves the problems of both constituents. It enables cloud administrators to publish cloud-ready applications in an online catalog. Cloud end users can use the catalog to deploy these on demand applications, reliably and consistently, with a button click.

1.2.1 Cloud administrators

For cloud administrators Murano provides UI and API to easily compose, deploy, run applications, and manage their lifecycle.

Designed to be operating system independent, it can handle apps on all manner of the environments in the cloud, either Windows or Linux/Unix-based operating systems.

It can be used to pre-configure and deploy anything that can run in the cloud, from low-level networking services to end-user applications. By automating these ongoing cloud application management activities, Murano speeds up the deployment, even for complex distributed applications, without sacrificing simplicity of use.

1.2.2 Cloud end users

Murano catalog lets cloud end users choose from the available applications and services, and compose reliable distributed environments with an intuitive UI. Even users unfamiliar with cloud environments can easily deploy cloud-aware applications.

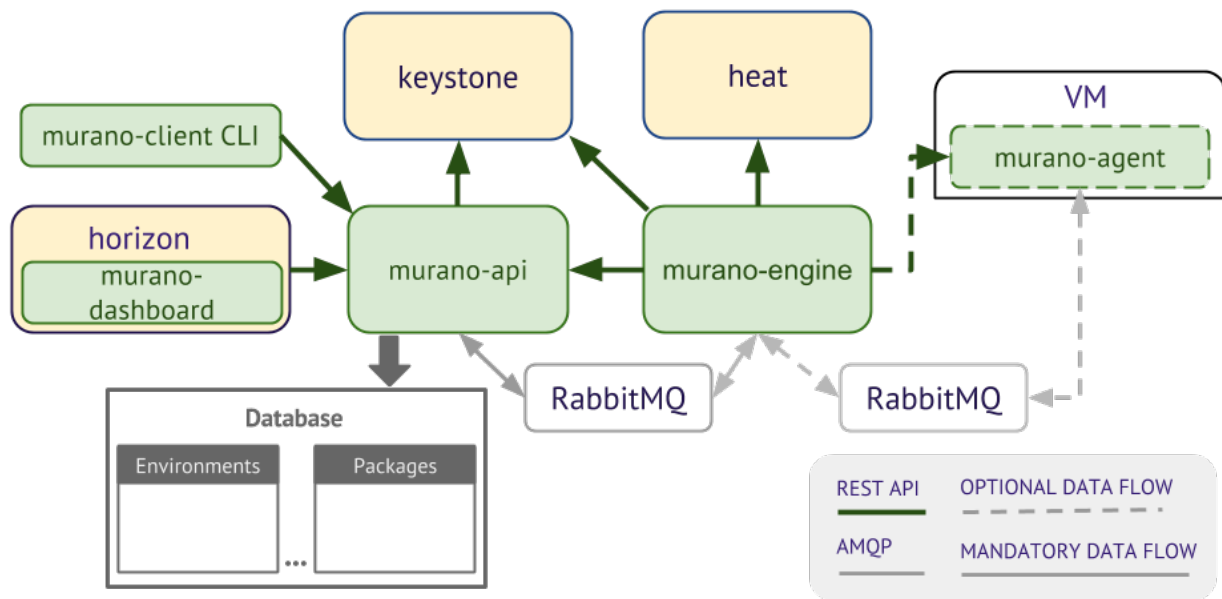
Murano masks cloud-infrastructure specifics from end users, letting them reliably compose and deploy applications in the cloud for the widest range of workloads and use cases without touching IaaS internals.

1.3 Architecture

Murano is composed of the following major components:

- murano command-line client
- murano-dashboard
- murano-api
- murano-engine
- murano-agent

They interact with each other as illustrated in the following diagram:



All remote operations on users' servers, such as software installation and configuration, are carried out through an AMQP queue to the murano-agent. Such communication can easily be configured on a separate instance of AMQP to ensure that the infrastructure and servers are isolated.

Besides, Murano uses other OpenStack services to prevent the reimplementing of the existing functionality. Murano interacts with these services using their REST API through their python clients.

The external services used by Murano are:

- the **Orchestration service** (Heat) to orchestrate infrastructural resources such as servers, volumes, and networks. Murano dynamically creates heat templates based on application definitions.

- the **Identity service** (Keystone) to make murano API available to all OpenStack users.

1.4 Use cases

IT-as-a-Service

An *IT organization* manages applications and controls the applications availability to different OpenStack cloud users in a simple and timesaving manner.

A *cloud end user* can easily find and deploy any available application from the catalog.

Self-service portal

An *application developer* and *quality assurance engineer* reduces efforts on testing an application for compatibility with other applications, databases, platforms, and other components it depends on, by configuring compound combinations of applications dynamically and deploying environments that satisfy all requirements within minutes.

Glue layer use case

A *cloud end user* is able to link an ever growing number of technologies to any application in an OpenStack cloud with a minimum cost due to the powerful Murano architecture.

Currently, Murano applications have been integrated with the following technologies: Docker, Legacy apps VMs or bare metal, apps outside of OpenStack, and others.

The following technologies are to become available in the future: Cloudify and TOSCA, Apache Brooklyn, and APS.

Using Murano

2.1 Quickstart

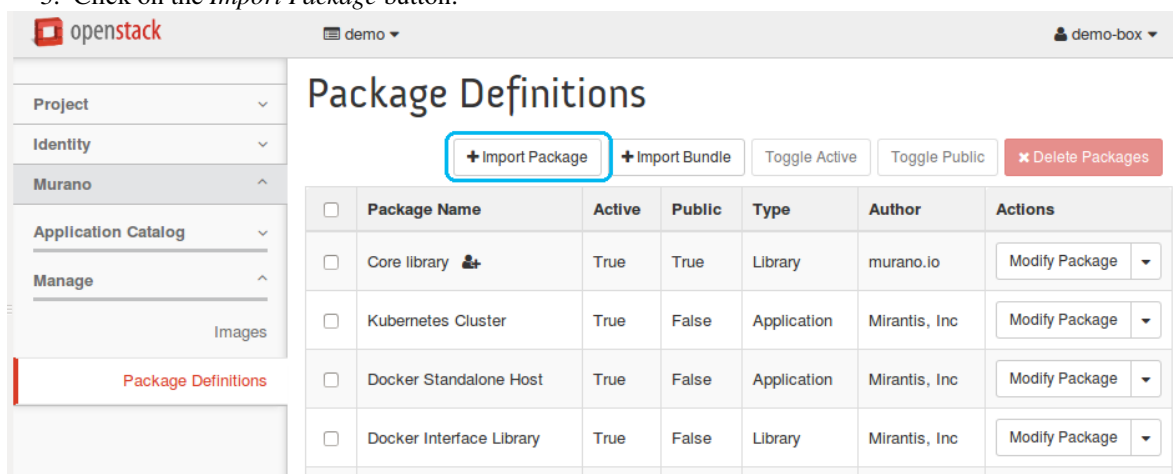
This is a brief walkthrough to quickly get you familiar with the basic operations you can perform when using the Application catalog directly from the dashboard.

For the detailed instructions on how to *manage your environments* and *applications*, please proceed with dedicated sections.

2.1.1 Upload an application

To upload an application to the catalog:

1. Log in to the OpenStack dashboard.
2. Navigate to *Murano > Manage > Packages*.
3. Click on the *Import Package* button:



The screenshot shows the OpenStack dashboard interface. On the left is a sidebar with navigation links: Project, Identity, Murano, Application Catalog, and Manage. The main content area is titled 'Package Definitions'. At the top of this area are several buttons: '+ Import Package' (highlighted with a red box), '+ Import Bundle', 'Toggle Active', 'Toggle Public', and a red 'Delete Packages' button. Below these buttons is a table listing installed packages.

<input type="checkbox"/>	Package Name	Active	Public	Type	Author	Actions
<input type="checkbox"/>	Core library	True	True	Library	murano.io	Modify Package
<input type="checkbox"/>	Kubernetes Cluster	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker Standalone Host	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker Interface Library	True	False	Library	Mirantis, Inc	Modify Package

4. In the *Import Package* dialog:

- Select URL from the Package Source drop-down list;
- Specify the URL in the *Package URL* field. Lets upload the Apache HTTP Server package using <http://storage.apps.openstack.org/apps/io.murano.apps.apache.ApacheHttpServer.zip>;
- Click *Next* to continue:

Import Package ✕

Package Source

URL

Package URL *

http://storage.apps.openstack.org/apps/io.murano.:

Description:

Package URL: HTTP/HTTPS URL of the package file.

Note: If the package depends upon other packages and/or requires specific glance images, those are going to be installed with it from murano repository.

Cancel

Next

5. View the package details in the new dialog, click *Next* to continue:

Import Package ✕

Name

Apache HTTP Server

Tags

HTTP, Server, WebServer, HTML, Apache

☐ Public

☒ Active

Description

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

Apache httpd has been the most popular web server on the Internet since

Description:

Name: Set up for identifying a package.

Tags: Used for identifying and filtering packages.

Public: Defines whether or not a package can be used by other tenants. (Applies to package dependencies)

Active: Allows to hide a package from the catalog. (Applies to package dependencies)

Description: Allows adding additional information about a package.

Cancel

Next

6. Select the *Application Servers* from the application category list, click *Create* to import the application package:

Import Package

Application Category

Application Servers
Key-Value Storage
SAP
Microsoft Services

Description:

Categories Select one or more categories for a package

Specifying a category helps to filter applications in the catalog

Cancel

Create

7. Now your application is available from *Murano > Application Catalog > Applications* page.


2.1.2 Deploy an application

To add an application to an environment's component list and deploy the environment:

1. Log in to the OpenStack dashboard.
2. Navigate to *Murano > Application Catalog > Applications*.
3. Click on the *Quick Deploy* button from the required application from the list. Lets deploy Apache HTTP Server, for example:

The screenshot shows the OpenStack Murano Applications page. The left sidebar contains navigation links: Project, Identity, Murano, Application Catalog, Environments, Applications (highlighted), and Manage. The main content area is titled "Applications" and shows "Recent Activity" with a list of applications. The first application is "Apache HTTP Ser..." with a description: "The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server". It has a "Details" link and two buttons: "Create Env" and "Quick Deploy". Below this, there are filters for "App Category" (set to "All") and "Environment" (set to "Create Environment"). There are also search filters. The second application is "Apache Tomcat" with a description: "Apache Tomcat is an open source software implementation of the Java Servlet and". It also has a "Details" link and "Create Env" and "Quick Deploy" buttons.

4. Check *Assign Floating IP* and click *Next* to proceed:



Configure Application: Apache HTTP Server

Application Name *

☐ Enable PHP

☒ Assign Floating IP

Apache HTTP Server

Apache License, Version 2.0


Application Name: Enter a desired name for the application. Just A-Z, a-z, 0-9, dash and underline are allowed

Enable PHP: Add php support to the Apache WebServer

➤ **Assign Floating IP:** Select to true to assign floating IP automatically

Next

5. Select the *Instance Image* from the drop-down list and click *Create*:



Configure Application: Apache HTTP Server

Instance flavor

Instance image *

Key Pair

Availability zone

Instance Naming Pattern ?

Apache HTTP Server

Specify some instance parameters on which the application would be created

Instance flavor: Select registered in Openstack flavor. Consider that application performance depends on this parameter.

➤ **Instance image:** Select valid image for the application. Image should already be prepared and registered in glance.

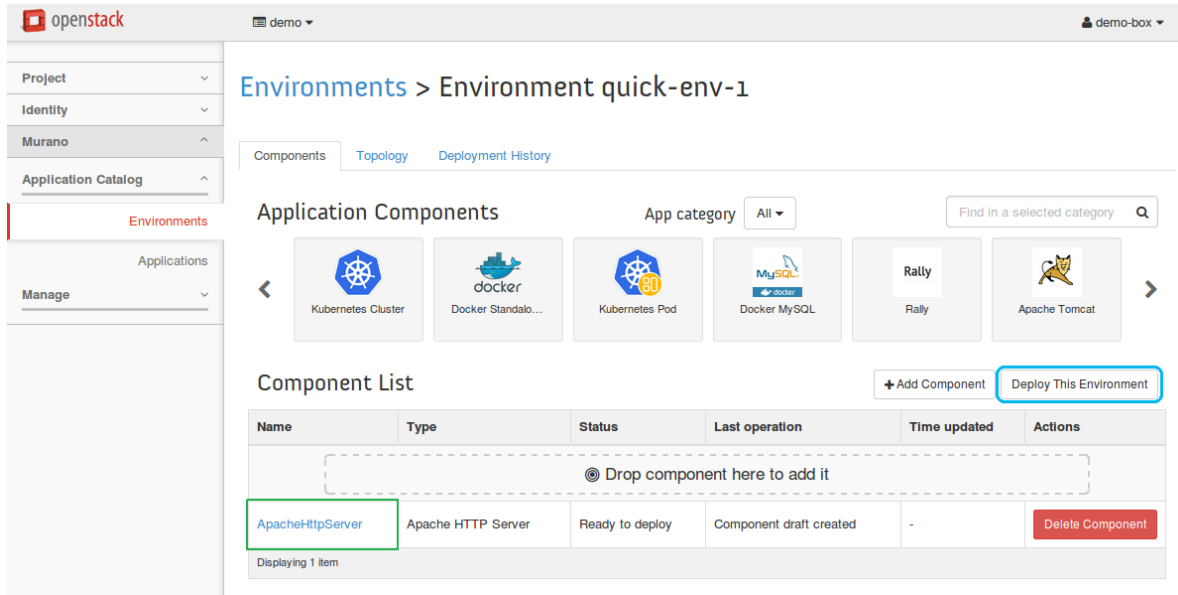
Key Pair: Select the Key Pair to control access to instances. You can login to instances using this KeyPair after the deployment of application.

Availability zone: Select availability zone where application would be installed.

Instance Naming Pattern: Specify a string, that will be used in instance hostname. Just A-Z, a-z, 0-9, dash and underline are allowed.

Back Create

6. Now the Apache HTTP Server application is successfully added to the newly created `quick-env-1` environment. Click the *Deploy This Environment* button to start the deployment:



It may take some time for the environment to deploy. Wait until the status is changed from *Deploying* to *Ready*.

7. Navigate to *Murano > Application Catalog > Environments* to view the details.

2.1.3 Delete an application

To delete an application that belongs to the environment:

1. Log in to the OpenStack dashboard.
2. Navigate to *Murano > Application Catalog > Environments*.
3. Click on the name of the environment to view its details, which include components, topology, and deployment history.
4. In the *Component List* section, click on the *Delete Component* button next to the application to be deleted. Confirm the deletion.

Note: If an application that you are deleting has already been deployed, you should redeploy it to apply the recent changes. If the environment has not been deployed with this component, the changes are applied immediately on receiving the confirmation.

Warning: Due to a known bug in Murano Kilo, resources allocated by a deleted application might not be reclaimed until the deletion of an environment. See [LP1417136](#) for the details.

2.2 Managing environments

An environment is a set of logically connected applications that are grouped together for an easy management. By default, each environment has a single network for all its applications, and the deployment of the environment is defined in a single heat stack. Applications in different environments are always independent from one another.

An environment is a single unit of deployment. This means that you deploy not an application but an environment that contains one or multiple applications.

Using OpenStack Dashboard you can easily perform such actions with an environment as creating, editing, reviewing, deploying, and others.

2.2.1 Create an environment

To create an environment, perform the following steps:

1. In OpenStack Dashboard, navigate to Murano > Application Catalog > Environments.
2. On the *Environments* page, click the *Add New* button.
3. In the *Environment Name* field, enter the name for the new environment.
4. From the *Environment Default Network* drop-down list, choose a specific network, if necessary, or leave the default *Create New* option to generate a new network.

5. Click the rightmost *Create* button. You will be redirected to the page with the environment components.

Alternatively, you can create an environment automatically using the *Quick Deploy* button below any application in the Application Catalog. For more information, see: [Quick deploy](#).

2.2.2 Edit an environment

You can edit the name of an environment. For this, perform the following steps:

1. In OpenStack Dashboard, navigate to Murano > Application Catalog > Environments.
2. Position your mouse pointer over the environment name and click the appeared pencil icon.
3. Edit the name of the environment.
4. Click the tick icon to apply the change.

2.2.3 Review an environment

This section provides a general overview of an environment, its structure, possible statuses, and actions. An environment groups applications together. An application that is added to an environment is called a component.

To see an environment status, navigate to *Murano > Application Catalog > Environments*. Environments may have one of the following statuses:

- **Ready to configure.** When the environment is new and contains no components.
- **Ready to deploy.** When the environment contains a component or multiple components and is ready for deployment.

- **Ready.** When the environment has been successfully deployed.
- **Deploying.** When the deploying is in progress.
- **Deploy FAILURE.** When the deployment finished with errors.
- **Deleting.** When deleting of an environment is in progress.
- **Delete FAILURE.** You can abandon the environment in this case.

Currently, the component status corresponds to the environment status.

To review an environment and its components, or reconfigure the environment, click the name of an environment or simply click the rightmost *Manage Components* button.

- On the *Components* tab you can:
 - Add or delete a component from an environment
 - Send an environment to deploy
 - Track a component status
 - Call murano actions of a particular application in a deployed environment:

The screenshot shows the OpenStack Murano web interface. The top navigation bar includes the OpenStack logo and a 'demo-box' dropdown. The left sidebar has a navigation menu with 'Environments' selected. The main content area is titled 'Environments > Environment quick-env-3'. Below the title are tabs for 'Components', 'Topology', 'Deployment History', and 'Latest Deployment Log'. The 'Components' tab is active, showing a grid of application components. Below the grid is a 'Component List' table. The table has columns: Name, Type, Status, Last operation, Time updated, and Actions. A 'KubernetesCluster' is listed with a status of 'Ready'. An 'Actions' dropdown menu is open for the 'KubernetesCluster', showing options like 'Delete Component', 'exportConfig', 'scaleGatewaysUp', 'scaleNodesDown', and 'scaleNodesUp'.

Name	Type	Status	Last operation	Time updated	Actions
KubernetesCluster	Kubernetes Cluster	Ready	Kubernetes cluster is up and running	Nov. 18, 2015, 11:12 a.m.	Delete Component exportConfig scaleGatewaysUp scaleNodesDown scaleNodesUp

For more information on murano actions, see: *Murano actions*.

- On the *Topology*, *Deployment History*, and *Latest Deployment Log* tabs of the environment page you can view the following:
 - The application topology of an environment. For more information, see: *Application topology*.
 - The log of a particular deployment. For more information, see: *Deployment history*.
 - The information on the latest deployment of an environment. For more information, see: *Latest deployment log*.

2.3 Managing applications

In murano, each application, as well as the form of application data entry, is defined by its package. The murano dashboard allows you to import and manage packages as well as search, filter, and add applications from catalog to environments.

This section provides detailed instructions on how to import application packages into murano and then add applications to an environment and deploy it. This section also shows you how to find component details, application topology, and deployment logs.

2.3.1 Import an application package

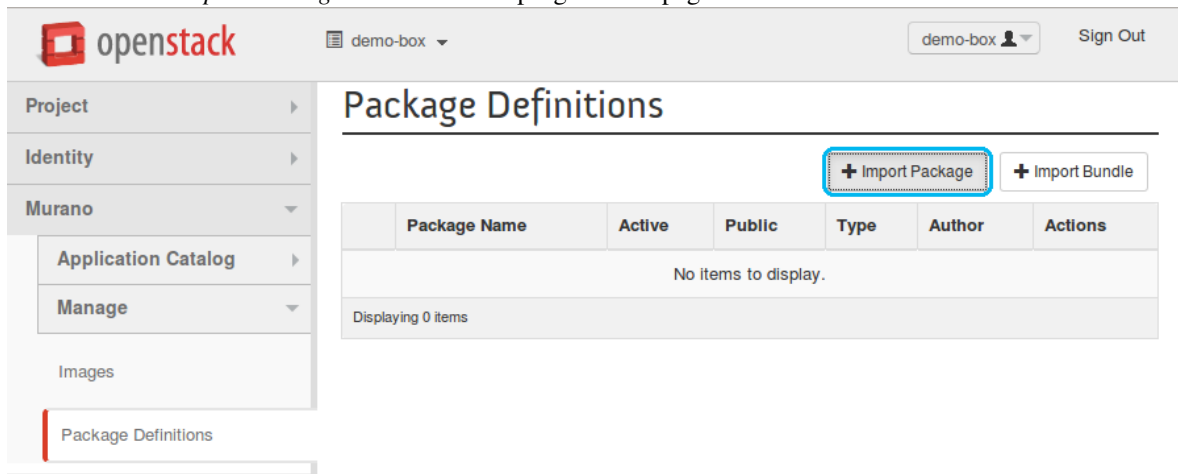
There are several ways of importing an application package into murano:

- *from a zip file*
- *from murano applications repository*
- *from bundles of applications*

From a zip file

Perform the following steps to import an application package from a .zip file:

1. In OpenStack dashboard, navigate to *Murano > Manage > Packages*.
2. Click the *Import Package* button on the top right of the page.



3. From the *Package source* drop-down list choose *File*, then click *Browse* to select a .zip file you want to import, and then click *Next*.

Import Package

Package Source

File

Application Package * ?

Browse...

No file selected.

Description:

Choose a Zip archive to upload into the catalog.

Packages should contain:

- * Manifest file
- * UI definition folder
- * Classes definition folder
- * Execution plans folder

Note: If the package depends upon other packages and/or requires specific glance images, those are going to be installed with it from murano repository.

Cancel

Next

4. At this step, the package is already uploaded. Choose a category from the *Application Category* menu. You can select multiple categories while holding down the `Ctrl` key. If necessary, verify and update the information about the package, then click the *Create* button.

Import Package ✕

Name

Application Category

Web
 Load Balancers
 Message Queue
Databases
 Key-Value Storage

Tags ?

☐ Public

☒ Active

Description

MySql is a relational database management system (RDBMS), and ships with no GUI tools to administer MySQL databases or manage data contained within the databases.

Description:

Name is a human-readable name of a package.

Categories are a predefined set of values used to filter the packages.

Tags are an arbitrary comma-separated values also used to filter the packages.

Public Defines whether or not a package is available for use by other tenants. (Applies to package dependencies)

Active Allows the status of a package to be changed. (Applies to package dependencies)

Description consists of several sentences about the package's purpose.

Note: Though specifying a category is optional, we recommend that you specify at least one. It helps to filter applications in the catalog.

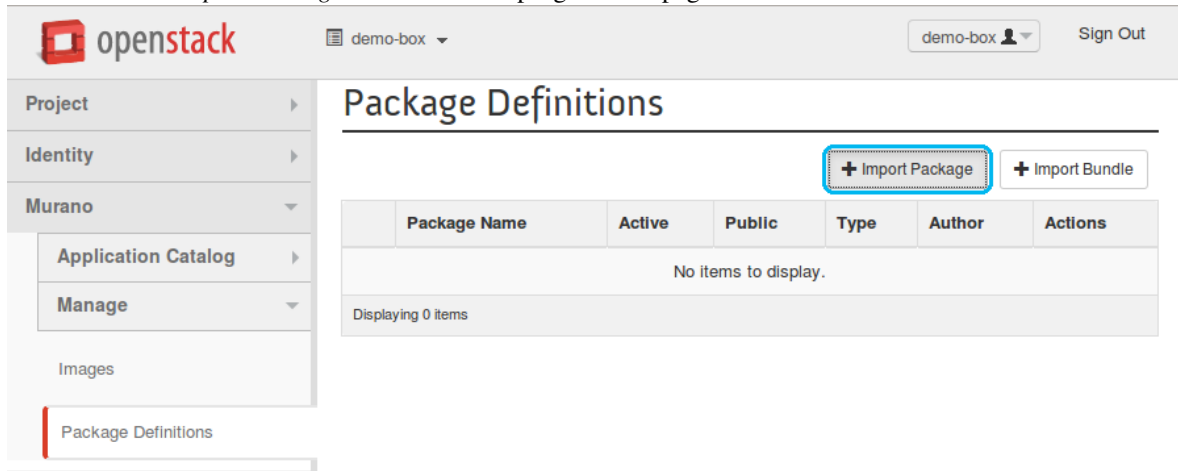
Green messages appear at the top right corner when the application is successfully uploaded. In case of a failure, you will see a red message with the problem description. For more information, please refer to the logs.

From a repository

Perform the following steps to import an application package from murano applications repository:

Note: To import an application package from a repository, you need to know the full name of the package. For the packages names, go to <http://apps.openstack.org/#tab=murano-apps> and click on the desired package to see its full name.

1. In dashboard, navigate to *Murano > Manage > Packages*.
2. Click the *Import Package* button on the top right of the page.



3. From the *Package source* drop-down list, choose *Repository*, enter the package name, and then click *Next*. Note that you may also specify the version of the package.

Import Package

Package Source

Repository

Package Name * ?

|

Package version

Optional

Description:

Package Name: Fully qualified package name.

Package Version: Version of the package (optional).

The package is going to be imported from <http://storage.apps.openstack.org/> repository.

Note: If the package depends upon other packages and/or requires specific glance images, those are going to be installed with it from murano repository.

Cancel

Next

4. At this step, the package is already uploaded. Choose a category from the *Application Category* menu. You can select multiple categories while holding down the **Ctrl** key. If necessary, verify and update the information about the package, then click the *Create* button.

Import Package ✕

Name

Application Category

Web
 Load Balancers
 Message Queue
Databases
 Key-Value Storage

Tags ?

☐ Public

☒ Active

Description

MySql is a relational database management system (RDBMS), and ships with no GUI tools to administer MySQL databases or manage data contained within the databases.

Description:

Name is a human-readable name of a package.

Categories are a predefined set of values used to filter the packages.

Tags are an arbitrary comma-separated values also used to filter the packages.

Public Defines whether or not a package is available for use by other tenants. (Applies to package dependencies)

Active Allows the status of a package to be changed. (Applies to package dependencies)

Description consists of several sentences about the package's purpose.

From a bundle of applications

Perform the following steps to import a bundle of applications:

Note: To import an application bundle from a repository, you need to know the full name of the package bundle. To find it out, go to <http://apps.openstack.org/#tab=murano-apps> and click on the desired bundle to see its full name.

1. In dashboard, navigate to *Murano > Manage > Packages*.
2. Click the *Import Bundle* button on the top right of the page.

- From the *Package Bundle Source* drop-down list, choose *Repository*, enter the bundle name, and then click *Create*.

Import Bundle

Package Bundle Source

Repository

Bundle Name *

Description:

Bundle Name: Bundle's full name.

The bundle is going to be installed from <http://storage.apps.openstack.org/> repository.

Note: You'll have to configure each package installed from this bundle separately.

If packages depend upon other packages and/or require specific glance images, those are going to be installed with them from murano repository.

Cancel

Create

2.3.2 Search for an application in the catalog

When you have imported many applications and want to quickly find a required one, you can filter them by category, tags and words that the application name or description contains:

In dashboard, navigate to *Murano > Application Catalog > Applications*.

The page is divided into two sections:

- Recent Activity** shows the most recently imported or deployed applications.
- The bottom section contains all the available applications sorted alphabetically.

To view all the applications of a specific category, select it from the *App Category* drop-down list:

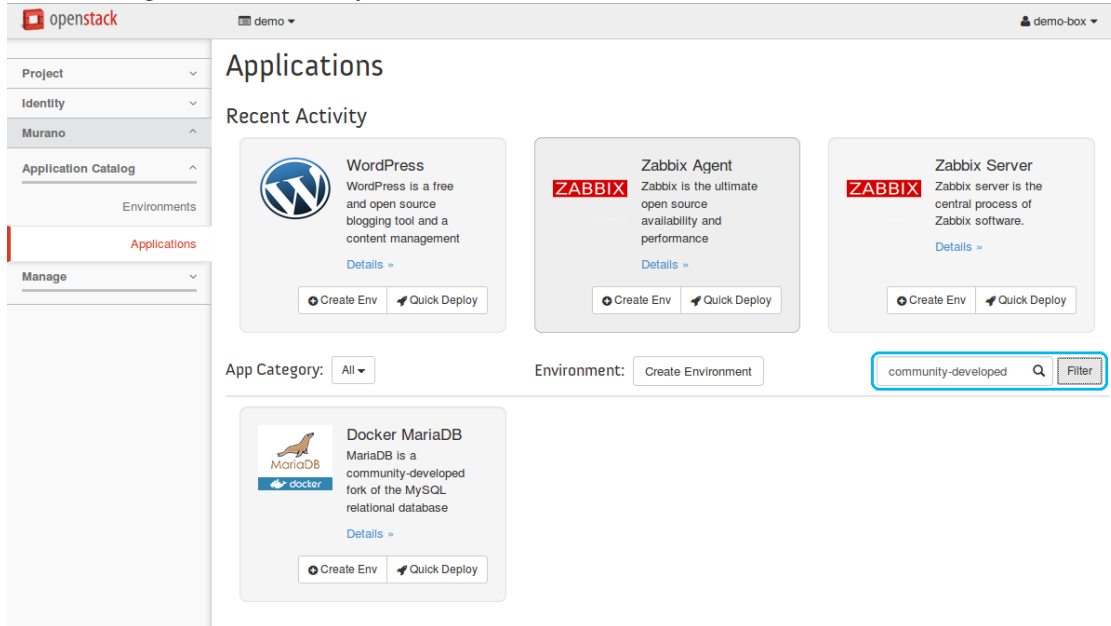
The screenshot shows the OpenStack Murano Applications page. The left sidebar contains navigation links: Project, Identity, Murano, Application Catalog, Environments, Applications (highlighted), and Manage. The main content area is titled "Applications" and "Recent Activity". It displays a grid of application cards, each with a logo, name, description, and buttons for "Create Env" and "Quick Deploy". The cards include WordPress, Zabbix Agent, Zabbix Server, Apache Tomcat, Docker MariaDB, Docker Nginx, Docker Redis, and MySQL. An "App Category:" dropdown menu is open, showing a list of categories: All, Application Servers, Key-Value Storage, SAP, Microsoft Services, Databases, Message Queue, Web, Big Data, and Load Balancers. The "Environment:" section has a "Create Environment" button and a "Filter" input field.

To filter applications by tags or words from the application name or description, use the rightmost filter:

This screenshot shows the same OpenStack Murano Applications page, but with the "rally" filter applied in the search bar. The "App Category:" dropdown is now set to "All". The "Environment:" section still has the "Create Environment" button and the "Filter" input field. The application grid now only displays the "Rally" application card, which is described as "Rally is an official OpenStack benchmarking and performance analysis".

Note: Tags can be specified during the import of an application package.

For example, there is an application that has the word *community-developed* in description. Let's find it with the filter. The following screenshot shows you the result.



2.3.3 Delete an application package

To delete an application package from the catalog, please perform the following steps:

1. In dashboard, navigate to *Murano > Manage > Package Definitions*.
2. Select a package or multiple packages you want to delete and click *Delete Packages*.

The screenshot shows the OpenStack Murano interface. On the left is a sidebar with navigation links: Project, Identity, Murano (selected), Application Catalog, Manage, Images, and Package Definitions. The main content area is titled 'Package Definitions' and contains a table of packages. Above the table are buttons for '+ Import Package', '+ Import Bundle', 'Toggle Active', 'Toggle Public', and a red 'Delete Packages' button. The table lists 13 packages, with 'PostgreSQL' and 'Kubernetes Cluster' selected. Each row has a 'Modify Package' button in the Actions column. At the bottom of the table, it says 'Displaying 13 items'.

<input type="checkbox"/>	Package Name	Active	Public	Type	Author	Actions
<input type="checkbox"/>	SQL Library	True	False	Library	Mirantis, Inc	Modify Package
<input type="checkbox"/>	MySQL	True	False	Application	Mirantis, Inc	Modify Package
<input checked="" type="checkbox"/>	PostgreSQL	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker Interface Library	True	False	Library	Mirantis, Inc	Modify Package
<input checked="" type="checkbox"/>	Kubernetes Cluster	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker Standalone Host	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Kubernetes Pod	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker Jenkins	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Zabbix Server	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Zabbix Agent	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	WordPress	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Apache HTTP Server	True	False	Application	Mirantis, Inc	Modify Package
<input type="checkbox"/>	Docker MariaDB	True	False	Application	Mirantis, Inc	Modify Package

Displaying 13 items

3. Confirm the deletion.

2.3.4 Add an application to environment

After uploading an application, the second step is to add it to an environment. You can do this:

- *from environment details page*
- *from applications catalog page*

From environment details page

1. In OpenStack dashboard, navigate to *Murano > Application catalog > Environments*.
2. Find the environment you want to manage and click *Manage Components*, or simply click on the environment's name.
3. Proceed with the *Drop Components here* field or the *Add Component* button.

Use of Drop Components here field

1. On the Environment Components page, drag and drop a desired application into the *Drop Components here* field under the *Application Components* section.

openstack demo-box demo-box

Environments > Environment Environment-3

Components Deployment History

Application Components App category All Find in a selected category

Apache HTTP S... Apache Tomcat Docker Jenkins Docker MariaDB Docker Standalo... Kubernetes Clus...

Drop Components here

Component List + Add Component

Name	Type	Status	Last operation	Time updated	Actions
No components					
Displaying 0 items					

2. Configure the application. Note that the settings may vary from app to app and are predefined by the application author. When done, click *Next*, then click *Create*.

Now the application appears in the *Component List* section on the Environment Components page. **Use of Add Component button**

1. On the Environment Components page, click *Add Component*.

openstack demo-box demo-box

Environments > Environment Environment-3

Components Topology Deployment History

Application Components App category All Find in a selected category

Apache HTTP S... Apache Tomcat Docker Jenkins Docker MariaDB Docker Standalo... Kubernetes Clus...

Drop Components here

Component List + Add Component

Name	Type	Status	Last operation	Time updated	Actions
No components					
Displaying 0 items					

2. Find the application you want to add and click *Add to Env*.

3. Configure the application and click *Next*. Note that the settings may vary from app to app and are predefined by the application author.
4. To add more applications, check *Add more applications to the environment*, then click *Create* and repeat the steps above. Otherwise, just click *Create*.

Now the application appears in the *Component List* section on the Environment Components page.

From applications catalog page

1. In OpenStack dashboard, navigate to *Murano > Application catalog > Applications*.
2. On the Applications catalog page, use one of the following methods:


- **Quick deploy.** Automatically creates an environment, adds the selected application, and redirects you to the page with the environment components.
- **Add to Env.** Adds an application to an already existing environment.

Quick Deploy button

1. Find the application you want to add and click *Quick Deploy*. Let's add Apache Tomcat, for example.

The screenshot displays the OpenStack Murano Application Catalog interface. On the left, a sidebar contains navigation links: Project, Identity, Murano, Application Catalog, Environments, Applications (highlighted in red), and Manage. The main area is titled 'Applications' and shows 'Recent Activity' with three application cards: WordPress, Zabbix Server, and Apache HTTP Server. Below this, there is a grid of application cards. The 'Apache Tomcat' card is highlighted with a blue border. Each card includes an icon, name, description, 'Details' link, and 'Create Env' and 'Quick Deploy' buttons. At the bottom, there is a 'Next Page' button.

2. Configure the application. Note that the settings may vary from app to app and are predefined by the application author. When done, click *Next*, then click *Create*. In the example below we assign a floating IP address.



Configure Application: Apache Tomcat

Application Name *

Tomcat

☒ Assign Floating IP

Apache Tomcat


Apache License, Version 2.0

Application Name: Enter a desired name for the application. Just A-Z, a-z, 0-9, dash and underline are allowed

Assign Floating IP: Select to true to assign floating IP automatically

Next

Now the Apache Tomcat application is successfully added to an automatically created `quick-env-1` environment.



demo-box

Project

Identity

Murano

Application Catalog

Environments

Applications

Manage

Environments > Environment quick-env-1

Components

Topology

Deployment History

Application Components

App category All

Find in a selected category

Apache

Apache Tomcat

Docker Jenkins

Docker MariaDB

Docker Standalone

Kubernetes Cluster

Drop Components here

Component List

+ Add Component

Deploy This Environment

Name	Type	Status	Last operation	Time updated	Actions
Tomcat	Apache Tomcat	Ready to deploy	Component draft created	-	Delete Component

Displaying 1 item

Success: The 'Apache Tomcat' application successfully added to environment.

Add to Env button

1. From the *Environment* drop-down list, select the required environment.

The screenshot shows the OpenStack Murano web interface. On the left is a sidebar with navigation options: Project, Identity, Murano, Application Catalog, Environments, Applications (highlighted in red), and Manage. The main area is titled 'Applications' and 'Recent Activity'. It features a grid of application cards. Each card represents an application like WordPress, MySQL, Zabbix Agent, Apache HTTP Server, Docker Jenkins, Docker MariaDB, Docker Standalone, and Kubernetes Clusters. Each card has a logo, a brief description, a 'Details' link, and two buttons: 'Add to Env' and 'Quick Deploy'. Above the grid, there are filters for 'App Category' (set to 'All') and 'Environment' (set to 'Environment-3', with a dropdown menu open showing 'quick-env-1' and 'Environment-2'). There is also a search filter input. At the bottom left of the grid is a 'Next Page' button.

- Find the application you want to add and click *Add to Env*. Let's add Apache Tomcat, for example.

3. Configure the application and click *Next*. Note that the settings may vary from app to app and are predefined by the application author. In the example below we assign a floating IP address.

4. To add more applications, check *Add more applications to the environment*, then click *Create* and repeat the steps above. Otherwise, just click *Create*.

Configure Application: Apache Tomcat

☐ Add more applications to the environment

If checked, you will be returned to the Application Catalog page. If not - to the Environment page, where you can deploy the application.

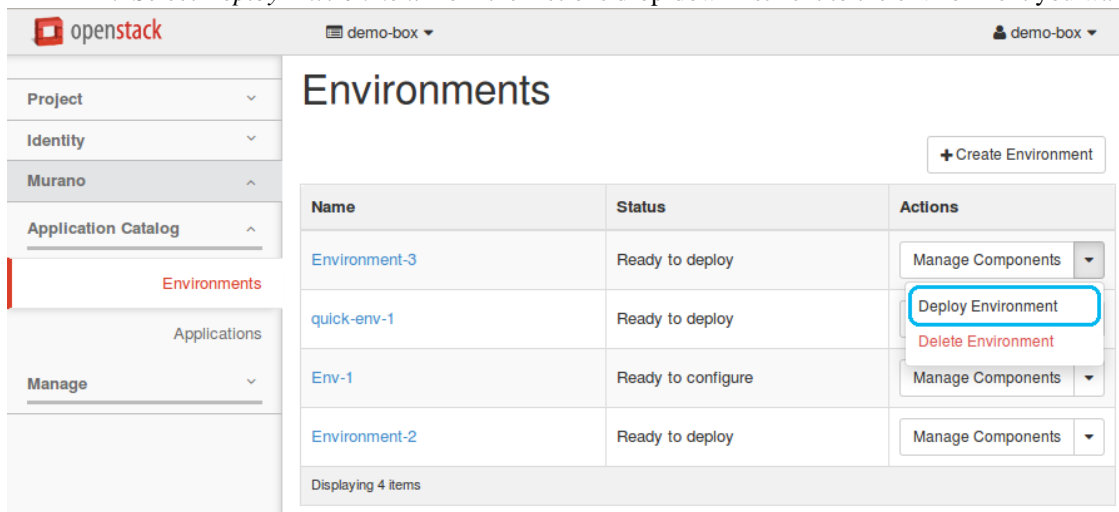
Back

Create

2.3.5 Deploy an environment

Make sure to add necessary applications to your environment, then deploy it following one of the options below:

- Deploy an environment from the Environments page
 1. In OpenStack dashboard, navigate to *Murano > Application Catalog > Environments*.
 2. Select *Deploy Environment* from the Actions drop-down list next to the environment you want to deploy.



The screenshot shows the OpenStack Murano dashboard. On the left is a sidebar with navigation links: Project, Identity, Murano, Application Catalog, Environments (selected), Applications, and Manage. The main content area is titled 'Environments' and features a '+ Create Environment' button. Below this is a table with the following data:

Name	Status	Actions
Environment-3	Ready to deploy	Manage Components
quick-env-1	Ready to deploy	Deploy Environment (highlighted), Delete Environment
Env-1	Ready to configure	Manage Components
Environment-2	Ready to deploy	Manage Components

At the bottom of the table, it says 'Displaying 4 items'.

It may take some time for the environment to deploy. Wait for the status to change from *Deploying* to *Ready*. You cannot add applications to your environment during deployment.

- Deploy an environment from the Environment Components page
 1. In OpenStack dashboard, navigate to *Murano > Application Catalog > Environments*.
 2. Click the name of the environment you want to deploy.

Name	Status	Actions
Env-1	Ready to deploy	Manage Components
Environment-2	Ready to deploy	Manage Components
Environment-3	Ready to deploy	Manage Components
quick-env-1	Ready to deploy	Manage Components

Displaying 4 items

3. On the Environment Components page, click *Deploy This Environment* to start the deployment.

Environments > Environment Env-1

Components | Topology | Deployment History

Application Components

App category: All

Find in a selected category

Drop Components here

Component List

Name	Type	Status	Last operation	Time updated	Actions
Tomcat	Apache Tomcat	Ready to deploy	Component draft created	-	Delete Component

Displaying 1 item

It may take some time for the environment to deploy. You cannot add applications to your environment during deployment. Wait for the status to change from *Deploying* to *Ready*. You can check the status either on the Environments page or on the Environment Components page.

Browse component details

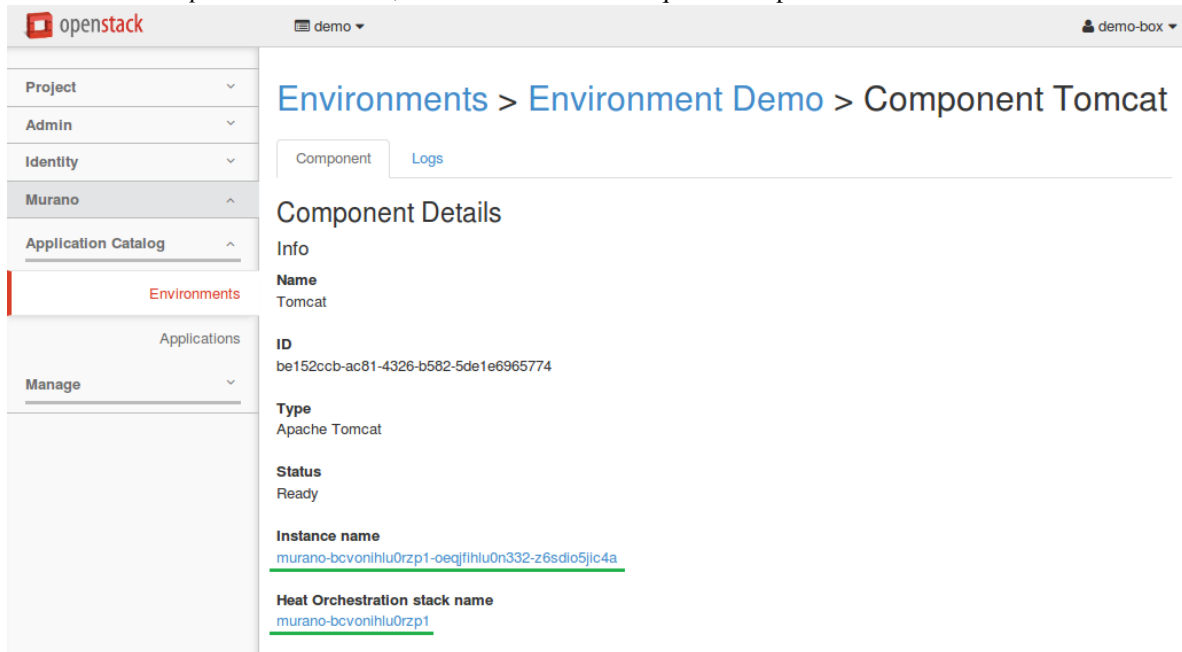
You can browse component details to find the following information about a component:

- Name
- ID
- Type
- Instance name (available only after deployment)
- Heat orchestration stack name (available only after deployment)

To browse a component details, perform the following steps:

1. In OpenStack Dashboard, navigate to *Murano > Application Catalog > Environments*.

2. Click the name of the required environment.
3. In the *Component List* section, click the name of the required component.



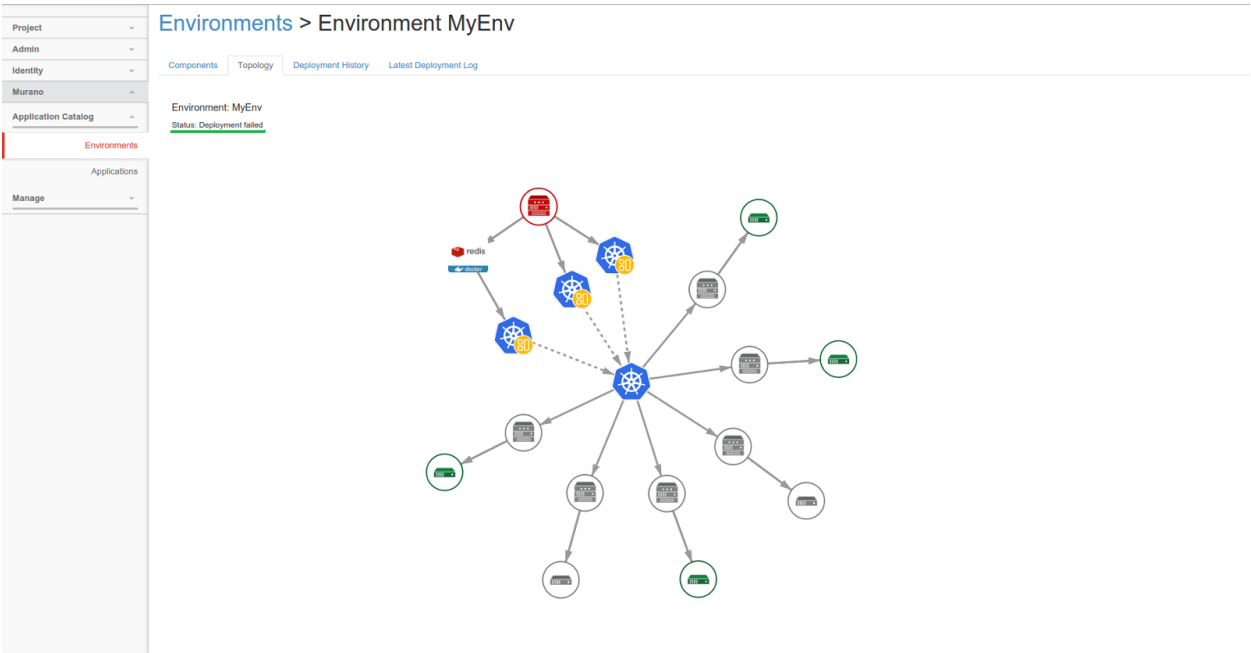
The links redirect to corresponding horizon pages with the detailed information on instance and heat stack.

Application topology

Once you add an application to your environment, the application topology of this environment becomes available in a separate tab. The topology represents an elastic diagram showing the relationship between a component and the infrastructure it runs on. To view the topology:

1. In OpenStack Dashboard, navigate to *Murano > Application Catalog > Environments*.
2. Click the name of the necessary environment.
3. Click the *Topology* tab.

The topology is helpful to visually display complex components, for example Kubernetes. The red icons reflect errors during the deployment while the green ones show success.



The following elements of the topology are virtual machine and an instance of dependent MuranoPL class:

Element	Meaning
	Virtual machine
	Instance

Position your mouse pointer over an element to see its name, ID, and other details.

openstack kate

Environments > Environment quick-env-1

Components Topology Deployment History Latest Deployment Log

Environment: quick-env-1
Status: Deployed

Name: yquidlgv70d45
Availabilityzone: nova
Openstackid: a4187ea9-0943-4509-8109-c3689add3981
Securitygroupname: None
Image: 1b9f37e-df3-4308-be08-9185705dad91
Id: 1079939a-1cf2-4c50-8477-eb41b086c336
Keyname: None
Floatingipaddress: None
Flavor: m1_medium
Type: io.murano.resources.LinuxMuranoInstance
Assignfloatingip: False

MySQL Apache

Deployment logs

To get detailed information on a deployment, use:

- *Deployment history*, which contains logs and deployment structure of an environment.
- *Latest deployment log*, which contains information on the latest deployment of an environment.
- *Component logs*, which contain logs on a particular component in an environment.

Deployment history

To see the log of a particular deployment, proceed with the steps below:

1. In OpenStack Dashboard, navigate to *Murano > Application Catalog > Environments*.
2. Click the name of the required environment.
3. Click the *Deployment History* tab.
4. Find the required deployment and click *Show Details*.
5. Click the *Logs* tab to see the logs.

The screenshot shows the OpenStack Murano dashboard interface. The top navigation bar includes the OpenStack logo, a user profile for 'kate', and a dropdown menu. The left sidebar contains a navigation menu with 'Project', 'Admin', 'Identity', 'Murano', and 'Application Catalog'. The main content area displays the breadcrumb 'Environments > Environment Demo > Deployment at 2015-10-23 12:57:00'. Below this, there are tabs for 'Configuration' and 'Logs'. The 'Logs' tab is active, showing a list of deployment logs for the 'Environment Demo'.

Deployment Logs

- 2015-10-23 12:57:00 — Action deploy is scheduled
- 2015-10-23 12:57:03 — Creating VM for Apache Server.
- 2015-10-23 12:57:03 — Creating VM for Tomcat
- 2015-10-23 12:57:33 — Instance is created. Deploying Apache
- 2015-10-23 12:57:52 — Instance is created. Deploying Tomcat
- 2015-10-23 12:58:04 — Apache is available at <http://10.0.95.2>
- 2015-10-23 12:58:04 — Apache is installed.
- 2015-10-23 12:58:42 — Tomcat is installed
- 2015-10-23 12:58:43 — Tomcat is available at <http://10.0.95.4>
- 2015-10-23 12:58:44 — Deployment finished

Latest deployment log

To see the latest deployment log, proceed with the steps below:

1. In OpenStack Dashboard, navigate to *Murano > Application Catalog > Environments*.
2. Click the name of the required environment.
3. Click the *Latest Deployment Log* tab to see the logs.

Component logs

To see the logs of a particular component of an environment, proceed with the steps below:

1. In OpenStack Dashboard, navigate to *Murano > Application Catalog > Environments*.
2. Click the name of the required environment.
3. In the *Component List* section, click the required component.
4. Click the *Logs* tab to see the component logs.

The screenshot shows the OpenStack Murano dashboard interface. The top navigation bar includes the OpenStack logo, a user profile for 'kate', and a dropdown menu. The left sidebar contains a navigation menu with 'Project', 'Admin', 'Identity', 'Murano', and 'Application Catalog'. The main content area displays the breadcrumb 'Environments > Environment Demo > Component ApacheHttpServer'. Below this, there are tabs for 'Component' and 'Logs'. The 'Logs' tab is active, showing a list of component logs for the 'ApacheHttpServer' component.

Component Logs

- 2015-10-23 12:57:03 - Creating VM for Apache Server.
- 2015-10-23 12:57:33 - Instance is created. Deploying Apache
- 2015-10-23 12:58:04 - Apache is available at <http://10.0.95.2>
- 2015-10-23 12:58:04 - Apache is installed.

2.3.6 Delete an application

To delete an application that belongs to the environment:

1. In OpenStack dashboard, navigate to *Murano > Application Catalog > Environments*.
2. Click on the name of the environment you want to delete an application from.

The screenshot shows the OpenStack Murano 'Environments' page. The left sidebar contains a navigation menu with 'Environments' selected. The main content area displays a table of environments. The table has columns for Name, Status, and Actions. The environments listed are Env-1, Environment-2, Environment-3, and quick-env-1, all with a status of 'Ready to deploy'. A 'Delete Component' button is visible in the Actions column for Env-1.

Name	Status	Actions
Env-1	Ready to deploy	Manage Components
Environment-2	Ready to deploy	Manage Components
Environment-3	Ready to deploy	Manage Components
quick-env-1	Ready to deploy	Manage Components

3. In the *Component List* section, click the *Delete Component* button next to the application you want to delete. Then confirm the deletion.

The screenshot shows the OpenStack Murano 'Environment Env-1' page. The left sidebar contains a navigation menu with 'Environments' selected. The main content area displays the 'Application Components' section. It shows a list of components: Apache HTTP S..., Apache Tomcat, Docker Jenkins, Docker MariaDB, Docker Standalo..., and Kubernetes Clus... The 'Delete Component' button is visible in the Actions column for the 'Tomcat' component.

Name	Type	Status	Last operation	Time updated	Actions
Tomcat	Apache Tomcat	Ready to deploy	Component draft created	-	Delete Component

Note: If the application that you are deleting has already been deployed, you should redeploy the environment to apply the recent changes. If the environment has not been deployed with this component, the changes are applied immediately on receiving the confirmation.

Warning: Due to a known bug in murano as of Kilo release, the OS resources allocated by a deleted application might not be reclaimed until you delete the environment. See the [Deallocating stack resources](#) blueprint for details.

2.4 Log into murano-spawned instance

After the application is successfully deployed, you may need to log into the virtual machine with the installed application. Follow the steps below. Follow the steps below

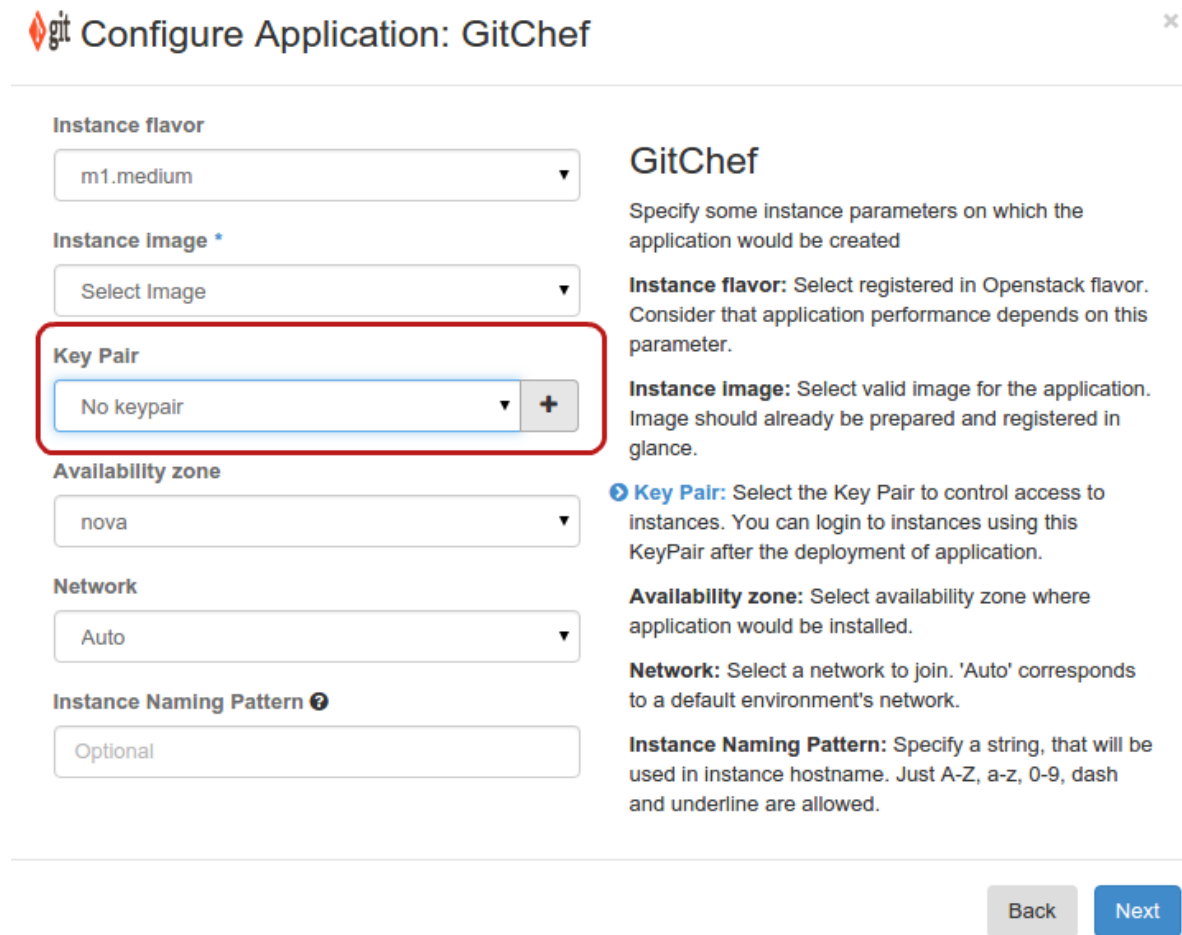
All cloud images (including images imported from [The OpenStack Application Catalog](#)) have password authentication turned off. That is why it is not possible to log in from the dashboard console. So SSH is used to reach an instance spawned by murano.


Possible default image users are:

- *ec2-user*
- *ubuntu* or *debian* (depending on the operation system)

1. Prepare a key pair.

To log in through SSH, provide a key pair during the application creation. If you do not have a key pair, click the plus sign to create one directly from the Configure Application dialog.




Configure Application: GitChef
×

Instance flavor

m1.medium ▼

Instance image *

Select Image ▼

Key Pair

No keypair ▼ +

Availability zone

nova ▼

Network

Auto ▼

Instance Naming Pattern ?

Optional

GitChef

Specify some instance parameters on which the application would be created

Instance flavor: Select registered in Openstack flavor. Consider that application performance depends on this parameter.

Instance image: Select valid image for the application. Image should already be prepared and registered in glance.

Key Pair: Select the Key Pair to control access to instances. You can login to instances using this KeyPair after the deployment of application.

Availability zone: Select availability zone where application would be installed.

Network: Select a network to join. 'Auto' corresponds to a default environment's network.

Instance Naming Pattern: Specify a string, that will be used in instance hostname. Just A-Z, a-z, 0-9, dash and underline are allowed.

Back

Next

2. After the deployment is completed, find out the instance IP address.

Check out:

- Deployment logs

openstack kate

Environments > Environment quick-env-1

Components Topology Deployment History Latest Deployment Log

Deployment Logs

```

2015-09-28 15:06:53 - Action deploy is scheduled
2015-09-28 15:06:55 - Creating VM for Git Chef example
2015-09-28 15:07:42 - Instance is created. Deploying Git Chef
2015-09-28 15:08:23 - Git Chef is installed at 10.0.15.2
2015-09-28 15:08:24 - Deployment finished
  
```

- Detailed instance parameters.

See the *Instance name* link on the Component Details page.

openstack kate

Environments > Environment quick-env-1 > Component GitChefKate9

Component Logs

Component Details

Info

Name
GitChefKate9

ID
3b302721-e9b3-41dd-a4b8-6126d19d8bac

Type
GitChef

Status
Ready

Instance name
[murano-seprpf42cj2-qmyhiif42j8042-m2bgt2occca4](#)

Heat Orchestration stack name
[murano-seprpf42cj2](#)

- To connect to the instance through SSH with the key pair, run:

```
$ ssh ec2-user@<IP> -i <key.location>
```

2.5 Deploying environments using CLI

The main tool for deploying murano environments is murano-dashboard. It is designed to be easy-to-use and intuitive. But it is not the only tool you can use to deploy a murano environment, murano CLI client also possesses required functionality for the task. This is an advanced scenario, however, that requires knowledge of *internal murano workflow*, *murano object model*, and *murano environment* lifecycle. This scenario is suitable for deployments without horizon or deployment automation.

Note: This is an advanced mechanism and you should use it only when you are confident in what you are doing. Otherwise, it is recommended that you use murano-dashboard.

2.5.1 Create an environment

The following command creates a new murano environment that is ready for configuration. For convenience, this guide refers to environment ID as `$ENV_ID`.

```
murano environment-create deployed_from_cli
```

ID	Name	Created	Updated
a66e5ea35e9d4da48c2abc37b5a9753a	deployed_from_cli	2015-10-06T13:50:45	2015-10-06T13:50:45

2.5.2 Create a configuration session

Murano uses configuration sessions to allow several users to edit and configure the same environment concurrently. Most of environment-related commands require the `--session-id` parameter. For convenience, this guide refers to session ID as `$SESS_ID`.

To create a configuration session, use the **murano environment-session-create `$ENV_ID`** command:

```
murano environment-session-create $ENV_ID
```

Property	Value
id	5cbe7e561ffc484ebf11aabf83f9f4c6

2.5.3 Add applications to an environment

To manipulate environments object model from CLI, use the **environment-apps-edit** command:

```
murano environment-apps-edit --session-id $SESS_ID $ENV_ID object_model_patch.json
```

The `object_model_patch.json` contains the jsonpatch object. This object is applied to the `/services` key of the environment in question. Below is an example of the `object_model_patch.json` file content:

```
[
  {
    "op": "add", "path": "/-", "value": {
      "instance": {
        "availabilityZone": "nova",
        "name": "xwvupifdxq27t1",
        "image": "fa578106-b3c1-4c42-8562-4e2e2d2a0a0c",
        "keyname": "",
        "flavor": "m1.small",
        "assignFloatingIp": false,
        "?": {
          "type": "io.murano.resources.LinuxMuranoInstance",
          "id": "===idl==="
        }
      }
    },
  },
]
```

```

    "name": "ApacheHttpServer",
    "enablePHP": true,
    "?": {
      "type": "io.murano.apps.apache.ApacheHttpServer",
      "id": "===id2==="
    }
  }
}
]

```

For convenience, the murano client replaces the "===id1===", "===id2===" (and so on) strings with UUIDs. This way you can ensure that object IDs inside your object model are unique. To learn more about jsonpatch, consult jsonpatch.com and [RFC 6902](https://tools.ietf.org/html/rfc6902). The **murano-environment-edit** command fully supports jsonpatch. This means that you can alter, add, or remove parts of your applications object model.

2.5.4 Verify your object model

To verify whether your object model is correct, check the environment by running the **environment-show** command with the `--session-id` parameter:

```
murano environment-show $ENV_ID --session-id $SESS_ID --only-apps
```

```

[
  {
    "instance": {
      "availabilityZone": "nova",
      "name": "xwvupifdxq27t1",
      "assignFloatingIp": false,
      "keyname": "",
      "flavor": "m1.small",
      "image": "fa578106-b3c1-4c42-8562-4e2e2d2a0a0c",
      "?": {
        "type": "io.murano.resources.LinuxMuranoInstance",
        "id": "fc4fe975f5454bab99bb0e309249e2d2"
      }
    },
    "?": {
      "status": "pending",
      "type": "io.murano.apps.apache.ApacheHttpServer",
      "id": "69cdf10d31e64196b4de894e7ea4f1be"
    },
    "enablePHP": true,
    "name": "ApacheHttpServer"
  }
]

```

2.5.5 Deploy your environment

To deploy a session `$SESS_ID` of your environment, use the **murano environment-deploy** command:

```
murano environment-deploy $ENV_ID --session-id $SESS_ID
```

You can later use the **murano environment-show** command to track the deployment status.

To view the deployed applications of a particular environment, use the **murano environment-show** command with the `--only-apps` parameter and specifying the environment ID:

```
murano environment-show $ENV_ID --only-apps
```

Developing Applications

3.1 Step-by-Step

The goal of this manual is to walk you through the steps that should be taken while composing an application package to get it ready for uploading to Murano.

This tutorial uses a demo application named ApacheHTTPServer to demonstrate how you can create your own Murano application from scratch. We will walk you through its source code and explain how to upload it.

ApacheHTTPServer is a simple Murano application that spawns a virtual machine and installs Apache HTTP Server on it. It may also install php if a user wants to.

The source code of ApacheHTTPServer is available at [github](#).

ApacheHTTPServer's source code is written in MuranoPL. This programming language is object-oriented, and we will see classes, objects and object instances. The detailed explanation of its syntax can be found in the [MuranoPL reference](#).

Warning: Before you start the Murano application creation process, please consider the *System prerequisites* and *Lab requirements* in order you do not risk starting with a wrong environment

3.1.1 Step 1. Create the structure of the package

You should structure an application package very neatly in order the application could be managed and deployed in the catalog successfully.

The package structure of ApacheHTTPServer package is:

```
..
|_ Classes
|   |_ ApacheHttpServer.yaml
|
|_ Resources
|   |_ scripts
|       |_runApacheDeploy.sh
|   |_ DeployApache.template
|
|_ UI
|   |_ ui.yaml
|
|_ logo.png
```

```
|  
|_ manifest.yaml
```

The detailed information regarding the package structure can be found in the *Murano packages* section.

3.1.2 Step 2. Create the manifest file

The application manifest file contains general application metadata. It is an entry-point for each Murano application, and is very similar to the manifest of a jar archive. It has a fixed format based on YAML.

The ApacheHTTPServer’s manifest file:

```
1  Format: 1.0  
2  Type: Application  
3  FullName: io.murano.apps.apache.ApacheHttpServer  
4  Name: Apache HTTP Server  
5  Description: |  
6    The Apache HTTP Server Project is an effort to develop and maintain an  
7    open-source HTTP server for modern operating systems including UNIX and  
8    Windows NT.  
9    ...  
10 Author: Mirantis, Inc  
11 Tags: [HTTP, Server, WebServer, HTML, Apache]  
12 Classes:  
13   io.murano.apps.apache.ApacheHttpServer: ApacheHttpServer.yaml
```

Now, let’s inspect `manifest.yaml` line-by-line.

Format

Specifies the version of the format for `manifest.yaml` to track the syntax changes. Format key presents in each manifest file. Currently, `1.0` is the only available version:

```
Format: 1.0
```

Type

Specifies the type of the package:

```
Type: Application
```

Note: `Application` starts with the capital letter. This is the naming convention for all the pre-defined values in Murano code.

FullName

Stands for the unique service application name. That name allows to easily recognize to which scope an application belongs. All other applications can address to the Apache application methods by this name.

To ensure the global uniqueness, the same naming convention as the naming convention of Java packages and classes is followed. The `io.murano.apps.apache.` part is the “package” part of the name, while `ApacheHttpServer` stands for the “class” part of the name:

FullName: io.murano.apps.apache.ApacheHttpServer

Note: It is not necessary that all applications belong to one domain. This naming allows to determine an application group by its name. OpenStack-related applications may have full names, started with `org.openstack.apps`, for example, `org.openstack.apps.Rally`

Name

Stands for the display name of the application. You will be able to reset a display name when you upload ApacheHTTPServer package to Murano:

Name: Apache HTTP Server

Description

Contains the application description rendered under the application title:

```
1 Description: |
2   The Apache HTTP Server Project is an effort to develop and maintain an
3   open-source HTTP server for modern operating systems including UNIX and
4   Windows NT. The goal of this project is to provide a secure, efficient and
5   extensible server that provides HTTP services in sync with the current HTTP
6   standards.
7   Apache httpd has been the most popular web server on the Internet since
8   April 1996, and celebrated its 17th birthday as a project this February.
```

Let's take a closer look at the syntax:

The vertical line `|` symbol comes from YAML syntax. The `>` symbol can be used interchangeably. These are the [YAML block style indicators](#), which mean that all the leading indents and new line symbols should be preserved. This is very useful for long, multi-line descriptions, because this affects how they are displayed on the UI.

Warning: Avoid tab symbols inside YAML files. If YAML contains the tab symbol, it will not be parsed correctly. The error message may be cryptic or misleading. We recommend that you check the YAML syntax before composing the application package using any of the available online tools.

Author

Contains the name of the author of an application, it is only displayed in the application details and does not affect anything.

Author: Mirantis, Inc

Note: Single quotes usage is optional here: Author: `'Mirantis, Inc'`, thus they are omitted in the code extract below.

Tags

Is an array of tags. You can search an application by its tag. You may want to specify several tags for one application:

Tags: [HTTP, Server, WebServer, HTML, Apache]

Besides, YAML allows tag specification using another syntax, which is an equivalent to the one given above:

```
Tags:
  - HTTP
  - Server
  - WebServer
  - HTML
  - Apache
```

Classes

Is a mapping between all classes present in ApacheHttpServer application and the file names where these classes are defined in. This is one-to-one relationship, which means that there is one and the only class per a single file.

The line `io.murano.apps.apache.ApacheHttpServer: ApacheHttpServer.yaml` says that the class `io.murano.apps.apache.ApacheHttpServer` is defined in the file `ApacheHttpServer.yaml`:

```
Classes:
  io.murano.apps.apache.ApacheHttpServer: ApacheHttpServer.yaml
```

3.1.3 Step 3. Create the execution plan template

The execution plan template contains the instructions understandable to the murano agent on what should be executed to deploy an application. It is the file with the `.template` extension located in the `/APP_NAME/Resources` directory.

The ApacheHTTPServer's `DeployApache.template`:

```
1 FormatVersion: 2.0.0
2 Version: 1.0.0
3 Name: Deploy Apache
4
5 Parameters:
6   enablePHP: $enablePHP
7
8 Body: |
9   return apacheDeploy('{0}'.format(args.enablePHP)).stdout
10
11 Scripts:
12   apacheDeploy:
13     Type: Application
14     Version: 1.0.0
15     EntryPoint: runApacheDeploy.sh
16     Files: []
17     Options:
18       captureStdout: true
19       captureStderr: true
```

As it can be viewed from the source code, besides specifying versions of different items, ApacheHTTPServer execution plan accepts the `enablePHP` parameter. This parameter is an input parameter to the `apacheDeploy.sh` script. This script initiates `runApacheDeploy.sh` execution, which is also located at the `Resources` directory and installs apache app and php if selected.

For the detailed information regarding the execution plan template, its sections and syntax, please refer to the [Execution plan template](#).

3.1.4 Step 4. Create the dynamic UI form definition

ApacheHTTPServer's ui.yaml source code:

```

1  Version: 2
2
3  Application:
4    ?:
5      type: io.murano.apps.apache.ApacheHttpServer
6      name: $.appConfiguration.name
7      enablePHP: $.appConfiguration.enablePHP
8      instance:
9        ?:
10         type: io.murano.resources.LinuxMuranoInstance
11         name: generateHostname($.instanceConfiguration.unitNamingPattern, 1)
12         flavor: $.instanceConfiguration.flavor
13         image: $.instanceConfiguration.osImage
14         keyname: $.instanceConfiguration.keyPair
15         availabilityZone: $.instanceConfiguration.availabilityZone
16         assignFloatingIp: $.appConfiguration.assignFloatingIP
17
18  Forms:
19    - appConfiguration:
20      fields:
21        - name: license
22          type: string
23          description: Apache License, Version 2.0
24          hidden: true
25          required: false
26        - name: name
27          type: string
28          label: Application Name
29          initial: 'ApacheHttpServer'
30          description: >-
31            Enter a desired name for the application. Just A-Z, a-z, 0-9, dash and
32            underline are allowed
33        - name: enablePHP
34          label: Enable PHP
35          type: boolean
36          description: >-
37            Add php support to the Apache WebServer
38          initial: false
39          required: false
40          widgetMedia:
41            css: {all: ['muranodashboard/css/checkbox.css']}
42        - name: assignFloatingIP
43          type: boolean
44          label: Assign Floating IP
45          description: >-
46            Select to true to assign floating IP automatically
47          initial: false
48          required: false
49          widgetMedia:
50            css: {all: ['muranodashboard/css/checkbox.css']}
51        - name: dcInstances
52          type: integer
53          hidden: true
54          initial: 1
55
```

56 . . .

Now, let's inspect it line-by-line.

Application

Defines the object model by which engine deploys the ApacheHTTPServer application, and includes YAQL expressions.

The section contains the reference to the Apache class, the one that is provided in the manifest, named with the `?` symbol. This indicates system information:

```
1  Application:
2    ? :
3    type: io.murano.apps.apache.ApacheHttpServer
```

For ApacheHTTPServer application it is defined that the user should input the application name, some instance parameters and decide whether PHP should be enabled or not:

```
enablePHP: $.appConfiguration.enablePHP
```

The *instance* section assumes that the value, entered by the user in the first form named `appConfiguration` is stored in an application object module. The same applies for the instance parameter. Providing the question mark with the defined type `io.murano.resources.LinuxMuranoInstance` indicates an instance of `MuranoPI` object.

```
1  instance:
2    ? :
3    type: io.murano.resources.LinuxMuranoInstance
```

Note: This parameter is named `instance` here because its class definition property has the `instance` name. You can specify any name in the *class definition file*, and then use it in the UI form definition.

Forms

Contains UI forms prototypes that are merged to the application creation wizard.

Each form field will be translated to the Django field and most of the parameters correspond to parameters in the Django form field. All fields are required by default. Hidden fields are used to print extra information in the form description.

After the upload, the section content will be browsed on the left side of the form and its description on the right.

Please take a look at the *Configure Application: Apache HTTP Server* dialog:

Configure Application: Apache HTTP Server

Step 1 Step 2

Application Name *

ApacheHttpServer

☐ Enable PHP

☐ Assign Floating IP

Apache HTTP Server

Apache License, Version 2.0

Application Name: Enter a desired name for the application. Just A-Z, a-z, 0-9, dash and underline are allowed

Enable PHP: Add php support to the Apache WebServer

Assign Floating IP: Select to true to assign floating IP automatically

Next

Note: The *assignFloatingIP* and *enablePHP* boolean fields are shown as checkboxes.

Here is how the second dialog looks like:

Configure Application: Apache HTTP Server

Step 1 **Step 2**

Instance flavor

m1.medium

Instance image *

Select Image

Key Pair

No keypair +

Availability zone

nova

Instance Naming Pattern ?

Optional

Apache HTTP Server

Specify some instance parameters on which the application would be created

Instance flavor: Select registered in Openstack flavor. Consider that application performance depends on this parameter.

Instance image: Select valid image for the application. Image should already be prepared and registered in glance.

Key Pair: Select the Key Pair to control access to instances. You can login to instances using this KeyPair after the deployment of application.

Availability zone: Select availability zone where application would be installed.

Instance Naming Pattern: Specify a string, that will be used in instance hostname. Just A-Z, a-z, 0-9, dash and underline are allowed.

Back Create

For more information about Dynamic UI, please refer to *the main reference*.

3.1.5 Step 5: Define MuranoPL class definitions

All application classes are located in the `Classes` folder. As `ApacheHttpServer` uses only one class, just one file can be found in this directory.

Here is how it looks like:

```

1  Namespaces:
2      =: io.murano.apps.apache
3      std: io.murano
4      res: io.murano.resources
5      sys: io.murano.system
6
7  Name: ApacheHttpServer
8
9  Extends: std:Application
10
11 Properties:
12     name:
13         Contract: $.string().NotNull()

```

```

14
15 enablePHP:
16     Contract: $.bool()
17     Default: false
18
19 instance:
20     Contract: $.class(res:Instance).notNull()
21
22 Methods:
23     initialize:
24         Body:
25             - $_environment: $.find(std:Environment).require()
26
27     deploy:
28         Body:
29             - If: not $.getAttr(deployed, false)
30             Then:
31                 - $_environment.reporter.report($this, 'Creating VM for Apache Server.')
32                 - $securityGroupIngress:
33                     ...
34                 - $_environment.securityGroupManager.addGroupIngress($securityGroupIngress)
35                 - $.instance.deploy()
36                 - $resources: new(sys:Resources)
37                 - $template: $resources.yaml('DeployApache.template').bind(dict(enablePHP => $.enablePHP))
38                 - $_environment.reporter.report($this, 'Instance is created. Deploying Apache')
39                 - $.instance.agent.call($template, $resources)
40                 - $_environment.reporter.report($this, 'Apache is installed.')
41                 - $.setAttr(deployed, true)

```

Now, let's inspect it line-by-line.

Namespaces

Can be named *shortcuts* since this is an additional section which enables short names instead of the long ones:

```

1 Namespaces:
2     =: io.murano.apps.apache
3     std: io.murano
4     res: io.murano.resources
5     sys: io.murano.system

```

Note: `=:` refers to the *current* namespace

Name

Contains the class name that is defined in this file. So full class name will be current namespace and name, provided by corresponding key: `io.murano.apps.apache.ApacheHttpServer`:

Name: `ApacheHttpServer`

Note: One .yaml file should contain only one class definition.

Extends

Determines inheritance, and `io.murano.Application` should be a parent for all the murano applications.

This class has defined `deploy` method and only instances of that class can be used in `Environment` class. `Environment` class, in its turn, is responsible for the deployment configurations. Definition of both classes are located at `meta/io.murano` folder of murano repository.

Thus, if you want to have some modifications of `ApacheHttpServer`, you can set `io.murano.apps.apache.ApacheHttpServer` in the `Extends` section of a new `Application` class:

```
Extends: std:Application
```

Properties

Defines the dictionary. `Apache HTTP Server` application has three properties: `name`, `enablePHP` and `instance`. For each of them certain `Contract` is defined.

Only `enablePHP` is optional, and its default value equals to `false`.

`Instance` is the required parameter and should be an instance of the predefined in core library `io.murano.resources.Instance` class.

Methods

The `initialize` method is like `__init__` in Python, and executes together with properties initialization.

It accesses the environment, which the application belongs to, and is used only for sending reports about the deployment state.

Private variable `_environment` is defined as follows:

```
1 initialize:
2     Body:
3         - $_environment: $.find(std:Environment).require()
```

The `deploy` method sets up instance spawning and configuration. This method should be executed only once. So in the first order deployed variable is checked to be `false` in the current scope.

It performs the following actions:

- configures `securityGroups`;
- initiates new virtual machine spawning: `$.instance.deploy()`
- loads the execution plan template, located in the `Resources` directory to the instance of `resources` class: `$.resources.yaml('DeployApache.template')`
- updates the plan with parameters taken from the user: `bind(dict(enablePHP => $.enablePHP))`
- sends ready-to-execute-plan to murano agent: `$.instance.agent.call($template, $resources)`

3.1.6 Step 6. Add the application logo (optional)

Download or create your own `.png` image associated with your application.

The recommended size is 70x70 px, and the square shape is preferable. There are no limits regarding the image filename. In `Apache HTTP Server` we use the default name `logo.png`:



3.1.7 Step 7. Compose a zip archive

Select all the files prepared for the package and create an archive in zip format. If the command is executed from the console, do not forget to add the `-r` option to include all the attachments.

Note: The manifest file should not contain the root folder. In other words, the manifest should be located in the archive root directory.

Congratulations! Your application is ready to be uploaded to the application catalog.

3.2 Execution plan template

An execution plan template is a set of metadata that describes the installation process of an application on a virtual machine. It is a minimal executable unit that can be triggered in Murano workflows and is understandable to the Murano agent, which is responsible for receiving, correctness verification and execution of the statements included in the template.

The execution plan template is able to trigger any type of script that executes commands and installs application components as the result. Each script included in the execution plan template may consist of a single file or a set of interrelated files. A single script can be reused across several execution plans.

This section is devoted to the structure and syntax of an execution plan template. For different configurations of templates, please refer to the [Examples](#) section.

3.2.1 Template sections

The table below contains the list of the sections that can be included in the execution plan template with the description of their meaning and the default attributes which are used by the agent if any of the listed parameters is not specified.

Section name	Meaning and default value
FormatVersion	a version of the execution plan template syntax format. Default is 1.0.0. Optional
Name	a human-readable name for the execution plan to be used for logging. Optional
Version	a version of the execution plan itself, is used for logging and tracing. Each time the content of the template content changes (main script, attached scripts, properties, etc.), the version value should be incremented. This is in contrast with <code>FormatVersion</code> , which is used to distinguish the execution plan format. The default value is 0.0.0. Optional
Body	string that represents the Python statement and is executed by the murano-agent. Scripts defined in the Scripts section are invoked from here. Required
Parameters	a dictionary of the <code>String->JsonObject</code> type that maps parameter names to their values. Optional.
Scripts	a dictionary that maps script names to their script definitions. Required

3.2.2 FormatVersion property

`FormatVersion` is a property that all other depend on. That is why it is very important to specify it correctly.

`FormatVersion` 1.0.0 (default) is still used by Windows murano-agent. New features that are introduced in Kilo, such as Chef or Puppet, and downloadable files require version 2.1.0, while nearly all the applications in murano-apps repository work with `FormatVersion` 2.0.0. And if you omit the `FormatVersion` property or put something like `<2.0.0`, it will lead to the incorrect behaviour. The same happens if, for example, `FormatVersion=2.1.0`, and a VM has the pre-Kilo agent.

3.2.3 Scripts section

Scripts are the building blocks of execution plan templates. As the name implies those are the scripts for different deployment platforms.

Each script may consists of one or more files. Those files are script's program modules, resource files, configs, certificates etc.

Scripts may be executed as a whole (like a single piece of code), expose some functions that can be independently called in an execution plan script or both. This depends on deployment platform and executor capabilities.

Scripts are specified using `Scripts` attribute of execution plan. This attribute maps script name to a structure (document) that describes the script. It has the following properties:

Type the name of a deployment platform the script is targeted to. The available alternative options for `version>=2.1.0` are `Application`, `Chef`, `Puppet`, and for `version<2.1.0` is `Application` only. String, required.

Version the minimum version of the deployment platform/executor required by the script. String, optional.

EntryPoint the name of the script file that is an entry point for this execution plan template. String, required.

Files the filenames of the additional files required for the script. Thus, if the script specified in the `EntryPoint` section imports other scripts, they should be provided in this section.

The filenames may include slashes that the agent preserve on VM. If a filename is enclosed in the angle brackets (`<...>`) it will be base64-encoded. Otherwise, it will be treated as a plain-text that may affect line endings.

In Kilo, entries for this property may be not just strings but also dictionaries (for example, `filename: URL`) to specify downloadable files or git repositories.

The default value is `[]` that means that no extra files are used. Array, optional.

Options an optional dictionary of type `String->JsonObject` that contains additional options for the script executor. If not provided, an empty dictionary is assumed.

Available alternatives are: `captureStdout`, `captureStderr`, `verifyExitcode` (raise an exception if result is not positive). As Options are executor-dependent, these three alternatives are available for the Application executor, but may have no sense for other types. `captureStdout`, `captureStderr` and `verifyExitcode` require boolean values, and have `True` as their default values.

Dictionary, optional.

Please make sure the files specified in `EntryPoint` and `Files` sections exist.

3.3 HOT packages

3.3.1 Compose a package

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

Automatic package composing

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

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

Note, that optional parameters could be specified:

- name** an application name, copied from a template by default
- logo** an application square logo, by default the heat logo will be used
- description** text information about an application, by default copied from a template
- author** a name of an application author
- output** a name of an output file archive to save locally
- full-name** a fully qualified domain name that specifies exact application location

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

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

Manual package composing

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

- *Step 1. Choose the desired heat orchestration template*
For this example `chef-server.yaml` template will be used.

- *Step 2. Rename it to `template.yaml`*
- *Step 3. Prepare an application logo (optional step)*

It could be any picture associated with the application.

- *Step 4. Create `manifest.yaml` file*

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

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

Type defines a manifest type, should be set to `Application`

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

Description text information about an application

Author a name of an application author or a company

Tags keywords associated with the application

Logo a name of a logo file for an application

Take a look at the example:

```
Format: Heat.HOT/1.0
Type: Application
FullName: io.murano.apps.Chef-Server
Name: Chef Server
Description: "Heat template to deploy Open Source CHEF server on a VM"
Author: Kate
Tags:
  - hot-based
Logo: logo.png
```

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

Applications page looks like:

Project

Admin

Identity

Murano

Application Catalog


Environments

Applications

Manage

Applications

Recent Activity



Chef Server


Heat template to deploy Open Source CHEF server on a VM

[Details »](#)

Add to Env

Quick Deploy

The configuration form, where you can enter template parameters, will be generated automatically and looks as follows:



Configure Application: Chef Server

Step 1

Application Name *

Ssh Key Name *

Chef Flavor Name *

Chef Port

4000

Chef Server Name

OpenSourceChefServer

Chef Image Name *

Rabbit Password

secrete

Chef Server

Heat template to deploy Open Source CHEF server on a VM

Application Name: Enter a desired name for the application. Just A-Z, a-z, 0-9, and dash are allowed

Ssh Key Name: Name of a Key Pair to enable SSH access to the Instance

Chef Flavor Name: Name Flavor to use for server

Chef Port: Port Number

Chef Server Name: The Instance Name

Chef Image Name: Name of image to use for server

Rabbit Password: Password for RabbitMQ

Create

After filling the form the application is ready to be deployed.

3.4 MuranoPL Reference

To develop applications, murano project refers to Murano Programming Language (MuranoPL). It is represented by easily readable YAML and YAQL languages. The sections below describe these languages.

3.4.1 YAML

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

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

MuranoPL engine relies on a host deserialization code when detecting YAQL expressions in a source definition. It provides them as instances of the `YaqlExpression` class rather than plain strings. Usually, YAQL expressions can be distinguished by the presence of `$` (the dollar sign) and operators, but in YAML, a developer can always state the type by using YAML tags explicitly. For example:

```
1  Some text - a string
2  $.something() - a YAQL expression
3  "$.something()" - a string because quotes are used
4  !!str $ - a string because a YAML tag is used
5  !yaql "text" - a YAQL expression because a YAML tag is used
```

3.4.2 YAQL

YAQL (Yet Another Query Language) is a query language that was also designed as a part of the murano project. MuranoPL makes an extensive use of YAQL. A description of YAQL can be found [here](#).

Simply speaking, YAQL is the language for expression evaluation. The following examples are all valid YAQL expressions: `2 + 2`, `foo() > bar()`, `true != false`.

The interesting thing in YAQL is that it has no built in list of functions. Everything YAQL can access is customizable. YAQL cannot call any function that was not explicitly registered to be accessible by YAQL. The same is true for operators. So the result of the expression `2 * foo(3, 4)` completely depends on explicitly provided implementations of “foo” and “operator_*”.

YAQL uses a dollar sign (\$) to access external variables, which are also explicitly provided by the host application, and function arguments. `$variable` is a syntax to get a value of the variable “\$variable”, `$1`, `$2`, etc. are the names for function arguments. “\$” is a name for current object: data on which an expression is evaluated, or a name of a single argument. Thus, “\$” in the beginning of an expression and “\$” in the middle of it can refer to different things.

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

Note: There is no assignment operator in YAQL, and = means comparison, the same what == means in Python.

As YAQL has no access to underlying operating system resources and is fully controllable by the host, it is secure to execute YAQL expressions without establishing a trust to the executed code. Also, because functions are not predefined, different methods can be accessible in different context. So, YAQL expressions that are used to specify property contracts are not necessarily valid in workflow definitions.

3.4.3 Common class structure

Here is a common template for class declarations. Note, that it is in the YAML format.

```

1 Name: class name
2 Namespaces: namespaces specification
3 Extends: [list of parent classes]
4 Properties: properties declaration
5 Methods:
6     methodName:
7         Arguments:
8             - list
9             - of
10            - arguments
11        Body:
12            - list
13            - of
14            - instructions

```

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

Class name

Class names are alphanumeric names of the classes. Traditionally, all class names begin with an upper-case letter symbol and are written in PascalCasing.

In MuranoPL all class names are unique. At the same time, MuranoPL supports namespaces. So, in different namespaces you can have classes with the same name. You can specify a namespace explicitly, like *ns:MyName*. If you omit the namespace specification, MyName is expanded using the default namespace =:. Therefore, MyName equals =:MyName if = is a valid namespace.

Namespaces

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

```

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

```

In the example above, the `srv: Something` class name is automatically translated to `io.murano.services.Something`.

= means the current namespace, so that `MyClass` means `io.murano.services.windows.MyClass`.

If the class name contains the period (.) in its name, then it is assumed to be already fully namespace qualified and is not expanded. Thus `ns.Myclass` remains as is.

Note: To make class names globally unique, we recommend specifying a developer's domain name as a part of the namespace.

Extends

MuranoPL supports multiple inheritance. If present, the `Extends` section shows base classes that are extended. If the list consists of a single entry, then you can write it as a scalar string instead of an array. If you do not specify any parents or omit the key, then the class extends `io.murano.Object`. Thus, `io.murano.Object` is the root class for all class hierarchies.

Properties

Properties are class attributes that together with methods create public class interface. Usually, but not always, properties are the values, and reference other objects that have to be entered in an environment designer prior to a workflow invocation.

Properties have the following declaration format:

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

Contract

Contract is a YAQL expression that says what type of the value is expected for the property as well as additional constraints imposed on a property. Using contracts you can define what value can be assigned to a property or argument. In case of invalid input data it may be automatically transformed to confirm to the contract. For example, if `bool` value is expected and user passes any not null value it will be converted to `True`. If converting is impossible exception `ContractViolationException` will be raised.

The following contracts are available:

Operation	Definition
<code>\$.int()</code>	an integer value (may be null). String values consisting of digits are converted to integers
<code>\$.int().notNull()</code>	a mandatory integer
<code>\$.string()</code> <code>\$.string().notNull()</code>	a string. If the value is not a string, it is converted to a string
<code>\$.bool()</code> <code>\$.bool().notNull()</code>	bools are true and false. 0 is converted to false, other integers to true
<code>\$.class(ns:ClassName)</code> <code>\$.class(ns:ClassName).notNull()</code>	value must be a reference to an instance of specified class name
<code>\$.class(ns:ClassName, ns:DefaultClassName)</code>	create instance of the <code>ns:DefaultClassName</code> class if no instance provided
<code>\$.class(ns:Name).check(\$.p = 12)</code>	the value must be of the <code>ns:Name</code> type and have the <code>p</code> property equal to 12
<code>\$.class(ns:Name).owned()</code>	a current object must be direct or indirect owner of the value
<code>\$.class(ns:Name).notOwned()</code>	the value must be owned by any object except current one
<code>[\$.int()]</code> <code>[\$.int().notNull()]</code>	an array of integers. Similar to other types.
<code>[\$.int().check(\$ > 0)]</code>	an array of the positive integers (thus not null)
<code>[\$.int(), \$.string()]</code>	an array that has at least two elements, first is int and others are strings
3.4. MuranoPL Reference <code>[\$.int(), 2]</code> <code>[\$.int(), 2, 5]</code>	an array of ints with at least 2 items an array of ints with at least 2 items, and maximum of 5 items

In the example above property `port` must be int value greater than 0 and less than 65536; `scope` must be a string value and one of 'public', 'cloud', 'host' or 'internal', and `protocol` must be a string value and either 'TCP' or 'UDP'. When user passes some values to these properties it will be checked that values confirm to the contracts.

Namespaces:

```
=: io.murano.apps.docker
std: io.murano
```

Name: ApplicationPort

Properties:

```
port:
  Contract: $.int().notNull().check($ > 0 and $ < 65536)

scope:
  Contract: $.string().notNull().check($ in list(public, cloud, host, internal))
  Default: private

protocol:
  Contract: $.string().notNull().check($ in list(TCP, UDP))
  Default: TCP
```

Methods:

```
getRepresentation:
  Body:
    Return:
      port: $.port
      scope: $.scope
      protocol: $.protocol
```

Usage

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

Property	Explanation
In	Input property. Values of such properties are obtained from a user and cannot be modified in MuranoPL workflows. This is the default value for the Usage key.
Out	A value is obtained from executing MuranoPL workflow and cannot be modified by a user.
InOut	A value can be modified both by user and by workflow.
Const	The same as In but once workflow is executed a property cannot be changed neither by a user nor by a workflow.
Runtime	A property is visible only from within workflows. It is neither read from input nor serialized to a workflow output.

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

If the workflow tries to write to a property that is not declared with one of the types above, it is considered to be private and accessible only to that class (and not serialized to output and thus would be lost upon the next deployment). An attempt to read the property that was not initialized results in an exception.

Default

Default is a value that is used if the property value is not mentioned in the input object model, but not when it is set to null. Default, if specified, must conform to a declared property contract. If Default is not specified, then null is the default.

For properties that are references to other classes, Default can modify a default value of the referenced objects. For example:

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

This overrides default for the a property of MyClass for instance of MyClass that is created for this property.

Workflow

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

In a typical scenario, the root object in an input data model is of the `io.murano.Environment` type, and has the `deploy` method. This method invocation causes a series of infrastructure activities (typically, a Heat stack modifi-

cation) and the deployment scripts execution initiated by VM agents commands. The role of the workflow is to map data from the input object model, or a result of previously executed actions, to the parameters of these activities and to initiate these activities in a correct order.

Methods

Methods have input parameters, and can return a value to a caller. Methods are defined in the Workflow section of the class using the following template:

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

Action is an optional parameter that specifies methods to be executed by direct triggering after deployment.

Arguments are optional too, and are declared using the same syntax as class properties, except for the Usage attribute that is meaningless for method parameters. For example, arguments also have a contract and optional default:

```
scaleRc:
  Arguments:
    - rcName:
      Contract: $.string().NotNull()
    - newSize:
      Contract: $.int().NotNull()
```

The Method body is an array of instructions that get executed sequentially. There are 3 types of instructions that can be found in a workflow body:

- expressions,
- assignments,
- block constructs.

Expressions

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

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

Assignment

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

Assignment	Explanation
<code>\$x: value</code>	assigns <code>value</code> to the local variable <code>\$x</code>
<code>\$.x: value</code> <code>\$this.x: value</code>	assign the value to the object's property
<code>\$.x: \$.y</code>	copies the value of the property <code>y</code> to the property <code>x</code>
<code>\$x: [\$a, \$b]</code>	sets <code>\$x</code> to the array of two values: <code>\$a</code> and <code>\$b</code>
<code>\$x:</code> <code>SomeKey:</code> <code>NestedKey: \$variable</code>	structures of any level of complexity can be evaluated
<code>\$.x[0]: value</code>	assigns the value to the first array entry of the <code>x</code> property
<code>\$.x.append(): value</code>	appends the value to an array in the <code>x</code> property
<code>\$.x.insert(1): value</code>	inserts the value into the position 1
<code>\$x: [\$a, \$b].delete(0)</code>	sets <code>\$x</code> to the array without 0 index item
<code>\$.x.key.subKey: value</code> <code>\$.x[key][subKey]: value</code>	deep dictionary modification

Block constructs

Block constructs control a program flow. They are dictionaries that have strings as all their keys.

The following block constructs are available:

Assignment	Explanation
Return: value	Returns value from a method
If: predicate() Then: - code - block Else: - code - block	<p><code>predicate()</code> is a YAQL expression that must be evaluated to <code>True</code> or <code>False</code></p> <p>The <code>Else</code> section is optional</p> <p>One-line code blocks can be written as scalars rather than an array.</p>
While: predicate() Do: - code - block	<code>predicate()</code> must be evaluated to <code>True</code> or <code>False</code>
For: variableName In: collection Do: - code - block	<p><code>collection</code> must be a YAQL expression returning iterable collection or evaluable array as in assignment instructions, for example, <code>[1, 2, \$x]</code></p> <p>Inside a code block loop, a variable is accessible as <code>\$variableName</code></p>
Repeat: Do: - code - block	Repeats the code block specified number of times
Break:	Breaks from loop
Match: case1: - code - block case2: - code - block	<p>Matches the result of <code>\$valExpression()</code> against a set of possible values (cases). The code block of first matched case is executed.</p> <p>If no case matched and the default key is present than the <code>Default</code> code block get executed.</p> <p>The case values are constant values (not expressions).</p>
Value: \$valExpression() Default: - code - block	

3.4. MuranoPL Reference**65**

Switch:
 \$predicate1():

All code blocks that have their predicate evaluated to `True` are executed, but the order of predicate evaluation is not fixed.

Notice, that if you have more than one block construct in your workflow, you need to insert dashes before each construct. For example:

```
Body:
- If: predicate1()
  Then:
    - code
    - block
- While: predicate2()
  Do:
    - code
    - block
```

Object model

Object model is a JSON serialized representation of objects and their properties. Everything you do in the OpenStack dashboard is reflected in an object model. The object model is sent to the Application catalog engine when the user decides to deploy the built environment. On the engine side, MuranoPL objects are constructed and initialized from the received Object model, and a predefined method is executed on the root object.

Objects are serialized to JSON using the following template:

```
1  {
2      "?": {
3          "id": "globally unique object ID (UUID)",
4          "type": "fully namespace-qualified class name",
5
6          "optional designer-related entries can be placed here": {
7              "key": "value"
8          }
9      },
10
11     "classProperty1": "propertyValue",
12     "classProperty2": 123,
13     "classProperty3": ["value1", "value2"],
14
15     "reference1": {
16         "?": {
17             "id": "object id",
18             "type": "object type"
19         },
20
21         "property": "value"
22     },
23
24     "reference2": "referenced object id"
25 }
```

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

There are two ways to specify references:

1. `reference1` as in the example above. This method allows inline definition of an object. When the instance of the referenced object is created, an outer object becomes its parent/owner that is responsible for the object. The object itself may require that its parent (direct or indirect) be of a specified type, like all applications require to have `Environment` somewhere in a parent chain.
2. Referring to an object by specifying other object ID. That object must be defined elsewhere in an object tree. Object references distinguished from strings having the same value by evaluating property contracts. The former

case would have `$.class (Name)` while the later - the `$.string()` contract.

3.4.4 MuranoPL Core Library

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

Classes included in the Murano core library are as follows:

io.murano

- *object*
- *application*
- *security-group-manager*
- *environment*

io.murano.resources

- *instance*
- *network*

io.murano.system

- *logger*

Class: Object

A parent class for all MuranoPL classes. It implements the `initialize`, `setAttr`, and `getAttr` methods defined in the pythonic part of the Object class. All MuranoPL classes are implicitly inherited from this class.

See also:

Source [Object.yaml](#) file.

Class: Application

Defines an application itself. All custom applications must be derived from this class.

See also:

Source [Application.yaml](#) file.

Class: SecurityGroupManager

Manages security groups during an application deployment.

See also:

Source [SecurityGroupManager.yaml](#) file.

Class: Environment

Defines an environment in terms of the deployment process and groups all Applications and their related infrastructures. It also able to deploy them at once.

Environments is intent to group applications to manage them easily.

Table 3.1: Environment class properties

Property	Description	Default usage
name	An environment name.	In
applications	A list of applications belonging to an environment.	In
agentListener	A property containing the <code>io.murano.system.AgentListener</code> object that can be used to interact with Murano Agent.	Runtime
stack	A property containing a <code>HeatStack</code> object that can be used to interact with Heat.	Runtime
instanceNotifier	A property containing the <code>io.murano.system.InstanceNotifier</code> object that can be used to keep track of the amount of deployed instances.	Runtime
defaultNetwork	A property containing user-defined Networks (<code>io.murano.resources.Network</code>) that can be used as default networks for the instances in this environment.	In
securityGroupManager	A property containing the <code>SecurityGroupManager</code> object that can be used to construct a security group associated with this environment.	Runtime

See also:

Source [Environment.yaml](#) file.

Class: Instance

Defines virtual machine parameters and manages an instance lifecycle: spawning, deploying, joining to the network, applying security group, and deleting.

Table 3.2: Instance class properties

Property	Description	Default usage
name	An instance name.	In
flavor	An instance flavor defining virtual machine hardware parameters.	In
image	An instance image defining operation system.	In
keyname	Optional. A key pair name used to connect easily to the instance.	In
agent	Configures interaction with the Murano agent using <code>io.murano.system.Agent</code> .	Runtime
ipAddresses	A list of all IP addresses assigned to an instance.	Out
networks	Specifies the networks that an instance will be joined to. Custom networks that extend <i>Network class</i> can be specified. An instance will be connected to them and for the default environment network or flat network if corresponding values are set to <code>True</code> . Without additional configuration, instance will be joined to the default network that is set in the current environment.	In
assignFloatingIp	Determines if floating IP is required. Default is <code>False</code> .	In
floatingIpAddresses	IP addresses assigned to an instance after an application deployment.	Out
securityGroup	Optional. A security group that an instance will be joined to.	In

See also:

Source [Instance.yaml](#) file.

Resources

Instance class uses the following resources:

Agent-v2.template Python Murano Agent template.

Note: This agent is supposed to be unified. Currently, only Linux-based machines are supported. Windows support will be added later.

linux-init.sh Python Murano Agent initialization script that sets up an agent with valid information containing an updated agent template.

Agent-v1.template Windows Murano Agent template.

windows-init.sh Windows Murano Agent initialization script.

Class: Network

The basic abstract class for all MuranoPL classes representing networks.

See also:

Source [Network.yaml](#) file.

Class: Logger

Logging API is the part of core library since Liberty release. It was introduced to improve debuggability of MuranoPL programs.

You can get a logger instance by calling a `logger` function which is located in `io.murano.system` namespace. The `logger` function takes a logger name as the only parameter. It is a common recommendation to use full class name as a logger name within that class. This convention avoids names conflicts in logs and ensures a better logging subsystem configurability.

Logger class instantiation:

```
$log: logger('io.murano.apps.activeDirectory.ActiveDirectory')
```

Table 3.3: Log levels prioritized in order of severity

Level	Description
CRITICAL	Very severe error events that will presumably lead the application to abort.
ERROR	Error events that might not prevent the application from running.
WARNING	Events that are potentially harmful but will allow the application to continue running.
INFO	Informational messages highlighting the progress of the application at the coarse-grained level.
DEBUG	Detailed informational events that are useful when debugging an application.
TRACE	Even more detailed informational events comparing to the DEBUG level.

There are several methods that fully correspond to the log levels you can use for logging events. They are `debug`, `trace`, `info`, `warning`, `error`, and `critical`.

Logging example:

```
$log.info('print my info message {message}', message=>message)
```

Logging methods use the same format rules as the YAQL **format** function. Thus the line above is equal to the:

```
$log.info('print my info message {message}'.format(message=>message))
```

To print an exception stacktrace, use the **exception** method. This method uses the **ERROR** level:

```
Try:
- Throw: exceptionName
  Message: exception message
Catch:
With: exceptionName
As: e
Do:
- $log.exception($e, 'something bad happen "{message}"', message=>message)
```

Note: You can configure the logging subsystem through the `logging.conf` file of the Murano Engine.

See also:

- Source [Logger.yaml](#) file.
- [OpenStack networking logging configuration](#).

3.5 Murano actions

Murano action is a type of MuranoPL method. The differences from a regular MuranoPL method are:

- Action is executed on deployed objects.
- Action execution is initiated by API request, you do not have to call the method manually.

So murano action allows performing any operations on objects:

- Getting information from the VM, like a config that is generated during the deployment
- VM rebooting
- Scaling

A list of available actions is formed during the environment deployment. Right after the deployment is finished, you can call action asynchronously. Murano engine generates a task for every action. Therefore, the action status can be tracked.

Note: Actions may be called against any MuranoPL object, including Environment, Application, and any other objects.

To mark a method as an action, use `Usage: Action`.

The following example shows an action that returns an archive with a configuration file:

```
exportConfig:
  Usage: Action
  Body:
    - $__environment.reporter.report($this, 'Action exportConfig called')
    - $resources: new(sys:Resources)
    - $template: $resources.yaml('ExportConfig.template')
    - $result: $__masterNode.instance.agent.call($template, $resources)
    - $__environment.reporter.report($this, 'Got archive from Kubernetes')
    - Return: new(std:File, base64Content => $result.content,
      filename => 'application.tar.gz')
```

List of available actions can be found with environment details or application details API calls. It's located in object model special data. Take a look at the following example:

Request: `http://localhost:8082/v1/environments/<id>/services/<id>`

Response:

```
{
  "name": "SimpleVM",
  "?": {
    "_26411a1861294160833743e45d0eaa9": {
      "name": "SimpleApp"
    },
    "type": "io.murano.apps.Simple",
    "id": "e34c317a-f5ee-4f3d-ad2f-d07421b13d67",
    "_actions": {
      "e34c317a-f5ee-4f3d-ad2f-d07421b13d67_exportConfig": {
        "enabled": true,
        "name": "exportConfig"
      }
    }
  }
}
```

3.6 Murano packages

3.6.1 Package structure

The structure of the Murano application package is predefined. An application could be successfully uploaded to an application catalog.

The application package root folder should contain the following:

manifest.yaml file is an application entry point.

Note: the filename is fixed, do not use any custom names.

Classes folder contains MuranoPL class definitions.

Resources folder contains execution plan templates and the **scripts** folder with all the files required for an application deployment located in it.

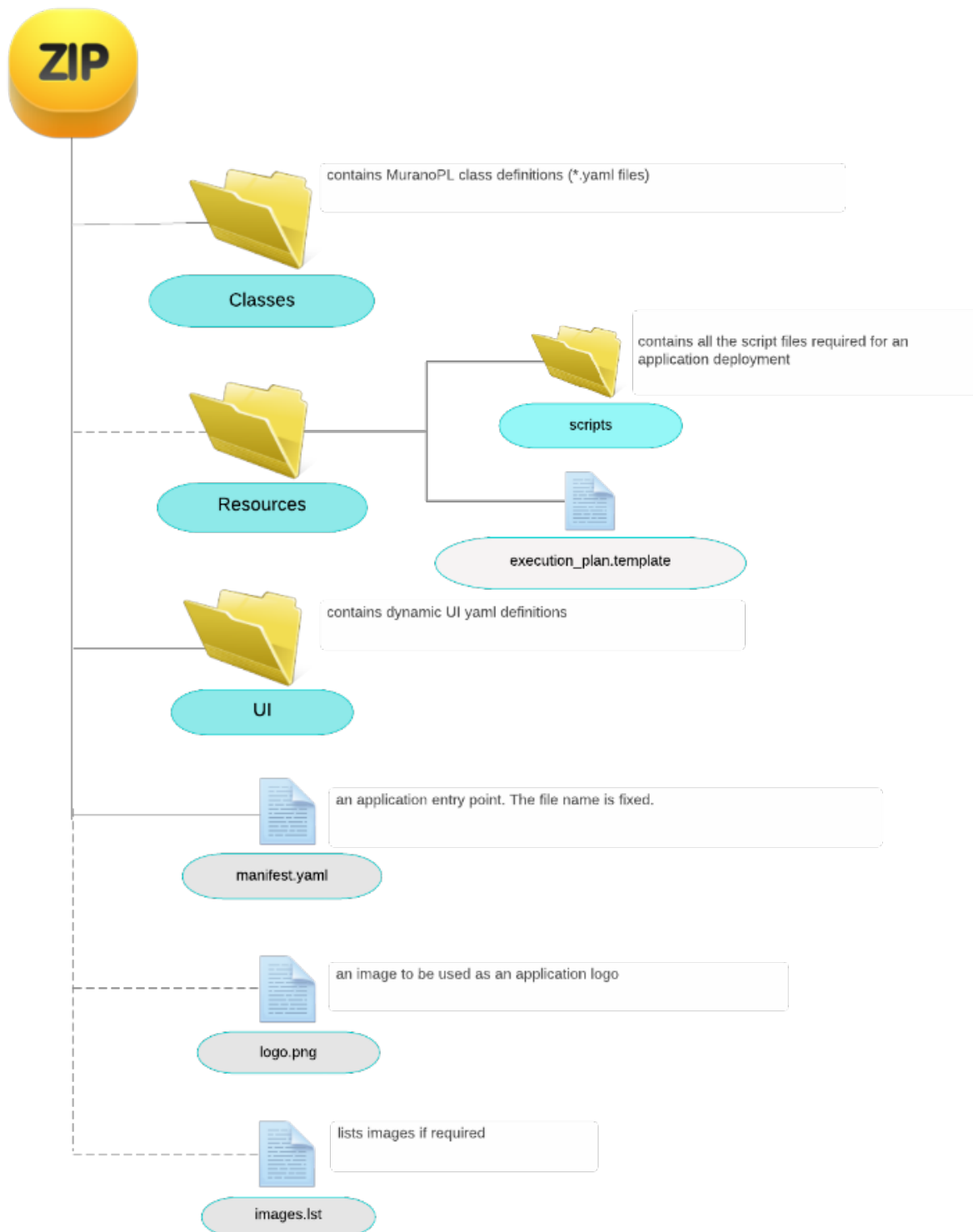
UI folder contains the dynamic UI yaml definitions.

logo.png file (optional) is an image file associated to your application.

Note: There are no any special limitations regarding an image filename. Though, if it differs from the default `logo.png`, specify it in an application manifest file.

images.lst file (optional) contains a list of images required by an application.

Here is the visual representation of the Murano application package structure:



3.6.2 Dynamic UI definition specification

The main purpose of Dynamic UI is to generate application creation forms “on-the-fly”. The Murano dashboard does not know anything about applications that will be presented in the catalog and which web forms are required to create an application instance. So all application definitions should contain an instruction, which tells the dashboard how to create an application and what validations need to be applied. This document will help you to compose a valid UI definition for your application.

The UI definition should be a valid YAML file and may contain the following sections (for version 2.x):

- **Version** Points out the syntax version in use. *Optional*
- **Templates** An auxiliary section, used together with an Application section to help with object model composing. *Optional*
- **Application** Object model description passed to murano engine and used for application deployment. *Required*
- **Forms** Web form definitions. *Required*

Version

The syntax and format of dynamic UI definitions may change over time, so the concept of *format versions* is introduced. Each UI definition file may contain a top-level section called *Version* to indicate the minimum version of Murano Dynamic UI platform which is capable to process it. If the section is missing, the format version is assumed to be latest supported.

The version consists of two non-negative integer segments, separated by a dot, i.e. has a form of *MAJOR.MINOR*. Dynamic UI platforms having the same MAJOR version component are compatible: i.e. the platform having the higher version may process UI definitions with lower versions if their MAJOR segments are the same. For example, Murano Dynamic UI platform of version 2.2 is able to process UI definitions of versions 2.0, 2.1 and 2.2, but is unable to process 3.0 or 1.9.

Currently, the latest version of Dynamic UI platform is 2.3. It is incompatible with UI definitions of Version 1.0, which were used in Murano releases before Juno.

Note: Although the `Version` field is considered to be optional, its default value is the latest supported version. So if you intent to use applications with the previous stable murano version, verify that the version is set correctly.

3.6.3 Version history

Version	Changes	OpenStack Version
1.0	<ul style="list-style-type: none"> Initial Dynamic UI implementation 	Icehouse
2.0	<ul style="list-style-type: none"> <i>instance</i> field support is dropped New <i>Application</i> section that describes engine object model New <i>Templates</i> section for keeping reusable pieces of Object 	Juno, Kilo
2.1	<ul style="list-style-type: none"> New <i>network</i> field provides a selection of networks and their subnetworks as a dropdown populated with those which are available to the current tenant. 	Liberty
2.2	<ul style="list-style-type: none"> Now <i>application name</i> is added automatically to the last service form. It is needed for a user to recognize one created application from another in the UI. Previously all application definitions contained the <i>name</i> property. So to support backward compatibility, you need to manually remove <i>name</i> field from class properties. 	Liberty
2.3	<ul style="list-style-type: none"> Now <i>password</i> field supports <code>confirmInput</code> flag and validator overloading with single <code>regexValidator</code> or multiple <i>validators</i> attribute. 	Mitaka

Application and Templates

The Application section describes an *application object model*. This model will be translated into json, and an application will be deployed according to that json. The application section should contain all necessary keys that are required by the murano-engine to deploy an application. Note that the system section of the object model goes under the `?`. So murano recognizes that instead of simple value, MuranoPL object is used. You can pick parameters you got from a user (they should be described in the Forms section) and pick the right place where they should be set. To do this [YAQL](#) is used. Below is an example of how two YAQL functions are used for object model generation:

- generateHostname** is used for a machine hostname template generation; it accepts two arguments: name pattern (string) and index (integer). If '#' symbol is present in name pattern, it will be replaced with the index provided. If pattern is not given, a random name will be generated.

- **repeat** is used to produce a list of data snippets, given the template snippet (first argument) and number of times it should be reproduced (second argument). Inside that template snippet current step can be referenced as *\$index*.

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

Example:

Templates:

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

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

Application:

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

Forms

This section describes markup elements for defining forms, which are currently rendered and validated with Django. Each form has a name, field definitions (mandatory), and validator definitions (optionally).

Note that each form is splitted into 2 parts:

- **input area** - left side, where all the controls are located
- **description area** - right side, where descriptions of the controls are located

Each field should contain:

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

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

- *string* - text field (no inherent validations) with one-line text input
- *boolean* - boolean field, rendered as a checkbox
- *text* - same as string, but with a multi-line input
- *integer* - integer field with an appropriate validation, one-line text input
- *password* - text field with validation for strong password, rendered as two masked text inputs (second one is for password confirmation)
- *clusterip* - specific text field, used for entering cluster IP address (validations for valid IP address syntax and for that IP to belong to a fixed subnet)
- *databaselist* - specific field, a list of databases (comma-separated list of databases' names, where each name has the following syntax first symbol should be latin letter or underscore; subsequent symbols can be latin letter, numeric, underscore, at the sign, number sign or dollar sign), rendered as one-line text input
- *image* - specific field, used for filtering suitable images by image type provided in murano metadata in glance properties.
- *flavor* - specific field, used for selection instance flavor from a list
- *keypair* - specific field, used for selecting a keypair from a list
- *azone* - specific field, used for selecting instance availability zone from a list
- *network* - specific field, used to select a network and subnet from a list of the ones available to the current user
- any other value is considered to be a fully qualified name for some Application package and is rendered as a pair of controls: one for selecting already existing Applications of that type in an Environment, second - for creating a new Application of that type and selecting it

Other arguments (and whether they are required or not) depends on a field's type and other attributes values. Most of them are standard Django field attributes. The most common attributes are the following:

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

```
- name: someField
  type: string
  label: Domain Name
```



```

validators:
- expr:
    regexpValidator: '([^\.]+\$|^[^\.]{1,15}\..*\$)'
    message: >-
        NetBIOS name cannot be shorter than 1 symbol and
        longer than 15 symbols.
- expr:
    regexpValidator: '([^\.]+\$|^[^\.]*\.[^\.]{2,63}.*\$)'
    message: >-
        DNS host name cannot be shorter than 2 symbols and
        longer than 63 symbols.
helpText: >-
    Just letters, numbers and dashes are allowed.
    A dot can be used to create subdomains

```

- **widgetMedia** sets some custom *CSS* and *JavaScript* used for the field's widget rendering. Note, that files should be placed to Django static folder in advance. Mostly they are used to do some client-side field enabling/disabling, hiding/unhiding etc.
- **requirements** is used only with flavor field and prevents user to pick unstable for a deployment flavor. It allows to set minimum ram (in MBs), disk space (in GBs) or virtual CPU quantity.

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

```

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

```

- **include_subnets** is used only with network field. *True* by default. If *True*, the field list includes all the possible combinations of network and subnet. E.g. if there are two available networks X and Y, and X has two subnets A and B, while Y has a single subnet C, then the list will include 3 items: (X, A), (X, B), (Y, C). If set to *False* only network names will be listed, without their subnets.
- **filter** is used only with network field. *None* by default. If set to a regexp string, will be used to display only the networks with names matching the given regexp.
- **murano_networks** is used only with network field. *None* by default. May have values *None*, *exclude* or *translate*. Defines the handling of networks which are created by murano. Such networks usually have very long randomly generated names, and thus look ugly when displayed in the list. If this value is set to *exclude* then these networks are not shown in the list at all. If set to *translate* the names of such networks are replaced by a string `Network of %env_name%`.

Note: This functionality is based on the simple string matching of the network name prefix and the names of all the accessible murano environments. If the environment is renamed after the initial deployment this feature will not be able to properly translate or exclude its network name.

- **allow_auto** is used only with network field. *True* by default. Defines if the default value of the dropdown (labeled "Auto") should be present in the list. The default value is a tuple consisting of two *None* values. The logic on how to treat this value is up to application developer. It is suggested to use this field to indicate that the instance should join default environment network. For use-cases where such behavior is not desired, this parameter should be set to *False*.

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

Example

Forms:

- appConfiguration:
 - fields:
 - name: dcInstances
 - type: integer
 - hidden: true
 - initial: 1
 - required: false
 - maxLength: 15
 - helpText: Optional field for a machine hostname template
 - name: unitNamingPattern
 - type: string
 - label: Instance Naming Pattern
 - required: false
 - maxLength: 64
 - regexpValidator: '^[a-zA-Z][-_\w]*\$'
 - errorMessages:
 - invalid: Just letters, numbers, underscores and hyphens are allowed.
 - helpText: Just letters, numbers, underscores and hyphens are allowed.
 - description: >-
 - Specify a string that will be used in a hostname instance.
 - Just A-Z, a-z, 0-9, dash, and underline are allowed.
- instanceConfiguration:
 - fields:
 - name: title
 - type: string
 - required: false
 - hidden: true
 - descriptionTitle: Instance Configuration
 - description: Specify some instance parameters based on which service will be created.
 - name: flavor
 - type: flavor
 - label: Instance flavor
 - description: >-
 - Select a flavor registered in OpenStack. Consider that service performance depends on this parameter.
 - required: false
 - name: osImage
 - type: image
 - imageType: windows
 - label: Instance image
 - description: >-
 - Select valid image for a service. Image should already be prepared and registered in glance.
 - name: availabilityZone
 - type: azone
 - label: Availability zone
 - description: Select an availability zone, where service will be installed.
 - required: false

3.6.4 Murano package repository

Murano client and dashboard can install both packages and bundles of packages from murano repository. To do so you should set `MURANO_REPO_URL` settings in murano dashboard or `MURANO_REPO_URL` env variable for the CLI client, and use a respective command to import the package. These commands automatically import all the prerequisites required to install the application along with any images mentioned in the applications.

Setting up your own repository

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

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

When importing an application by name, the client appends any version info, if present to the application name, `.zip` file extension and searches for that file in the `apps` directory.

When importing a bundle by name, the client appends `.bundle` file extension to the bundle name and searches it in the `bundles` directory. A bundle file is a json or a yaml file with the following structure:

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

Glance images can be auto-imported by the client, when mentioned in `images.lst` inside the package. Please see [Step-by-Step](#) for more information about package composition. When importing images from the `image.lst` file, the client simply searches for a file with the same name as the name attribute of the image in the `images` directory of the repository.

3.7 Migrating applications between releases

This document describes how a developer of murano application can update existing packages to make them synchronized with all implemented features and requirements.

3.7.1 Migrate applications from Murano v0.5 to Stable/Juno

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

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

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

For example:

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

Should be changed to:

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

Change the Instance type in the UI definition ‘Application’ section

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

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

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

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

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

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

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

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

to:

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

3.7.2 Migrate applications to Stable/Kilo

In Kilo, there are no breaking changes that affect backward compatibility. But there are two new features which you can use since Kilo.

1. Pluggable Pythonic classes for murano

Now you can create plug-ins for MuranoPL. A plug-in (extension) is an independent Python package implementing functionality which you want to add to the workflow of your application.

For a demo application demonstrating the usage of plug-ins, see the `murano/contrib/plugins/murano_exampleplugin` folder.

The application consist of the following components:

- An `ImageValidatorMixin` class that inherits the generic instance class (`io.murano.resources.Instance`) and adds a method capable of validating the instance image for having an appropriate murano metadata type. This class may be used as a mixin when added to inheritance hierarchy of concrete instance classes.
- A concrete class called `DemoInstance` that inherits from `io.murano.resources.LinuxMuranoInstance` and `ImageValidatorMixin` to add the image validation logic to a standard, murano-enabled and Linux-based instance.
- An application that deploys a single VM using the `DemoInstance` class if the tag on the user-supplied image matches the user-supplied constant.

The **ImageValidatorMixin** demonstrates the instantiation of plug-in provided class and its usage, as well as handling of exception which may be thrown if the plug-in is not installed in the environment.

2. Murano mistral integration

The core library has a new system class for mistral client that allows to call Mistral APIs from the murano application model.

The system class allows you to:

- Upload a mistral workflow to mistral.
- Trigger the mistral workflow that is already deployed, wait for completion and return the execution output.

To use this feature, add some mistral workflow to `Resources` folder of your package. For example, create file `TestEcho_MistralWorkflow.yaml`:

```
version: '2.0'

test_echo:
  type: direct
  input:
    - input_1
  output:
    out_1: <% $.task1_output_1 %>
    out_2: <% $.task2_output_2 %>
    out_3: <% $.input_1 %>
  tasks:
    my_echo_test:
      action: std.echo output='just a string'
      publish:
        task1_output_1: 'task1_output_1_value'
        task1_output_2: 'task1_output_2_value'
      on-success:
        - my_echo_test_2

    my_echo_test_2:
      action: std.echo output='just a string'
      publish:
```

```
task2_output_1: 'task2_output_1_value'
task2_output_2: 'task2_output_2_value'
```

And provide workflow to use the mistral client:

```
Namespaces:
  =: io.murano.apps.test
  std: io.murano
  sys: io.murano.system

Name: MistralShowcaseApp

Extends: std:Application

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

  mistralClient:
    Contract: $.class(sys:MistralClient)
    Usage: Runtime

Methods:
  initialize:
    Body:
      - $this.mistralClient: new(sys:MistralClient)

  deploy:
    Body:
      - $resources: new('io.murano.system.Resources')
      - $workflow: $resources.string('TestEcho_MistralWorkflow.yaml')
      - $.mistralClient.upload(definition => $workflow)
      - $output: $.mistralClient.run(name => 'test_echo', inputs => dict(input_1 => input_1_value))
      - $this.find(std:Environment).reporter.report($this, $output.get('out_3'))
```

3.7.3 Migrate applications to Stable/Liberty

In Liberty a number of useful features that can be used by developers creating their murano applications were implemented. This document describes these features and steps required to include them to new apps.

1. Versioning

Package version

Now murano packages have a new optional attribute in their manifest called *Version* - a standard SemVer format version string. All MuranoPL classes have the version of the package they contained in. To specify the version of your package, add a new section to the manifest file:

```
Version: 0.1.0
```

If no version specified, the package version will be equal to *0.0.0*.

Package requirements

There are cases when packages may require other packages for their work. Now you need to list such packages in the *Require* section of the manifest file:

```
Require:
  package1_FQN: version_spec_1
  ...
  packageN_FQN: version_spec_N
```

version_spec here denotes the allowed version range. It can be either in semantic_version specification pip-like format or as partial version string. If you do not want to specify the package version, leave this value empty:

```
Require:
  package1_FQN: >=0.0.3
  package2_FQN:
```

In this case, the last dependency *0.x.y* is used.

Note: All packages depend on the *io.murano* package (core library). If you do not specify this requirement in the list (or the list is empty or even there is no *Require* key in package manifest), then dependency *io.murano: 0* will be automatically added.

Object version

Now you can specify the version of objects in UI definition when your application requires specific version of some class. To do this, add new key *classVersion* to section *?* describing object:

```
?:
  type: io.test.apps.TestApp
  classVersion: 0.0.1
```

classVersion of all classes included to package equals *Version* of this package.

2. YAQL

In Liberty, murano was updated to use *yaql 1.0.0*. The new version of yaql allows you to use a number of new functions and features that help to increase the speed of developing new applications.

Note: Usage of these features makes your applications incompatible with older versions of murano.

Also, in Liberty you can change *Format* in the manifest of package from *1.0* to *1.1* or *1.2*.

- **1.0** - supported by all versions of murano.
- **1.1** - supported by Liberty+. Specify it, if you want to use features from *yaql 0.2* and *yaql 1.0.0* at the same time in your application.
- **1.2** - supported by Liberty+. A number of features from *yaql 0.2* do not work with this format (see the list below). We recommend you to use it for new applications where compatibility with Kilo is not required.

Some examples of *yaql 0.2* features that are not compatible with the *1.2* format

- Several functions now cannot be called as *MuranoObject* methods: *id()*, *cast()*, *super()*, *psuper()*, *type()*.

- Now you do not have the ability to compare non-comparable types. For example “string != false”
- Dicts are not iterable now, so you cannot do this: If: `$key in $dict`. Use `$key in $dict.keys()` or `$v in $dict.values()`
- Tuples are not available. => always means keyword argument.

3. Simple software configuration

Previously, you always had to create execution plans even when some short scripts had to be executed on a VM. This process included creating a template file, creating a script, and describing the sending of the execution plan to the murano agent.

Now you can use a new class **io.murano.configuration.Linux** from *murano core-library*. This allows sending short commands to the VM and putting files from the `Resources` folder of packages to some path on the VM without the need of creating execution plans.

To use this feature you need to:

- Declare a namespace (for convenience)

```
Namespaces:
  conf: io.murano.configuration
  ...
```

- Create object of `io.murano.configuration.Linux` class in workflow of your application:

```
$linux: new(conf:Linux)
```

- Run one of the two feature methods: `runCommand` or `putFile`:

```
# first agrument is agent of instance, second - your command
$linux.runCommand($.instance.agent, 'service apache2 restart')
```

or:

```
# getting content of file from 'Resources' folder
- $resources: new(sys:Resources)
- $fileContent: $resources.string('your_file.name')
# put this content to some directory on VM
- $linux.putFile($.instance.agent, $fileContent, '/tmp/your_file.name')
```

Note: At the moment, you can use this feature only if your app requires an instance of `LinuxMuranoInstance` type.

4. UI network selection element

Since Liberty, you can provide users with the ability to choose where to join their VM: to a new network created during the deployment, or to an already existing network. Dynamic UI now has a new type of field - `NetworkChoiseField`. This field provides a selection of networks and their subnetworks as a dropdown populated with those which are available to the current tenant.

To use this feature, you should make the following updates in the Dynamic UI of an application:

- Add `network` field:


```

fields:
  - name: network
    type: network
    label: Network
    description: Select a network to join. 'Auto' corresponds to a default environment's network
    required: false
    murano_networks: translate

```

To see the full list of the network field arguments, refer to the UI forms *specification*.

- Add template:

```

Templates:
  customJoinNet:
    - ?:
      type: io.murano.resources.ExistingNeutronNetwork
      internalNetworkName: $.instanceConfiguration.network[0]
      internalSubnetworkName: $.instanceConfiguration.network[1]

```

- Add declaration of *networks* instance property:

```

Application:
  ?:
    type: io.murano.apps.exampleApp
  instance:
    ?:
      type: io.murano.resources.LinuxMuranoInstance
  networks:
    useEnvironmentNetwork: $.instanceConfiguration.network[0]=null
    useFlatNetwork: false
    customNetworks: switch($.instanceConfiguration.network[0], $=null=>list(), $!=null=>$customJ

```

For more details about this feature, see [use-cases](#)

Note: To use this feature, the version of UI definition must be **2.1+**

5. Remove name field from fields and object model in dynamic UI

Previously, each class of an application had a `name` property. It had no built-in predefined meaning for MuranoPL classes and mostly used for dynamic UI purposes.

Now you can create your applications without this property in classes and without a corresponding field in UI definitions. The field for app name will be automatically generated on the last management form before start of deployment. Bonus of deleting this - to remove unused property from muranopl class that is needed for dashboard only.

So, to update existing application developer should make 3 steps:

1. remove `name` field and property declaration from UI definition;
2. remove `name` property from class of application and make sure that it is not used anywhere in workflow
3. set version of UI definition to **2.2 or higher**

3.8 Application unit tests

Murano applications are written in *MuranoPL*. To make the development of applications easier and enable application testing, a special framework was created. So it is possible to add unit tests to an application package and check if the

application is in actual state. Also, application deployment can be simulated with unit tests, so you do not need to run the murano engine.

A separate service that is called *murano-test-runner* is used to run MuranoPL unit tests.

All application test cases should be:

- Specified in the MuranoPL class, inherited from `io.murano.test.testFixture`

This class supports loading object model with the corresponding *load(json)* function. Also it contains a minimal set of assertions such as `assertEqual` and etc.

Note, that test class has the following reserved methods are:

- *initialize* is executed once, like in any other murano application
- *setUp* is executed before each test case
- *tearDown* is executed after each test case

- Named with *test* prefix

```
usage: murano-test-runner [-h] [--config-file CONFIG_FILE]
                        [--os-auth-url OS_AUTH_URL]
                        [--os-username OS_USERNAME]
                        [--os-password OS_PASSWORD]
                        [--os-project-name OS_PROJECT_NAME]
                        [-l [</path1, /path2> [</path1, /path2> ...]]] [-v]
                        [--version]
                        <PACKAGE_FQN>
                        [<testMethod1, className.testMethod2> [<testMethod1, className.testMethod2> ...]]
```

positional arguments:

```
<PACKAGE_FQN>
    Full name of application package that is going to be
    tested
<testMethod1, className.testMethod2>
    List of method names to be tested
```

optional arguments:

```
-h, --help            show this help message and exit
--config-file CONFIG_FILE
                        Path to the murano config
--os-auth-url OS_AUTH_URL
                        Defaults to env[OS_AUTH_URL]
--os-username OS_USERNAME
                        Defaults to env[OS_USERNAME]
--os-password OS_PASSWORD
                        Defaults to env[OS_PASSWORD]
--os-project-name OS_PROJECT_NAME
                        Defaults to env[OS_PROJECT_NAME]
-l [</path1 /path2> [</path1 /path2> ...]], --load_packages_from [</path1 /path2> [</path1 /path2> ...]]
                        Directory to search packages from. Will be used instead of
                        directories, provided in the same option in murano configuration file.
-v, --verbose          increase output verbosity
--version              show program's version number and exit
```

The fully qualified name of a package is required to specify the test location. It can be an application package that contains one or several classes with all the test cases, or a separate package. You can specify a class name to execute all the tests located in it, or specify a particular test case name.

Authorization parameters can be provided in the murano configuration file, or with higher priority *-os-* parameters.

Consider the following example of test execution for the Tomcat application. Tests are located in the same package with application, but in a separate class called `io.murano.test.TomcatTest`. It contains `testDeploy1` and `testDeploy2` test cases. The application package is located in the `/package/location/directory` (murano-apps repository e.g). As the result of the following command, both test cases from the specified package and class will be executed.

```
murano-test-runner io.murano.apps.apache.Tomcat io.murano.test.TomcatTest -l /package/location/direct
```

The following command runs a single `testApacheDeploy` test case from the application package.

```
murano-test-runner io.murano.apps.apache.Tomcat io.murano.test.TomcatTest.testDeploy1
```

The main purpose of MuranoPL unit test framework is to enable mocking. Special *yaql* functions are registered for that:

def inject(target, target_method, mock_object, mock_name) *inject* to set up mock for *class* or *object*, where mock definition is a *name of the test class method*

def inject(target, target_method, yaql_expr) *inject* to set up mock for a *class* or *object*, where mock definition is a *YAQL expression*

Parameters description:

target MuranoPL class name (namespaces can be used or full class name in quotes) or MuranoPL object

target_method Method name to mock in target

mock_object Object, where mock definition is contained

mock_name Name of method, where mock definition is contained

yaql_expr YAQL expression, parameters are allowed

So the user is allowed to specify mock functions in the following ways:

- Specify a particular method name
- Provide a YAQL expression

Consider how the following functions may be used in the MuranoPL class with unit tests:

Namespaces:

```
=: io.murano.test
sys: io.murano.system
```

Extends: TestFixture

Name: TomcatTest

Methods:

```
initialize:
  Body:
    # Object model can be loaded from json file, or provided
    # directly in MuranoPL code as a yaml insertion.
    - $.appJson: new(sys:Resources).json('tomcat-for-mock.json')
    - $.heatOutput: new(sys:Resources).json('output.json')
    - $.log: logger('test')
    - $.agentCallCount: 0

    # Mock method to replace the original one
  agentMock:
    Arguments:
      - template:
```

```
        Contract: $
    - resources:
        Contract: $
    - timeout:
        Contract: $
        Default: null
Body:
    - $.log.info('Mocking murano agent')
    - $.assertEqual('Deploy Tomcat', $template.Name)
    - $.agentCallCount: $.agentCallCount + 1

# Mock method, that returns predefined heat stack output
getStackOut:
    Body:
        - $.log.info('Mocking heat stack')
        - Return: $.heatOutput

testDeploy1:
    Body:
        # Loading object model
        - $.env: $this.load($.appJson)

        # Set up mock for the push method of *io.murano.system.HeatStack* class
        - inject(sys:HeatStack, push, $.heatOutput)

        # Set up mock for the concrete object with mock method name
        - inject($.env.stack, output, $.heatOutput)

        # Set up mock with YAQL function
        - inject('io.murano.system.Agent', call, $this, agentMock)

        # Mocks will be called instead of original function during the deployment
        - $.env.deploy()

        # Check, that mock worked correctly
        - $.assertEqual(1, $.agentCallCount)

testDeploy2:
    Body:
        - inject(sys:HeatStack, push, $this, getStackOut)
        - inject(sys:HeatStack, output, $this, getStackOut)

        # Mock is defined with YAQL function and it will print the original variable (agent template)
        - inject(sys:Agent, call, withOriginal(t => $template) -> $.log.info('{0}', $t))

        - $.env: $this.load($.appJson)
        - $.env.deploy()

        - $isDeployed: $.env.applications[0].getAttr(deployed, false, 'io.murano.apps.apache.Tomcat')
        - $.assertEqual(true, $isDeployed)
```

Provided methods are test cases for the Tomcat application. Object model and heat stack output are predefined and located in the package Resources directory. By changing some object model or heat stack parameters, different cases may be tested without a real deployment. Note, that some asserts are used in those example. The first one is checked, that agent call function was called only once as needed. And assert from the second test case checks for a variable value at the end of the application deployment.

Test cases examples can be found in `TomcatTest.yaml` class of the Apache Tomcat application located at [murano-apps repository](#). You can run test cases with the commands provided above.

3.9 Examples

Application name	Description
Zabbix Agent	<p>Zabbix Agent is a simple application. It doesn't deploy a VM by itself, but is installed on a specific VM that may contain any other applications. This VM is tracked by Zabbix and by its configuration.</p> <p>So Murano performs the Zabbix agent configuration based on the user input. The user chooses the way of instance tracking - HTTP or ICMP that may perform some modifications in the application package.</p> <p>It is worth noting that application scripts are written in Python, not in Bash as usual. This application does not work without Zabbix server application since it's a required property, determined in the application definition. Zabbix Server application interacts with Zabbix Agent by calling its <code>setUpAgent</code> method and providing information about itself: IP and hostname of VM on which the server is installed.</p> <p>Server installs MySQL database and requests database name, password and some other parameters from the user.</p>
Zabbix Server	<p>This is a good example on how difficult logic may be simplified with the inheritance that is supported by MuranoPL. Definition of this app is simple, but the opportunity it provides is fantastic.</p> <p>Crate is a distributed database, in the Murano Application catalog it looks like a regular application. It may be deployed on Google Kubernetes or regular Docker server. The user picks the desired option while filling in the form since these options are set in the UI definition. The form field has a list of possible options:</p> <pre>... type: - io.murano.apps.docker.kubernetes.KubernetesPod - io.murano.apps.docker.DockerStandaloneHost</pre> <p>Information about the application itself (docker image and port that is needed to be opened) is contained in the <code>getContainer</code> method. All other actions for the application configuration are located at the <code>DockerStandaloneHost</code> definition and its dependencies. Note that this application doesn't have a <code>filename:Resources</code> folder at all since the installation is made by Docker itself.</p>
Docker Crate	

3.10 Use-cases

3.10.1 Performing application interconnections

Murano can handle application interconnections installed on virtual machines. The decision of how to combine applications is made by the author of an application.

To illustrate the way such interconnection can be configured, let's analyze the mechanisms applied in WordPress application, which uses MySQL.

MySQL is a very popular database and can be used in quite a number of various applications. Instead of the creation of a database inside definition of the WordPress application, it calls the methods from the MySQL class. At the same time MySQL remains an independent application.

MySQL has a number of methods:

- `deploy`
- `createDatabase`
- `createUser`
- `assignUser`
- `getConnectionString`

In the `io.murano.apps.WordPress` class definition the database property is a contact for the `io.murano.databases.MySql` class. So, the database configuration methods can be called with the parameters passed by the user in the main method:

```
- $.database.createDatabase($.dbName)
- $.database.createUser($.dbUser, $.dbPassword)
- $.database.assignUser($.dbUser, $.dbName)
```

Any other methods of any other class can be invoked the same way to make the proposal application installation algorithm clear and constructive. Also, it allows not to duplicate the code in new applications.

3.10.2 Using application already installed on the image

Suppose you have everything already prepared on image. And you want to share this image with others. This problem can be solved in several ways.

Let's use the [HDPSandbox](#) application to illustrate how this can be done with Murano.

Note: An image may not contain murano-agent at all.

Prepare an application package of the structure:

```
|_ Classes
|   |_ HDPSandbox.yaml
|
|_ UI
|   |_ ui.yaml
|
|_ logo.png
```

Note: The `Resources` folder is not included in the package since the image contains everything that user expects. So no extra instructions are needed to be executed on murano-agent.

UI is provided for specifying the application name, which is used for the application recognition in logging. And what is more, it contains the image name as a deployment instruction template (object model) in the `Application` section:

```

1 Application:
2   ?:
3     type: io.murano.apps.HDPSandbox
4     name: $.appConfiguration.name
5     instance:
6       ?:
7         type: io.murano.resources.LinuxMuranoInstance
8         name: generateHostname($.instanceConfiguration.unitNamingPattern, 1)
9         flavor: $.instanceConfiguration.flavor
10        image: 'hdp-sandbox'
11        assignFloatingIp: true

```

Moreover, the unsupported flavors can be specified here, so that the user can select only from the valid ones. Provide the requirements in the corresponding section to do this:

```

requirements:
  min_disk: 50          (Gb)
  min_memory_mb: 4096   (Mb)
  min_vcpus: 1

```

After the UI form creation, and the `HDPSandbox` application deployment, the VM with the predefined image is spawned. Such type of applications may interact with regular applications. Thus, if you have an image with Puppet, you can call the `deploy` method of the Puppet application and then puppet manifests or any shell scripts on the freshly spawned VM.

The presence of the `logo.png` should never be underestimated, since it helps to make your application recognizable among other applications included in the catalog.

3.10.3 Interacting with non-OpenStack services

This section tells about the interaction between an application and any non-OpenStack services, that have an API.

External load-balancer

Suppose, you have powerful load-balancer on a real server. And you want to run the application on an OpenStack VM. Murano can set up new applications to be managed by that external load-balancer (LB). Let's go into more details.

To implement this case the following apps are used:

- `LbApp`: its class methods call LB API
- `WebApp`: runs on the real LB

Several instances of `WebApp` are deployed with each of them calling two methods:

```

- $.loadBalancer.createPool()
- $.loadBalancer.addMember($instance)
# where $.loadBalancer is an instance of the LbApp class

```

The first method creates a pool and associates it with a virtual server. This happens once only. The second one registers a member in the newly created pool.

It is also possible to perform other modifications to the LB configuration, which are only restricted by the LB API functionality.

So, you need to specify the maximum instance number in the UI form related to the `WebApp` application. All of them are subsequently added to the LB pool. After the deployment, the LB virtual IP, by which an application is accessible, is displayed.

3.10.4 Configuring Network Access for VMs

By default, each VM instance deployed by `io.murano.resources.Instance` class or its descendants joins an environment's default network. This network gets created when the Environment is deployed for the first time, a subnet is created in it and is uplinked to a router which is detected automatically based on its name.

This behavior may be overridden in two different ways.

Using existing network as environment's default

This option is available for users when they create a new environment in the Dashboard. A dropdown control is displayed next to the input field prompting for the name of environment. By default this control provides to create a new network, but the user may opt to choose some already existing network to be the default for the environment being created. If the network has more than one subnet, the list will include all the available options with their CIDRs shown. The selected network will be used as environment's default, so no new network will be created.

Note: Murano does not check the configuration or topology of the network selected this way. It is up to the user to ensure that the network is uplinked to some external network via a router - otherwise the murano engine will not be able to communicate with the agents on the deployed VMs. If the Applications being deployed require internet connectivity it is up to the user to ensure that this net provides it, than DNS nameservers are set and accessible etc.

Modifying the App UI to prompt user for network

The application package may be designed to ask user about the network they want to use for the VMs deployed by this particular application. This allows to override the default environment's network setting regardless of its value.

To do this, application developer has to include a `network` field into the Dynamic UI definition of the app. The value returned by this field is a tuple of `network_id` and a `subnet_id`. This values may be passed as the input properties for `io.murano.resources.ExistingNeutronNetwork` object which may be in its turn passed to an instance of `io.murano.resources.Instance` as its `network` configuration.

The UI definition may look like this:

```
Templates:
  customJoinNet:
    - ?:
      type: io.murano.resources.ExistingNeutronNetwork
      internalNetworkName: $.instanceConfiguration.network[0]
      internalSubnetworkName: $.instanceConfiguration.network[1]
Application:
  ?:
    type: com.example.someApplicationName
  instance:
    ?:
      type: io.murano.resources.LinuxMuranoInstance
    networks:
      useEnvironmentNetwork: $.instanceConfiguration.network[0]=null
```



```

    useFlatNetwork: false
    customNetworks: switch($.instanceConfiguration.network[0], $=null=>list(), $!=null=>$customJoin)
Forms:
- instanceConfiguration:
  fields:
  - name: network
    type: network
    label: Network
    description: Select a network to join. 'Auto' corresponds to a default environment's network
    required: false
    murano_networks: translate

```

For more details on the Dynamic UI its controls and templates please refer to its *specification*.

3.11 FAQ

There are too many files in Murano package, why not to use a single Heat Template?

To install a simple Apache service to a new VM, Heat Template is definitely simpler. But the Apache service is useless without its applications running under it. Thus, a new Heat Template is necessary for every application that you want to run with Apache. In Murano, you can compose a result software to install it on a VM on-the-fly: it is possible to select an application that can run under Apache dynamically. Or you can set a VM where Apache is installed as a parameter. This way, the files in the application package allow to compose compound applications with multiple configuration options. For any single combination you need a separate Heat Template.

The Application section is defined in the UI form. Can I remove it?

No. The `Application` section is a template for Murano object model which is the instruction that helps you to understand the environment structure that you deploy. While filling the forms that are auto-generated from the `UI.yaml` file, object model is updated with the values entered by the user. Eventually, the Murano engine receives the resulted object model (.json file) after the environment is sent to the deploy.

The Templates section is defined in the UI form. What's the purpose?

Sometimes, the user needs to create several instances with the same configuration. A template defined by a variable in the `Templates` section is multiplied by the value of the number of instances that are set by the user. A YAQL `repeat` function is used for this operation.

Some properties have Usage, others do not. What does this affect?

`Usage` indicates how a particular property is used. The default value is `In`, so sometimes it is omitted. The `Out` property indicates that it is not set from outside, but is calculated in the class methods and is available for the `read` operation from other classes. If you don't want to initialize in the class constructor, and the property has no default value, you specify `Out` in the `Usage`.

Can I use multiple inheritance in my classes?

Yes. You can specify a list of parent classes instead of a single string in the regular YAML notation. The list with one element is also acceptable.

There are FullName and Name properties in the manifest file. What's the difference between them?

`Name` is displayed in the web UI catalog, and `FullName` is a system name used by the engine to get the class definition and resolve the class interconnections.

How does Murano know which class is the main one?

There is no `main` class term in the MuranoPL. Everything depends on a particular object model and an instance class representing the instance. Usually, an entry-point class has exactly the same name as the package `FullName`, and it uses other classes.

What is the difference between `$variable` and `$.variable` in the class definitions?

By default, `$` represents a current object (similar to `self` in Python or `this` in C++/Java/C#), so `$.variable` accesses the object field/property. In contrast, `$variable` (without a dot) means a local method variable. Note that `$` can change its value during execution of some YAQL functions like `select`, where it means a current value. A more safe form is to use a reserved variable `$this` instead of `$`. `$this.variable` always refers to an object-level value in any context.

Miscellaneous

Installation

4.1 Murano Installation Guide

4.1.1 Content

Prepare A Lab For Murano

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

System prerequisites

Supported Operating Systems

- Ubuntu Server 12.04 LTS
- RHEL/CentOS 6.4

System packages are required for Murano

Ubuntu

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

CentOS

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

Lab Requirements

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

Table: Hardware requirements

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

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

Test Your Lab Host Performance

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

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

Steps to reproduce test:

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

```
ps aux | grep kvm
```

3. Make 5 copies of Windows image file:

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

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

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

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

```
sudo ./start-vm.sh 1
```

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

```
sudo killall kvm
```

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

```
sudo ./start-vm.sh 5
```

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

```
sudo killall kvm
```

Baseline Data

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

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

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

Host Optimizations

Default KVM installation could be improved to provide better performance.

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

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

Installing and Running the Development Version

The [devstack](#) directory contains the files necessary to integrate Murano with [Devstack](#).

Enabling in Devstack

1. Download DevStack:

```
git clone https://git.openstack.org/openstack-dev/devstack
cd devstack
```

2. Edit local.conf to enable murano devstack plugin:

```
> cat local.conf
[[local|localrc]]
enable_plugin murano git://git.openstack.org/openstack/murano
```

3. If you want Murano Cloud Foundry Broker API service enabled, add the following line to local.conf:

```
enable_service murano-cfapi
```

4. (Optional) To import Murano packages when DevStack is up, define an ordered list of packages FQDNs in local.conf. Make sure to list all package dependencies. These packages will by default be imported from the murano-apps git repository.

Example:

```
MURANO_APPS=io.murano.apps.apache.Tomcat,io.murano.apps.Guacamole
```

You can also use the variables MURANO_APPS_REPO and MURANO_APPS_BRANCH to configure the git repository which will be used as the source for the imported packages.

5. Install DevStack:

```
./stack.sh
```

Installing and Running Manually

Prepare Environment

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

Ubuntu

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

Fedora

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

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

CentOS

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

Install tox

```
sudo pip install tox
```

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

Warning: Although murano could use a PostgreSQL database on the back end, it wasn't thoroughly tested and should be used with caution.

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

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

mysql -u root -p

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

Install the API service and Engine

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

```
mkdir ~/murano
```

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

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

3. Set up the murano config file

Murano has a common config file for API and Engine services.

First, generate a sample configuration file, using tox

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

And make a copy of it for further modifications

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

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

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

```
[DEFAULT]
debug = true
verbose = true
rabbit_host = %RABBITMQ_SERVER_IP%
rabbit_userid = %RABBITMQ_USER%
```

```
rabbit_password = %RABBITMQ_PASSWORD%
rabbit_virtual_host = %RABBITMQ_SERVER_VIRTUAL_HOST%
driver = messagingv2

...

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

...

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

...

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

...

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

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

[networking]
default_dns = 8.8.8.8 # In case openstack neutron has no default
                  # DNS configured
```

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

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

6. Create database tables for Murano.

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

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

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


8. Import Core Murano Library.

```
cd ~/murano/murano
pushd ./meta/io.murano
zip -r ../../io.murano.zip *
popd
tox -e venv -- murano --murano-url http://localhost:8082 \
    package-import --is-public io.murano.zip
```

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

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

Register in Keystone To make the murano API available to all OpenStack users, you need to register the Application Catalog service within the Identity service.

1. Add application-catalog service:

```
openstack service create --name murano --description "Application Catalog for OpenStack" applica
```

2. Provide an endpoint for that service:

```
openstack endpoint create --region RegionOne --publicurl http://<murano-ip>:8082 --internalurl h
```

where MURANO-SERVICE-ID is the unique service number that you can find in the **openstack service create** output.

Note: URLs (publicurl, internalurl and adminurl) may be different depending on your environment.

Install Murano Dashboard

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

1. Clone the repository with Murano Dashboard.

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

2. Clone horizon repository

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

3. Create venv and install muranodashboard as editable module.

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

4. Copy muranodashboard plugin file.

This step enables murano panel in horizon dashboard.

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

5. Prepare local settings.

To get more information, check out official [horizon documentation](#).

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

6. Customize local settings according to OpenStack installation.

```
...
ALLOWED_HOSTS = '*'

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

...

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

...

DEBUG_PROPAGATE_EXCEPTIONS = DEBUG
```

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

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

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

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

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

7. Perform database synchronization.

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

```
tox -e venv -- python manage.py migrate --noinput
```

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

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

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

Development server will be restarted automatically on every code change.

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

Import Murano Applications

Applications need to be imported to fill the catalog. This can be done via the dashboard, and via CLI:

1. Clone the murano apps repository.

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

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

```
cd ~/murano/murano
pushd ../murano-apps/Docker/Applications/%APP-NAME%/package
zip -r ~/murano/murano/app.zip *
popd
tox -e venv -- murano --murano-url http://localhost:8082 package-import app.zip
```

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

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

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

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

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

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

[networking]

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

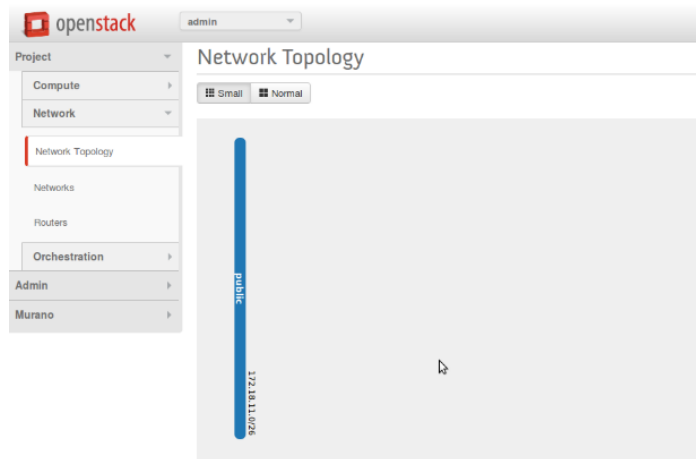
To figure out the name of the external network, perform the following command:

```
openstack network list --external
```

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

Manual neutron network configuration

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



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

Create Network

Network

Subnet *

Subnet Detail

Network Name

Local

Admin State

☒

From here you can create a new network.
In addition a subnet associated with the network can be
created in the next panel.

« Back

Next »

Create Network ✕

Network
Subnet *
Subnet Detail

Create Subnet

☒

Subnet Name

Network Address

IP Version *

Gateway IP

Disable Gateway

☐

You can create a subnet associated with the new network, in which case "Network Address" must be specified. If you wish to create a network WITHOUT a subnet, uncheck the "Create Subnet" checkbox.

« Back
Next »

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

Create Router ✕

Router Name *

Cancel
Create Router

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

```
[networking]

router_name = %SPECIFIED_NAME%
create_router = false
```

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

Add Interface ×

Subnet *

Select Subnet

IP Address (optional) ⓘ

192.168.2.1

Router Name *

murano-default-router

Router ID *

3702b290-f63e-4fba-aabb-ce6f565fdd14

Description:

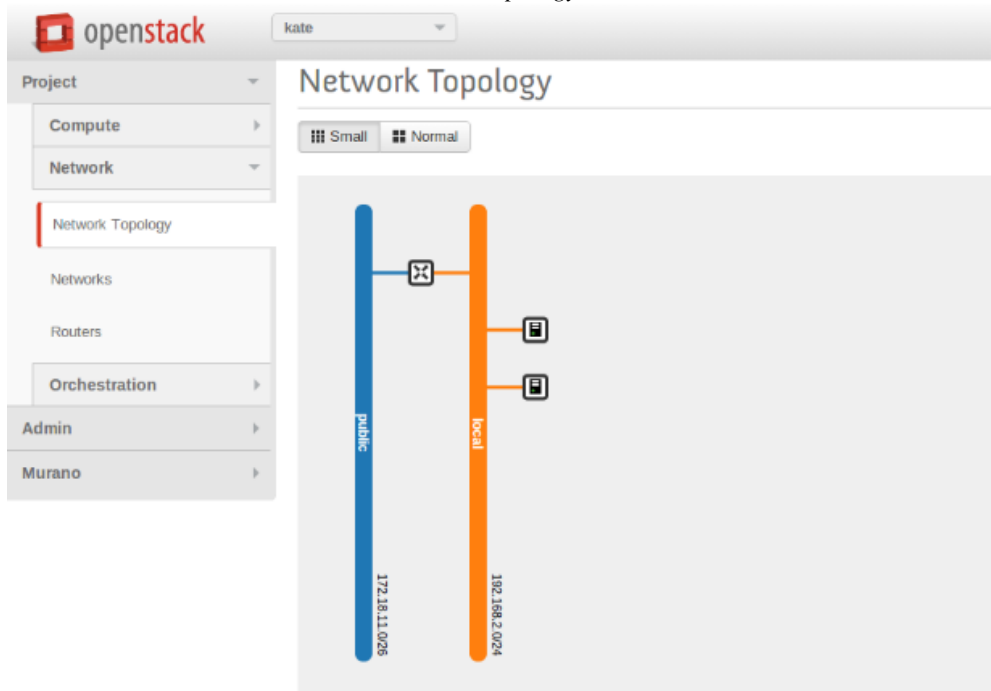
You can connect a specified subnet to the router.

The default IP address of the interface created is a gateway of the selected subnet. You can specify another IP address of the interface here. You must select a subnet to which the specified IP address belongs to from the above list.

Cancel

Add interface

And check the result in *Network Topology* tab.



SSL configuration

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

HTTPS for Murano API

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

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

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

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

SSL for RabbitMQ

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

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

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

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

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

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

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

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

Murano Agent -> Rabbit MQ exchange

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

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

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

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

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

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

</appSettings>
```

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

SSL for Murano Dashboard

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

Background Concepts for Murano

4.2 Murano workflow

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

4.2.1 Step 1. Begin deployment

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

- An *Environment* object (io.murano.Environment) with a *name*
- An object (or objects) referring to networks that need to be created or that already exist

- A list of Applications (e.g. `io.murano.apps.linux.Telnet`). Each Application will contain, or will reference, anything it requires. The Telnet example, has a property called *instance* whose contract states it must be of type `io.murano.resources.Instance`. In turn the Instance has properties it requires (like a name, a flavor, a keypair name).

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

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

The *Telnet class definition* refers to several other classes. It extends *Application* and it requires an *Instance*. It also refers to the *Environment* in which it will be contained, sends reports through the environment's `io.murano.system.StatusReporter` and adds security group rules to the *SecurityGroupManager*.

4.2.2 Step 2. Load definitions

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

- *Network*
- *Instance*
- *Object*
- *Environment*

4.2.3 Step 3. Deploy resources

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

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

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

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

4.2.4 Step 4. Software configuration via murano-agent

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

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

4.2.5 Step 5. Done

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

4.3 Murano Policy Enforcement

4.3.1 Murano Policy Enforcement Example

Introduction

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

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

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

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

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

Example

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

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

Example rules

1. Create `predeploy_errors` rule

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

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

```
predeploy_errors(eid, obj_id, msg) :-  
    murano:objects(obj_id, pid, type),  
    murano:objects(eid, tid, "io.murano.Environment"),  
    murano:connected(eid, pid),
```

```

murano:properties(obj_id, "flavor", flavor_name),
flavor_ram(flavor_name, ram),
gt(ram, 2048),
murano:properties(obj_id, "name", obj_name),
concat(obj_name, ": instance flavor has RAM size over 2048MB", msg)

```

Use this command to create the rule:

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

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

2. Create `flavor_ram` rule

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

Use this command to create the rule:

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

Example rules in murano app deployment

1. Create environment with simple application

- Choose Git application from murano applications
- Create with “**m1.medium**” instance flavor which uses 4096MB so validation will fail

Add Application to “quick-env-3”

Instance flavor

Instance Image

Key Pair
 +

Availability zone

Git Application
Specify some instance parameters on which the application would be created

Instance flavor: Select registered in Openstack flavor. Consider that application performance depends on this parameter.

Instance Image: Select valid image for the application. Image should have Murano agent installed and registered in Glance.

Key Pair: Select the Key Pair to control access to instances. You can login to instances using this KeyPair after application deployment

Availability zone: Select availability zone where the application would be installed.

Back **Create**

2. Deploy environment

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

The screenshot shows the OpenStack Murano web interface. On the left is a navigation menu with links for Project, Admin, Identity, Murano, Application Catalog, Environments, Applications, and Manage. The main content area is titled 'Deployment information' and shows the path 'environments > environment quick-env-2 > deployment at 2015-01-20 04:24:13'. Below this, there are tabs for 'Configuration' and 'Logs'. The 'Logs' tab is selected, displaying the 'Deployment Logs' section. The logs show a sequence of events: '2015-01-20 04:24:13 - Action deploy is scheduled', '2015-01-20 04:24:15 - Model validation failed: ftcen154s1ywb1: instance flavor has RAM size over 2048MB', and '2015-01-20 04:24:15 - Deployment finished with errors'.

4.3.2 Murano Policy Based Modification of Environment Example

Introduction

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

Example Use Cases:

- **install monitoring agent on each VM instance (adding component with the agent and creating relationship between agent and instance)**
- **all Apache server instances must have given certified version (version property is set on all Apache applications within environment to given version)**

These policies are evaluated over data in the form of tables (Congress data structures). A deployed Murano environment must be decomposed to Congress data structures. The decomposed environment is sent to congress for simulation. Congress simulates whether the resulting state needs to be modified. In case that modifications of deployed environment are needed congress returns list of actions which needs to be performed on given environment prior the deployment. Actions and its parameters are returned from congress in YAML format.

Example of action specification returned from congress:

- set keyname property on instance identified by object_id to value production-key

```
set-property: {object_id: c46770dec1db483ca2322914b842e50f, prop_name: keyname, value: production-key}
```

Administrator can use above one line action specification as output of congress rules. This action specification is parsed in murano. Given action class is loaded. Action instance is created. Parsed parameters are supplied to action `__init__` method. Then action is performed on given environment (modify method).

Example

In this example assume that we are in production environment. Administrator needs to enforce that all VM instances will be deployed with secure key pair used for production environment.

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

Example rules

1. Create `predeploy_modify` rule

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

```
predeploy_modify(eid, obj_id, action) :-
    murano:objects(obj_id, pid, type),
    murano:objects(eid, tid, "io.murano.Environment"),
    murano:connected(eid, pid),
    murano:properties(obj_id, "keyname", kn),
    concat("set-property: {object_id: ", obj_id, first_part),
    concat(first_part, ", prop_name: keyname, value: production-key}", action)
```

Use this command to create the rule:

```
congress policy rule create murano_system 'predeploy_modify(eid, obj_id, action):-murano:obj
```

Key pair `production-key` must exists or change it to any existing key pair.

2. Deploy environment and check modification

Deploy any environment and check that instances within the environment were deployed with the key pair specified above.

4.3.3 Murano Policy Enforcement Setup Guide

Introduction

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

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

Setup

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

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

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

1. Murano

Enable policy enforcement in Murano:

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

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

- restart `murano-engine`

2. Congress

Policy enforcement uses following policies:

- **murano policy**

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

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

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

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

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

Following command creates **murano_system** rule

```
# create murano_system policy
openstack congress policy create murano_system

# resolves objects within environment
openstack congress policy rule create murano_system 'murano_env_of_object(oid,eid):-murano:conne
```

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

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

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

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

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

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

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

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

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

4.3.4 Murano Policy Enforcement - Developer Guide

This document describes internals of murano policy enforcement.

Model Decomposition

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

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

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

`murano:objects(object_id, parent_id, type_name)`

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

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

Transformed to these rules:

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

Note: The owner of the environment is a tenant

`murano:properties(object_id, property_name, property_value)`

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

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

Transformed to this rule:

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

Inner properties are also supported using dot notation:

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

Transformed to this rule:

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

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

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

Transformed to these rules:

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

`murano:relationships(source, target, name)`

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

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

Transformed to this rule:

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

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

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

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

Transformed to this rule:

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

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

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

`murano:connected(source, target)`

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


```

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

```

Transformed to rules:

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

murano:parent_types(object_id, parent_name)

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

e.g. `LinuxMuranoInstance > LinuxInstance > Instance`

So this model:

```

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

```

Transformed to these rules:

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

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

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

murano:states(environment_id, state)

Currently only one record for environment is created:

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

Tutorials

4.4 Building Murano Image

4.4.1 MS Windows image builder for OpenStack Murano

Introduction

This repository contains MS Windows templates, powershell scripts and bash scripted logic used to create qcow2 images for QEMU/KVM based virtual machines used in OpenStack.

MS Windows Versions

Supported by builder versions with en_US localization:

- Windows 2012 R2
- Windows 2012 R2 Core
- Windows 2008 R2
- Windows 2008 R2 Core

Getting Started

Trial versions of Windows 2008 R2 / 2012 R2 used by default. You could use these images for 180 days without activation. You could download evaluation versions from official Microsoft website:

- [\[Windows 2012 R2 - download\]](#)
- [\[Windows 2008 R2 - download\]](#)

System requirements

- Debian based Linux distribution, like Ubuntu, Mint and so on.
- Packages required: `qemu-kvm virt-manager virt-goodies virtinst bridge-utils libvirt-bin uuid-runtime samba samba-common cifs-utils`
- User should be able to run sudo without password prompt!

```
sudo echo "${USER}    ALL = NOPASSWD: ALL" > /etc/sudoers.d/${USER}
sudo chmod 440 /etc/sudoers.d/${USER}
```
- Free disk space > 50G on partition where script will spawn virtual machines because of 40G required by virtual machine HDD image.
- Internet connectivity.
- Samba shared resource.

Configuring builder

Configuration parameters to tweak:

```
[default]
```

- `workdir` - place where script would prepare all software required by build scenarios. By *default* is not set, i.e. script directory would be used as root of working space.
- `vmsworkdir` - must contain valid path, this parameter tells script where it should spawn virtual machines.
- `runparallel` - *true* or *false*, **false** set by default. This parameter describes how to start virtual machines, one by one or in launch them in background.

[samba]

- `mode` - *local* or *remote*. In local mode script would try to install and configure Samba server locally. If set to *remote*, you should also provide information about connection.
- `host` - in local mode - is 192.168.122.1, otherwise set proper ip address.
- `user` - set to **guest** by default in case of guest rw access.
- `domain` - Samba server user domain, if not set *host* value used.
- `password` - Samba server user password.
- `image-builder-share` - Samba server remote directory.

MS Windows install preparation:

[win2k12r2] or [win2k8r2] - shortcuts for 2012 R2 and 2008 R2.

- `enabled` - *true* or *false*, include or exclude release processing by script.
- `editions` - standard, core or both(space used as delimiter).
- `iso` - local path to iso file

By default [win2k8r2] - disabled, if you need you can enable this release in *config.ini* file.

Run

Preparation

Run `chmod +x *.sh` in builder directory to make script files executable.

Command line parameters:

`runme.sh` - the main script

- `--help` - shows usage
- `--forceinstall-dependencies` - Runs dependencies install.
- `--check-smb` - Run checks or configuration of Samba server.
- `--download-requirements` - Download all required and configures software except MS Windows ISO.
- `--show-configured` - Shows configured and available to use MS Windows releases.
- `--run` - normal run

Experimental options:

- `--config-file` - Set configuration file location instead of default.

Use cases

All examples below describes changes in `config.ini` file

1. I want to build one image for specific version and edition. For example: version - **2012 R2** and edition - **standard**. Steps to reach the goal:

- Disable `[win2k8r2]` from script processing.

```
[win2k8r2]
enabled=false
```

- Update `[win2k12r2]` with desired edition(**standard**).

```
[win2k12r2]
enabled=true
editions=standard
```

- Execute `runme.sh --run`

2. I want to build two images for specific version with all supported by script editions. For example: **2012 R2** and editions - **standard** and **core**. Steps to reach the goal:

- Disable `[win2k8r2]` from script processing.

```
[win2k8r2]
enabled=false
```

- Update `[win2k12r2]` with desired editions(**standard** and **core**).

```
[win2k12r2]
enabled=true
editions=standard core
```

- Execute `runme.sh --run`

3. I want to build two images for all supported by script versions with specific editions. For example: versions - **2012 R2** and **2008 R2** and edition - **core**. Steps to reach the goal:

- Update `[win2k8r2]` with desired edition(**core**).

```
[win2k8r2]
enabled=true
editions=core
```

- Update `[win2k12r2]` with desired edition(**core**).

```
[win2k12r2]
enabled=true
editions=core
```

- Execute `runme.sh --run`

4.4.2 Linux Image

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

Note: Disk image builder requires sudo rights

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

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

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

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

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

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

Install additional packages required by disk image builder:

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

Export paths where additional dib elements are located:

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

Add `passenv = ELEMENTS_PATH` at `testenv:venv` section in `tox.ini`. And build Ubuntu-based image with the murano agent:

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

If you need a Fedora based image, replace ‘ubuntu’ to ‘fedora’ in the last command.

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

When you are done upload the `murano-agent.qcow2` image to glance and play :)

4.4.3 Upload image into glance

To deploy applications with murano, virtual machine images should be uploaded into glance in a special way - `murano_image_info` property should be set.

1. Use the openstack client image create command to import your disk image to glance:

```
openstack image create --public \
> --disk-format qcow2 --container-format bare \
> --file <IMAGE_FILE> --property <IMAGE_METADATA> <NAME>
```

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

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

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

where:

- **title** - user-friendly description of the image

- **type** - murano image type, see *Murano image types*

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

```
openstack image set --property <IMAGE_METADATA> <IMAGE_ID>
```

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

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

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

Note: Existing images could be marked in a simple way in the horizon UI with the murano dashboard installed. Navigate to *Murano -> Manage -> Images -> Mark Image* and fill up a form:

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

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

Murano image types

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

4.5 Murano automated tests description

This page describes automated tests for a Murano project:

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

4.5.1 Murano continuous integration service

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

Murano CI uses OpenStack QA cloud for testing infrastructure.

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

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

- “gate-murano-dashboard-selenium*” verifies each commit to murano-dashboard repository

- “gate-murano-integration*” verifies each commit to murano repository

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

4.5.2 Murano automated tests: UI tests

The murano project has a web user interface and all possible user scenarios should be tested. All UI tests are located at the <https://git.openstack.org/cgit/openstack/murano-dashboard/tree/muranodashboard/tests/functional>

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

Prerequisites:

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

Download and run tests:

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

- Clone murano-dashboard git repository:
 - `git clone git://git.openstack.org/openstack/murano-dashboard*`
- Change default settings:
 - Copy `muranodashboard/tests/functional/config/config.conf.example` to `config.conf`
 - Set appropriate urls and credentials for your OpenStack lab. Only admin users are appropriate.

[murano]

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

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

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

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

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

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

.....

Ran 34 tests in 1.440s

OK

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

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

`nosetests -h`

4.5.3 Murano Automated Tests: Tempest Tests

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

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

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

API Tests

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

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

Engine Tests

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

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

Command Line Tests

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

All tests have description and execution steps in there docstrings.

Client

4.6 Murano client

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

4.6.1 Installation

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

```
pip install python-muranoclient
```

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

4.6.2 Using CLI client

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

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

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

4.6.3 Bash completion

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

4.6.4 Listing currently installed packages

To get list of currently installed packages run:

```
murano package-list
```

To show details about specific package run:

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

4.6.5 Importing packages in Murano

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

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

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

To import a local package run:

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

To import a package from http url run:

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

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

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

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

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

For more info about specifying images and requirements for the package see package creation docs: [Step-by-Step](#).

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

4.6.6 Importing bundles of packages in Murano

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

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

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

To import a local bundle run:

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

To import a bundle from http url run:

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

To import a bundle from murano repository run:

```
murano bundle-import bundle_name
```

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

4.6.7 Deleting packages from murano

To delete a package run:

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

4.6.8 Downloading package file

Running:

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

would download the zip archive with specified package

4.6.9 Creating a package

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

```
murano help package-create
```

This command is useful, when application package files are spread across several directories, and for auto-generating packages from heat templates For more info about package composition please see package creation docs: [Step-by-Step](#).

4.6.10 Managing Environments

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

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

You can get list of deployments for environment with:

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

4.6.11 Managing Categories

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

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

4.6.12 Managing environment templates

It is possible to manage environment templates with following commands:

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

Guidelines

4.7 Contributing to Murano

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

4.7.1 Contributor License Agreement

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

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

4.7.2 Project Hosting Details

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

4.8 Development Guidelines

4.8.1 Coding Guidelines

For all the code in Murano we have a rule - it should pass [PEP 8](#).

To check your code against PEP 8 run:

```
tox -e pep8
```

See also:

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

4.8.2 Testing Guidelines

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

Unit tests are located at `murano/tests`.

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

```
tox -e py27
```

4.8.3 Documentation Guidelines

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

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

More specifically:

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

Run the following command to build docs locally.

```
tox -e docs
```

4.9 Murano Troubleshooting and Debug Tips

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

4.9.1 Problems during configuration

Log location

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

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

Possible problem list

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

4.9.2 Problems during deployment

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

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

API specification

4.10 Murano API v1 specification

4.10.1 General information

- **Introduction**

The murano service API is a programmatic interface used for interaction with murano. Other interaction mechanisms like the murano dashboard or the murano CLI should use the API as an underlying protocol for interaction.

- **Allowed HTTPs requests**

- *POST* : To create a resource
- *GET* : Get a resource or list of resources
- *DELETE* : To delete resource
- *PATCH* : To update a resource

- **Description Of Usual Server Responses**

- 200 OK - the request was successful.
- 201 Created - the request was successful and a resource was created.
- 204 No Content - the request was successful but there is no representation to return (i.e. the response is empty).
- 400 Bad Request - the request could not be understood or required parameters were missing.
- 401 Unauthorized - authentication failed or user didn't have permissions for requested operation.
- 403 Forbidden - access denied.
- 404 Not Found - resource was not found
- 405 Method Not Allowed - requested method is not supported for resource.
- 406 Not Acceptable - the requested resource is only capable of generating content not acceptable according to the Accept headers sent in the request.
- 409 Conflict - requested method resulted in a conflict with the current state of the resource.

- **Response of POSTs and PUTs**

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

- **Authentication**

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

4.10.2 Glossary

- **Environment**

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

- **Session**

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

- **Object Model**

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

- **Package**

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

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

4.10.3 Environment API

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name
created	datetime	Creation date and time in ISO format
updated	datetime	Modification date and time in ISO format
tenant_id	string	OpenStack tenant ID
version	int	Current version
networking	string	Network settings
acquired_by	string	Id of a session that acquired this environment (for example is deploying it)
status	string	Deployment status: ready, pending, deploying

Common response codes

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

List environments

Request

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

Parameters:

- *all_tenants* - boolean, indicates whether environments from all tenants are listed. *True* environments from all tenants are listed. Admin user required. *False* environments only from current tenant are listed (default like option unspecified).

Response

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


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

Create environment

Attribute	Type	Description
name	string	Environment name; at least one non-white space symbol

Request

Method	URI	Description
POST	/environments	Create new Environment

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

Response

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

Update environment

Attribute	Type	Description
name	string	Environment name; at least one non-white space symbol

Request

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

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

*Response***Content-Type** application/json

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

Code	Description
200	Edited environment
400	Environment name must contain at least one non-white space symbol
403	User is not authorized to access environment
404	Environment not found
409	Environment with specified name already exists

Get environment details*Request*

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

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

*Response***Content-Type** application/json

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

```

```

        "id": "14cce9d9-aaa1-4f09-84a9-c4bb859edaff"
      },
      {
        "name": "rewt4w56",
        "?": {
          "status": "ready",
          "_26411a1861294160833743e45d0ead9": {
            "name": "Telnet"
          },
          "type": "io.murano.apps.linux.Telnet",
          "id": "446373ef-03b5-4925-b095-6c56568fa518"
        }
      }
    ],
    "id": "20d4a012628e4073b48490a336a8acbf"
  }
}

```

Delete environment

Request

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

Parameters:

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

Response

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

4.10.4 Environment configuration API

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

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

Configure environment / open session

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

Request

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

Response

Content-Type application/json

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

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

Deploy session

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

Request

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

Response

Code	Description
200	Session status changes to <i>deploying</i>
401	User is not authorized to access this session
403	Session is already deployed or deployment is in progress
404	Not found. Specified session doesn't exist

Get session details

Request

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

Response

```
{
  "id": "4aecdc2178b9430cbbb8db44fb7ac384",
  "environment_id": "4dc8a2e8986fa8fa5bf24dc8a2e8986fa8",

```

```

    "created": "2013-11-30T03:23:42Z",
    "updated": "2013-11-30T03:23:54Z",
    "user_id": "d7b501094caf4daab08469663a9e1a2b",
    "version": 0,
    "state": "deploying"
  }

```

Code	Description
200	Session details information received
401	User is not authorized to access this session
403	Session is invalid
404	Not found. Specified session doesn't exist

Delete session

Request

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

Response

Code	Description
200	Session is deleted successfully
401	User is not authorized to access this session
403	Session is in deploying state and could not be deleted
404	Not found. Specified session doesn't exist

4.10.5 Environment deployments API

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

List Deployments

Returns information about all deployments of the specified environment.

Request

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

Response

Content-Type application/json

```

{
  "deployments": [
    {
      "updated": "2014-05-15T07:24:21",
      "environment_id": "744e44812da84e858946f5d817de4f72",
      "description": {
        "services": [
          {
            "instance": {
              "flavor": "m1.medium",

```

```
        "image": "cloud-fedora-v3",
        "?": {
            "type": "io.murano.resources.Instance",
            "id": "ef729199-c71e-4a4c-a314-0340e279add8"
        },
        "name": "xkaduhv7qeg4m7"
    },
    "name": "teslnet1",
    "?": {
        "_26411a1861294160833743e45d0eaad9": {
            "name": "Telnet"
        },
        "type": "io.murano.apps.linux.Telnet",
        "id": "6e437be2-b5bc-4263-8814-6fd57d6ddbd5"
    }
},
"defaultNetworks": {
    "environment": {
        "name": "test2-network",
        "?": {
            "type": "io.murano.lib.networks.neutron.NewNetwork",
            "id": "b6a1d515434047d5b4678a803646d556"
        }
    },
    "flat": null
},
"name": "test2",
"?": {
    "type": "io.murano.Environment",
    "id": "744e44812da84e858946f5d817de4f72"
}
},
"created": "2014-05-15T07:24:21",
"started": "2014-05-15T07:24:21",
"finished": null,
"state": "running",
"id": "327c81e0e34a4c93ad9b9052ef42b752"
}
]
}
```

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

4.10.6 Application management API

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

Get application details

Using GET requests to applications endpoint user works with list containing all applications for specified environment. A user can request a whole list, specific application, or specific attribute of a specific application using tree

traversing. To request a specific application, the user should add to endpoint part an application id, e.g.: `/environments/<env_id>/services/<application_id>`. For selection of specific attribute on application, simply appending part with attribute name will work. For example to request application name, user should use next endpoint: `/environments/<env_id>/services/<application_id>/name`

Request

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

Parameters:

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

Response

Content-Type application/json

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

POST applications

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

Request

Content-Type application/json

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

```
{
  "instance": {
    "flavor": "m1.medium",
    "image": "cloud-fedora-v3",
    "?": {
      "type": "io.murano.resources.Instance",
      "id": "bce8308e-5938-408b-a27a-0d3f0a2c52eb"
    },
  },
}
```

```
    "name": "nhekhv6r7mhd4"
  },
  "name": "sdf34sadf",
  "?": {
    "_26411a1861294160833743e45d0eaad9": {
      "name": "Telnet"
    },
    "type": "io.murano.apps.linux.Telnet",
    "id": "190c8705-5784-4782-83d7-0ab55a1449aa"
  }
}
```

Response

Created application returned

Content-Type application/json

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

Code	Description
200	Session is deleted successfully
401	User is not authorized to access this session
403	Session is in deploying state and could not be deleted
404	Not found. Specified session doesn't exist
400	Required header or body are not provided

Delete application from environment

Delete one or all applications from the environment

Request

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

Parameters:

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

4.10.7 Statistic API

Statistic API intends to provide billing feature

Instance environment statistics

Request

Get information about all deployed instances in the specified environment

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

Parameters:

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

Response

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

Content-Type application/json

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

Request

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

Response

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

Content-Type

application/json

```
[
  {
    "duration": 720,
```

```
    "count": 2,  
    "type": 200  
  }  
]
```

General Request Statistics

Request

Method	URI
GET	/stats

Response

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

Content-Type application/json

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

4.10.8 Actions API

Murano actions are simple MuranoPL methods, that can be called on deployed applications. Application contains a list with available actions. Actions may return a result.

Execute an action

Generate task with executing specified action. Input parameters may be provided.

Request

Content-Type application/json

Method	URI	Header
POST	/environments/<env_id>/actions/<action_id>	

Parameters:

- *env_id* - environment ID, required
- *actions_id* - action ID to execute, required

```
"{<action_property>: value}"
```

or

```
"{}" in case action has no properties
```

Response

Task ID that executes specified action is returned

Content-Type application/json

```
{
  "task_id": "620e883070ad40a3af566d465aa156ef"
}
```

GET action result

Request result value after action execution finish. Not all actions have return values.

Request

Method	URI	Header
GET	/environments/<env_id>/actions/<task_id>	

Parameters:

- *env_id* - environment ID, required
- *task_id* - task ID, generated on desired action execution

Response

Json, describing action result is returned. Result type and value are provided.

Content-Type application/json

```
{
  "isException": false,
  "result": ["item1", "item2"]
}
```

4.10.9 Application catalog API

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

4.10.10 Packages

Methods for application package management

Package Properties

- `id`: guid of a package (`fully_qualified_name` can also be used for some API functions)
- `fully_qualified_name`: fully qualified domain name - domain name that specifies exact application location
- `name`: user-friendly name
- `type`: package type, “library” or “application”
- `description`: text information about application
- `author`: name of application author
- `tags`: list of short names, connected with the package, which allows to search applications easily
- `categories`: list of application categories
- `class_definition`: list of class names used by a package
- `is_public`: determines whether the package is shared for other tenants
- `enabled`: determines whether the package is browsed in the Application Catalog
- `owner_id`: id of a tenant that owns the package

List packages

/v1/catalog/packages?{marker}{limit}{order_by}{type}{category}{fqn}{owned}{id}{catalog}{class_name}{name}
[GET]

This is the compound request to list and search through application catalog. If there are no search parameters all packages that is_public, enabled and belong to the user’s tenant will be listed. Default order is by ‘created’ field. For an admin role all packages are available.

Parameters

Response 200 (application/json)

```
{ "packages": [
  {
    "id": "fed57567c9fa42c192dcbe0566f8ea33",
    "fully_qualified_name" : "com.example.murano.services.linux.telnet",
    "is_public": false,
    "name": "Telnet",
    "type": "linux",
    "description": "Installs Telnet service",
    "author": "OpenStack, Inc.",
    "created": "2014-04-02T14:31:55",
    "enabled": true,
    "tags": ["linux", "telnet"],
    "categories": ["Utility"],
    "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
  },
  {
    "id": "fed57567c9fa42c192dcbe0566f8ea31",
    "fully_qualified_name": "com.example.murano.services.windows.WebServer",
    "is_public": true,
    "name": "Internet Information Services",
    "type": "windows",
    "description": "The Internet Information Service sets up an IIS server and joins it",
    "author": "OpenStack, Inc.",
```

```

        "created": "2014-04-02T14:31:55",
        "enabled": true,
        "tags": ["windows", "web"],
        "categories": ["Web"],
        "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
    }
}

```

Upload a new package[POST]

/v1/catalog/packages

See the example of multipart/form-data request, It should contain two parts - text (json string) and file object

Request (multipart/form-data)

```

Content-type: multipart/form-data, boundary=AaB03x
Content-Length: $requestlen

```

```

--AaB03x
content-disposition: form-data; name="submit-name"

--AaB03x
Content-Disposition: form-data; name="JsonString"
Content-Type: application/json

{"categories":["web"] , "tags": ["windows"], "is_public": false, "enabled": false}
`categories` - array, required
`tags` - array, optional
`name` - string, optional
`description` - string, optional
`is_public` - bool, optional
`enabled` - bool, optional

--AaB03x
content-disposition: file; name="file"; filename="test.tar"
Content-Type: targz
Content-Transfer-Encoding: binary

$binarydata
--AaB03x--

```

Response 200 (application/json)

```

{
  "updated": "2014-04-03T13:00:13",
  "description": "A domain service hosted in Windows environment by using Active Directory Role",
  "tags": ["windows"],
  "is_public": true,
  "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
  "categories": ["test1"],
  "name": "Active Directory",
  "author": "Mirantis, Inc",
  "created": "2014-04-03T13:00:13",
  "enabled": true,
  "class_definition": [
    "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "com.mirantis.murano.windows.activeDirectory.SecondaryController",
  ]
}

```

```
        "com.mirantis.murano.windows.activeDirectory.Controller",
        "com.mirantis.murano.windows.activeDirectory.PrimaryController"
    ],
    "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "type": "Application",
    "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```

Get package details

/v1/catalog/packages/{id} [GET]

Display details for a package.

Parameters

id (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/json)

```
{
  "updated": "2014-04-03T13:00:13",
  "description": "A domain service hosted in Windows environment by using Active Directory Role",
  "tags": ["windows"],
  "is_public": true,
  "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
  "categories": ["test1"],
  "name": "Active Directory",
  "author": "Mirantis, Inc",
  "created": "2014-04-03T13:00:13",
  "enabled": true,
  "class_definition": [
    "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "com.mirantis.murano.windows.activeDirectory.SecondaryController",
    "com.mirantis.murano.windows.activeDirectory.Controller",
    "com.mirantis.murano.windows.activeDirectory.PrimaryController"
  ],
  "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
  "type": "Application",
  "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```

Response 403

- In attempt to get a non-public package by a user whose tenant is not an owner of this package.

Response 404

- In case the specified package id doesn't exist.

4.10.11 Update a package

/v1/catalog/packages/{id} [PATCH]

Allows to edit mutable fields (categories, tags, name, description, is_public, enabled). See the full specification [here](#).

Parameters

id (required) Hexadecimal *id* (or fully qualified name) of the package

Content type

application/murano-packages-json-patch

Allowed operations:

```
[
  { "op": "add", "path": "/tags", "value": [ "foo", "bar" ] },
  { "op": "add", "path": "/categories", "value": [ "foo", "bar" ] },
  { "op": "remove", "path": "/tags", ["foo"] },
  { "op": "remove", "path": "/categories", ["foo"] },
  { "op": "replace", "path": "/tags", "value": [] },
  { "op": "replace", "path": "/categories", "value": ["bar"] },
  { "op": "replace", "path": "/is_public", "value": true },
  { "op": "replace", "path": "/enabled", "value": true },
  { "op": "replace", "path": "/description", "value": "New description" },
  { "op": "replace", "path": "/name", "value": "New name" }
]
```

Request 200 (application/murano-packages-json-patch)

```
[
  { "op": "add", "path": "/tags", "value": [ "windows", "directory" ] },
  { "op": "add", "path": "/categories", "value": [ "Directory" ] }
]
```

Response 200 (application/json)

```
{
  "updated": "2014-04-03T13:00:13",
  "description": "A domain service hosted in Windows environment by using Active Directory Role",
  "tags": ["windows", "directory"],
  "is_public": true,
  "id": "8f4f09bd6bcb47fb968afd29aacc0dc9",
  "categories": ["test1"],
  "name": "Active Directory",
  "author": "Mirantis, Inc",
  "created": "2014-04-03T13:00:13",
  "enabled": true,
  "class_definition": [
    "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
    "com.mirantis.murano.windows.activeDirectory.SecondaryController",
    "com.mirantis.murano.windows.activeDirectory.Controller",
    "com.mirantis.murano.windows.activeDirectory.PrimaryController"
  ],
  "fully_qualified_name": "com.mirantis.murano.windows.activeDirectory.ActiveDirectory",
  "type": "Application",
  "owner_id": "fed57567c9fa42c192dcbe0566f8ea40"
}
```

Response 403

- An attempt to update immutable fields
- An attempt to perform operation that is not allowed on the specified path
- An attempt to update non-public package by user whose tenant is not an owner of this package

Response 404

- An attempt to update package that doesn't exist

Delete application definition from the catalog

/v1/catalog/packages/{id} [DELETE]

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package to delete

Response 404

- An attempt to delete package that doesn't exist

Get application package

/v1/catalog/packages/{id}/download [GET]

Get application definition package

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/octetstream)

The sequence of bytes representing package content

Response 404

Specified package id doesn't exist

Get UI definition

/v1/catalog/packages/{id}/ui [GET]

Retrieve UI definition for a application which described in a package with provided id

Parameters

- `id` (required) Hexadecimal *id* (or fully qualified name) of the package

Response 200 (application/octet-stream)

The sequence of bytes representing UI definition

Response 404

Specified package id doesn't exist

Response 403

Specified package is not public and not owned by user tenant, performing the request

Response 404

- Specified package id doesn't exist

Get logo

Retrieve application logo which described in a package with provided id

/v1/catalog/packages/{id}/logo [GET]

Parameters

`id` (required) Hexadecimal `id` (or fully qualified name) of the package

Response 200 (application/octet-stream)

The sequence of bytes representing application logo

Response 403

Specified package is not public and not owned by user tenant, performing the request

Response 404

Specified package is not public and not owned by user tenant, performing the request

4.10.12 Categories

Provides category management. Categories are used in the Application Catalog to group application for easy browsing and search.

List categories

- `/v1/catalog/packages/categories [GET]`

!DEPRECATED (Plan to remove in L release) Retrieve list of all available application categories

Response 200 (application/json)

A list, containing category names

Content-Type application/json

```
{
  "categories": ["Web service", "Directory", "Database", "Storage"]
}
```

- `/v1/catalog/categories [GET]`

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

Retrieve list of all available application categories

Response 200 (application/json)

A list, containing detailed information about each category

Content-Type application/json

```
{
  "categories": [
    {
      "id": "0420045dce7445fabae7e5e61fff9e2f",
      "updated": "2014-12-26T13:57:04",
      "name": "Web",
      "created": "2014-12-26T13:57:04",
      "package_count": 1
    },
    {
      "id": "3dd486b1e26f40ac8f35416b63f52042",
      "updated": "2014-12-26T13:57:04",
      "name": "Databases",
      "created": "2014-12-26T13:57:04",
      "package_count": 0
    }
  ]
}
```

```
    }  
  }  
}
```

Get category details

/catalog/categories/<category_id> [GET]

Return detailed information for a provided category

Request

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

Parameters

- category_id - required, category ID, required

Response

Content-Type application/json

```
{  
  "id": "b308f7fa8a2f4a5eb419970c827f4466",  
  "updated": "2015-01-28T17:00:19",  
  "packages": [  
    {  
      "fully_qualified_name": "io.murano.apps.ZabbixServer",  
      "id": "4dfb566e69e6445fbd4aea5099fe95e9",  
      "name": "Zabbix Server"  
    }  
  ],  
  "name": "Web",  
  "created": "2015-01-28T17:00:19",  
  "package_count": 1  
}
```

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

Add new category

/catalog/categories [POST]

Add new category to the Application Catalog

Parameters

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

Request

Method	URI	Description
POST	/catalog/categories	Create new category

Content-Type application/json

Example {"name": "category_name"}

Response

```
{
  "id": "ce373a477f211e187a55404a662f968",
  "name": "category_name",
  "created": "2013-11-30T03:23:42Z",
  "updated": "2013-11-30T03:23:44Z",
  "package_count": 0
}
```

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

Delete category*/catalog/categories [DELETE]**Request*

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

Parameters:

- category_id - required, category ID, required

Response

Code	Description
200	OK. Category deleted successfully
401	User is not authorized to access this session
404	Not found. Specified category doesn't exist
403	Forbidden. Category with specified name is assigned to the package, presented in the catalog

4.10.13 Environment template API

Manage environment template definitions in murano. It is possible to create, update, delete, and deploy into OpenStack by translating it into an environment. In addition, applications can be added to or deleted from the environment template.

Environment Template Properties

Attribute	Type	Description
id	string	Unique ID
name	string	User-friendly name
created	datetime	Creation date and time in ISO format
updated	datetime	Modification date and time in ISO format
tenant_id	string	OpenStack tenant ID
version	int	Current version
networking	string	Network settings
description	string	The environment template specification

Common response codes

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

Methods for Environment Template API

List Environments Templates*Request*

Method	URI	Description
GET	/templates	Get a list of existing environment templates

Parameters:

- *is_public* - boolean, indicates whether public environment templates are listed or not. *True* public environments templates from all tenants are listed. *False* private environments templates from current tenant are listed *empty* all tenant templates plus public templates from all tenants are listed

Response

This call returns a list of environment templates. Only the basic properties are returned.

```
{
  "templates": [
    {
      "updated": "2014-05-14T13:02:54",
      "networking": {},
      "name": "test1",
      "created": "2014-05-14T13:02:46",
      "tenant_id": "726ed856965f43cc8e565bc991fa76c3",
      "version": 0,
      "is_public": false,
      "id": "2fa5ab704749444bbeafe7991b412c33"
    },
    {
      "updated": "2014-05-14T13:02:55",
      "networking": {},
      "name": "test2",
      "created": "2014-05-14T13:02:51",
      "tenant_id": "123452452345346345634563456345346",
      "version": 0,
      "is_public": true,
      "id": "744e44812da84e858946f5d817de4f72"
    }
  ]
}
```

Create environment template

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

Request

Method	URI	Description
POST	/templates	Create a new environment template

Content-Type application/json

Example {"name": "env_temp_name"}

Response

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

Code	Description
200	Operation completed successfully
401	User is not authorized to perform the operation
409	The environment template already exists

Get environment templates details

Request

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

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

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

Response

Content-Type application/json

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

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

Delete environment template

Request

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

Parameters:

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

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

Adding application to environment template

Request

Method	URI	Description
POST	/templates/{env-temp-id}/services	Create a new application

Parameters:

- *env-temp-id* - The environment-template id, required
- *payload* - the service description

Content-Type application/json

Example

```
{
  "instance": {
    "assignFloatingIp": "true",
    "keyname": "mykeyname",
    "image": "cloud-fedora-v3",
    "flavor": "m1.medium",
    "?": {
      "type": "io.murano.resources.LinuxMuranoInstance",
      "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
    }
  },
  "name": "orion",
  "port": "8080",
  "?": {
    "type": "io.murano.apps.apache.Tomcat",
    "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
  }
}
```

Response

```
{
  "instance":
  {
    "assignFloatingIp": "true",
    "keyname": "mykeyname",
    "image": "cloud-fedora-v3",
    "flavor": "m1.medium",
    "?":
    {
      "type": "io.murano.resources.LinuxMuranoInstance",
      "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
    }
  },
  "name": "orion",
  "?":
  {
    "type": "io.murano.apps.apache.Tomcat",
```

```

    "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
  },
  "port": "8080"
}

```

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

Get applications information from an environment template

Request

Method	URI Description
GET	/templates/{env-temp-id}/services It obtains the service description

Parameters:

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

Content-Type application/json

Response

```

[
  {
    "instance":
    {
      "assignFloatingIp": "true",
      "keyname": "mykeyname",
      "image": "cloud-fedora-v3",
      "flavor": "ml.medium",
      "?":
      {
        "type": "io.murano.resources.LinuxMuranoInstance",
        "id": "ef984a74-29a4-45c0-b1dc-2ab9f075732e"
      }
    },
    "name": "tomcat",
    "?":
    {
      "type": "io.murano.apps.apache.Tomcat",
      "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
    },
    "port": "8080"
  },
  {
    "instance": "ef984a74-29a4-45c0-b1dc-2ab9f075732e",
    "password": "XXX",
    "name": "mysql",
    "?":
    {
      "type": "io.murano.apps.database.MySQL",
      "id": "54cea43d-5970-4c73-b9ac-fea656f3c722"
    }
  }
]

```

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

Create an environment from an environment template

Request

Method	URI	Description
POST	/templates/{env-temp-id}/create-environment	Create an environment

Parameters:

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

Payload:

- 'environment name': The environment name to be created.

Content-Type application/json

Example

```
{
  "name": "environment_name"
}
```

Response

```
{
  "environment_id": "aa90fadfafca10e38e1c8c4bbf7",
  "name": "environment_name",
  "created": "2015-01-26T09:12:51",
  "tenant_id": "00000000000000000000000000000001",
  "version": 0,
  "session_id": "adf4dadfaa9033ca7ce245fca10e38e1c8c4bbf7",
}
```

Code	Description
200	OK. Environment template created successfully
401	User is not authorized to access this session
404	The environment template does not exist
409	The environment already exists

POST /templates/{env-temp-id}/clone

Request

Method	URI	Description
POST	/templates/{env-temp-id}/clone	It clones a public template from one tenant to another

Parameters:

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

Example Payload

```
{
  'name': 'cloned_env_template_name'
}
```

Content-Type application/json

Response

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

Code	Description
200	OK. Environment Template cloned successfully
401	User is not authorized to access this session
403	User has no access to these resources
404	The environment template does not exist
409	Conflict. The environment template name already exists

Indices and tables

- *genindex*
- *modindex*
- *search*