Iron.io + IronMQ

Application Type:	**Message Queue**
Application Version:	**1.0**
MOS Version:	**8.0**
OpenStack version:	**Liberty**
Murano package checksum:	**87a89c5383e500295295bf0e8bb07b4d**
Glance image checksum:	

Table of Contents

Document History

Version	Revision Date	Description
0.1	05-16-2016	Initial Version

1 Introduction

This document is to serve as a detailed Deployment Guide for IronMQ, a cloud native message queue solution. Iron.io offers IronMQ as a managed service or as a deployable product. This document describes the reference architecture, installation steps for certified MOS + IronMQ, limitations and testing procedures.

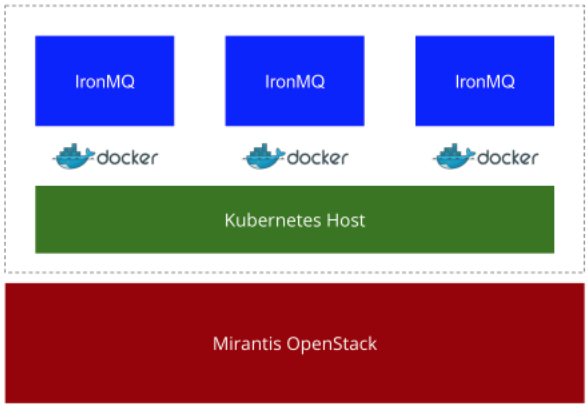
1.1 Target Audience

Developers building distributed applications on top of OpenStack need resilient ways to communicate across components and services. A common pattern is to use a message queue to mitigate the risks of data loss while in transit. Operators can deploy the IronMQ service as an application and expose the API to the developers.

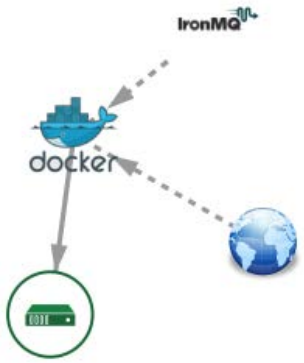
2 Application Overview

IronMQ is a cloud native message queue solution built for modern distributed applications. It provides the backbone for a microservices architecture by allowing services to communicate. Legacy message queue and ESB solutions are difficult to operate and can weigh down the entire system. IronMQ is a lightweight containerized service that communicates via HTTP through a REST API.

3 Joint Reference Architecture



4 Physical & Logical Network Topology



5 Installation & Configuration

5.1 Environment Preparation

The only requirement is that Murano be installed in order to run the IronMQ application component.

The following settings are provided as an example, as they were tested for the lab environment:

- **Environment name**: ironio
- **OpenStack Release**: Liberty on Ubuntu 14.04
- **Compute**: QEMU
- **Network**: Neutron with VLAN segmentation
- **Storage Backends**: Cinder LVM over iSCSI volumes
- **Additional Services**: Install Murano

5.2 MOS Installation

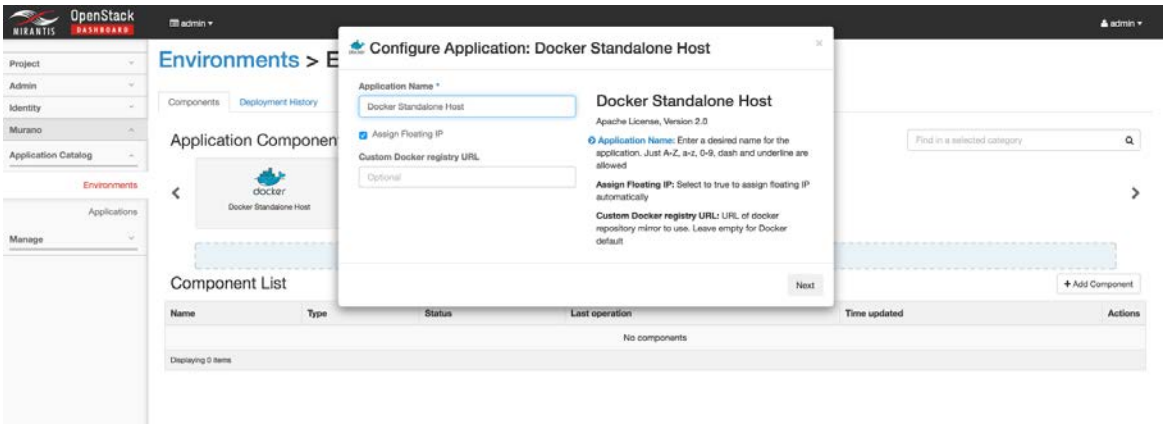
Guidelines and best practices published by Mirantis apply. Please follow Mirantis documentation on setting up a Fuel node and deploying your OpenStack environment - https://docs.mirantis.com/openstack/fuel/fuel-8.0/

5.2.1 Health Check Results

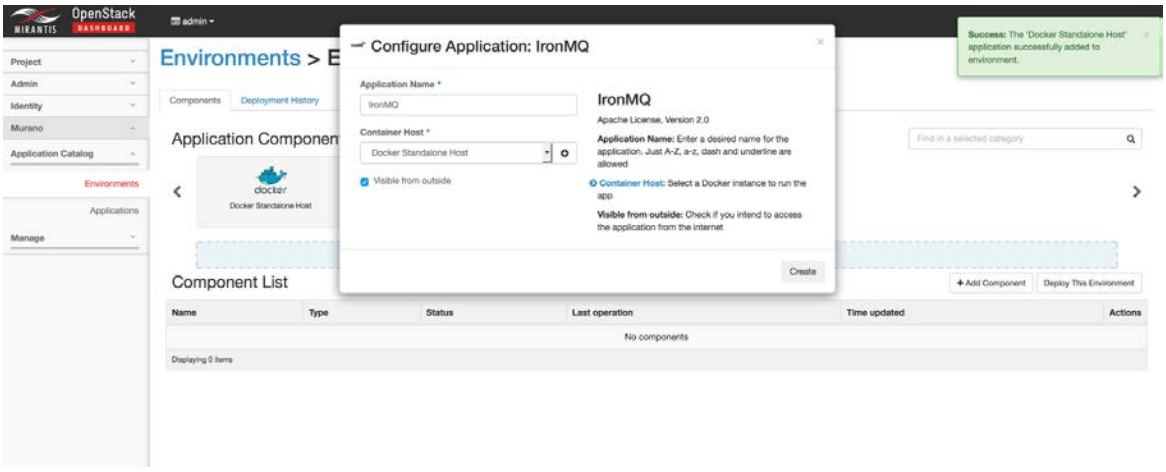
This guide assumes that all automated health checks pass following the deployment of the OpenStack environment. No additional health checks are required to validate the IronMQ Murano package.

5.3 IronMQ Installation Steps

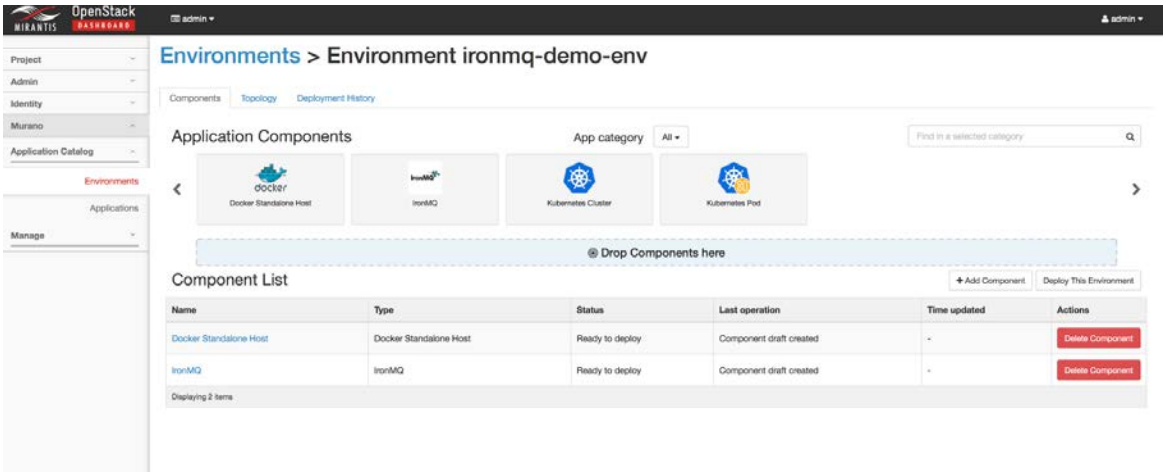
1. Add the IronMQ package from the Murano community app catalog - https://apps.openstack.org/
2. Create a Murano environment
    a. In the Horizon dashboard, navigate to *Murano* > *Application Catalog* > *Environments*
    b. On the *Environments* page, click the *Add New* button.
    c. In the *Environment Name* field, enter the name for the new environment.
    d. 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.
3. Add IronMQ component to the environment created, and follow the installation wizard
4. Configure Docker Standalone Host Component



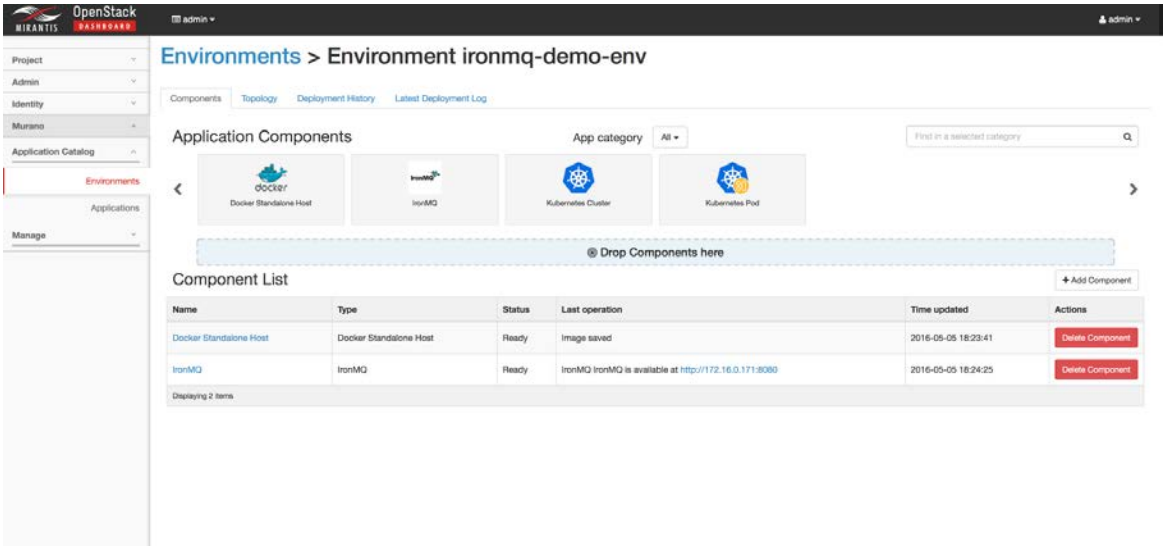
5. Click ‘Create’ to stage the application for deployment



6. Click ‘Deploy the Environment’



7. On successful deploy, note the floating ip assigned to IronMQ



5.4 Limitations

Dependencies: Docker Standalone Host

Data Storage: IronMQ persists messages until they've been completely processed. In the above setup, messages are stored ephemerally inside the applicable containers.
That means that when you shut down the iron/mq container, for example, you will lose all unprocessed messages.

If you would like to persist the data, we suggest that you create a persistent data volume container that iron/mq mounts on startup.

5.5 Testing

5.5.1 Test Cases

Testing the IronMQ installation was done manually, via the Go client library for IronMQ, which can be found here - https://github.com/iron-io/iron_go3

Links to all client libraries can be found here - http://dev.iron.io/mq/3/libraries/

Configure the client to communicate with the Murano-spawned instance for IronMQ

- Set the **Host** field to the floating IP assigned during deployment
- Set the **Scheme** to 'http'
- Set the **Port** to '8080'
- Set **ProjectId** and **Token** fields to a *non-empty string*. The actual values don't matter at this time because the IronMQ single-node deployment isn't tied to our authentication service.

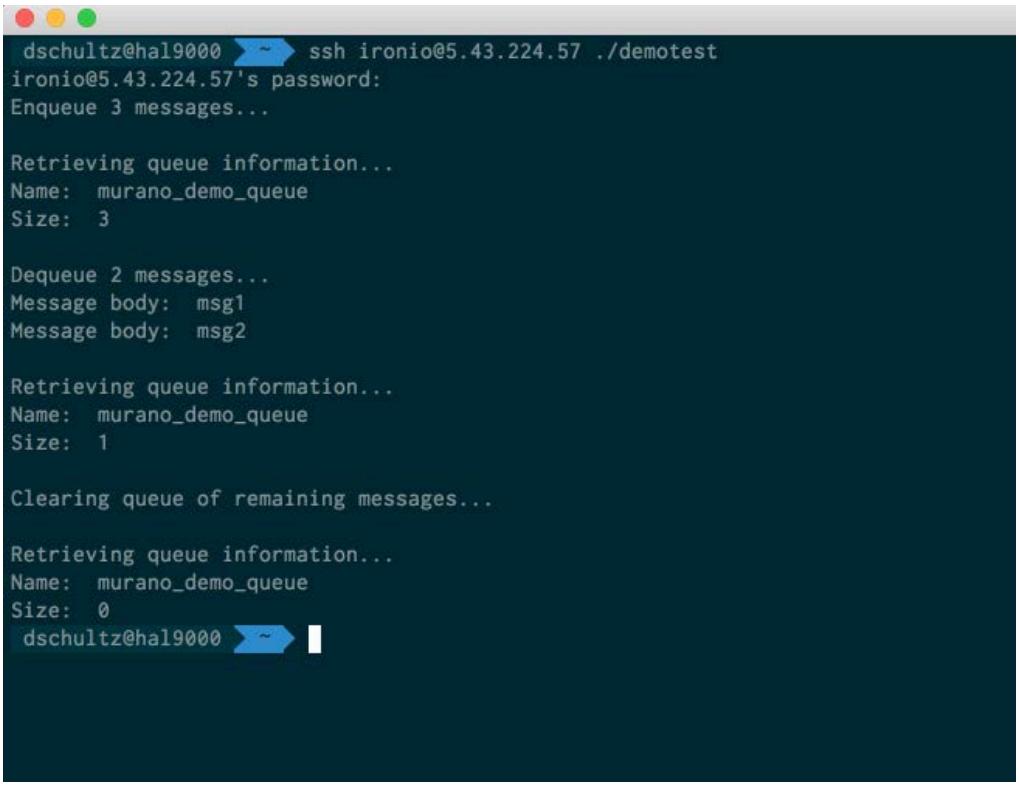
For more information, please refer to https://github.com/iron-io/iron_go3#configure

We used the client library to test the following methods:

1. Enqueue 3 messages
2. Dequeue 2 messages
3. Get queue info
4. Clear message queue of remaining messages

5.5.2 Test Results

Tests were run by executing a static Go binary on the Fuel master node, which was on the same network as the compute node hosting the IronMQ Docker application.



6 How To (Applicable for Murano packages & Glance images)

Pre-requisites:

- OS: Linux
- Mirantis OpenStack is up and running
- Murano-enabled environment
- Murano package is imported

6.1 Murano package check

1. Download Murano package
   Go to the Horizon -> Murano tab -> Package definitions
   Select ‘Download Package’ at the ‘Actions’ drop down list
2. Go to the downloaded catalog and execute the following command:
   # md5sum <package_name>
3. Add the md5 checksum to the runbook

6.2 Glance image check

1. Go to the controller node via SSH
2. Get the UUID of your glance image

 # source /root/openrc
 # glance image-list

3. Get your package's checksum
   # glance image-show <murano_image_UUID> | grep checksum
4. Add the image checksum to the title page