



INSTALLATION RUNBOOK FOR Iron.io + IronWorker

Application Type: **Job processing**

Application Version: **1.0**

MOS Version: **8.0**

OpenStack version: **Liberty**

*Murano package
checksum:* **27ba4000dc1885b830d3d370b3fb5b2**

*Glance image checksum
(docker):* **158e19b86e7532aea708267cc8092e32**

*Glance image checksum
(kubernetes):* **16e36fe2c6abac0103af1faae6203213**

Content

[Document History](#)

[1 Introduction](#)

[1.1 Target Audience](#)

[2 Application overview](#)

[3 Joint Reference Architecture](#)

[4 Physical & Logical Network Topology](#)

[5 Installation & Configuration](#)

[5.1 Environment preparation](#)

[5.2 MOS installation](#)

[5.2.1 Health Check Results](#)

[5.3 <Application name> installation steps](#)

[5.4 Limitations](#)

[5.5 Testing](#)

[5.5.1 Test cases](#)

[5.5.2 Test Results](#)

Document History

Version	Revision Date	Description
0.1	08-29-2016	Initial Version

1 Introduction

This document is to serve as a detailed Deployment Guide for Hybrid IronWorker. Iron.io offers IronWorker as a hosted service or as a Hybrid deployment. Hybrid IronWorker enables you to get all the power and benefits of IronWorker platform while running your workloads on your own hardware. You can run them on your own servers on any cloud or even in your own datacenter, behind the firewall.

This document describes the reference architecture, installation steps for Mirantis OpenStack (MOS) + Hybrid IronWorker, limitations and testing procedures

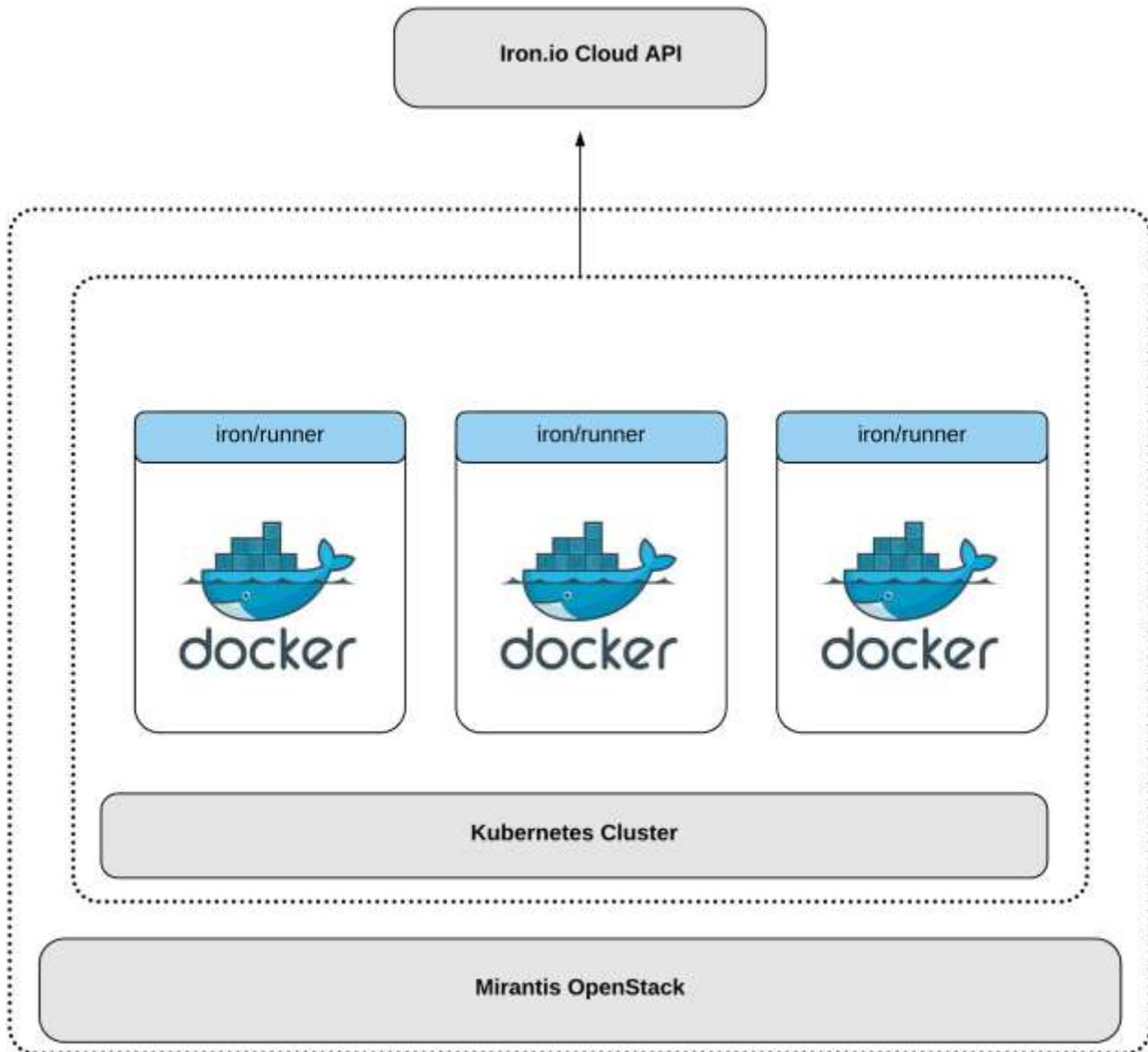
1.1 Target Audience

Developers building distributed applications on top of OpenStack need a batch of worker to process jobs. Such developers can register an account in Iron.io, deploy IronWorker on top of OpenStack on their side and post jobs in the Worker's API. Every posted job than will be executed on a deployed IronWorker instance.

2 Application overview

With Hybrid Iron.io, the API and all the complexity of our job processing system lives in the cloud, while the actual execution of the workloads is on-premise, behind your firewall, on your hardware (or in your own VPC). The only thing you need to run on your systems is our *runner* container; no databases to install and maintain, no API servers, or anything else. The *runner* container talks to the Iron.io API, asking for jobs, executing them, and dealing with all the things that can happen while running.

3 Joint Reference Architecture

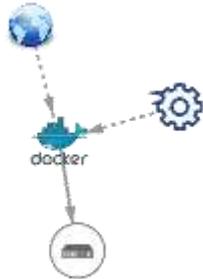


4 Physical & Logical Network Topology

The following are two supported installation methods that we'll cover in Section 5.3, steps 4-5.

4.1 Docker Standalone Host Installation

This installation method has a dependency on the Docker Murano package. A single Docker host is deployed with the IronWorker Docker image.

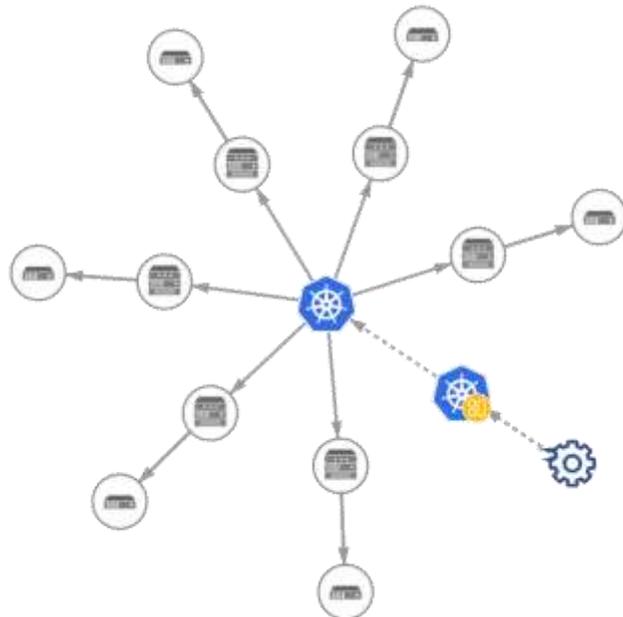


Application Components



4.2 Kubernetes Cluster Installation

This installation method has a dependency on the Kubernetes Murano package. A cluster of Kubernetes pods are deployed with the IronWorker Docker image.



Application Components



5 Installation & Configuration

5.1 Environment preparation

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

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

- 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 IronWorker Murano package.

5.3 IronWorker installation steps

1. Add the *IronWorker* package from the Murano community app catalog - <https://apps.openstack.org/#tab=murano-apps>

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 the *IronWorker* component to the environment created, and follow the installation wizard.

The screenshot shows a web form titled "Configure Application: Iron Worker". On the left side, there are several input fields with asterisks indicating they are required:

- Application Name ***: A text input field containing "IronWorker".
- Container Host ***: A dropdown menu with "Add Application" selected.
- Docker Image ***: A text input field containing "ironworker".
- API_URL ***: A text input field containing "http://worker-aws-us-east-1.iron.io".
- CLUSTER_ID ***: A text input field containing "5714adf1a6364000798664b".
- CLUSTER_TOKEN ***: A text input field containing "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbHV".
- CONCURRENCY ***: A numeric input field containing "2".
- MEMORY_PER_JOB ***: A numeric input field containing "256".

On the right side, there is a section titled "Iron Worker" with the following details:

- 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
- Container Host:** Select an instance of Docker container hosting provider to run the app
- Docker Image:** Enter docker image for runner
- API_URL:** Enter the IronWorker API URL
- CLUSTER_ID:** Enter your CLUSTER_ID
- CLUSTER_TOKEN:** Enter token to access to your cluster
- CONCURRENCY:** Number of concurrent jobs to run on each IronWorker instance.
- MEMORY_PER_JOB:** - Maximum amount of memory in MB a job will get.

At the bottom right of the form, there is a "Create" button.

Specify **CLUSTER_ID** and **CLUSTER_TOKEN** of your IronWorker cluster. Other values could remain with their defaults. To retrieve your CLUSTER_ID and CLUSTER_TOKEN please follow the Prepare the environment section of the Test Cases section of this document.

4. For IronWorker on *Docker Standalone Host*

- a. Click *Add Application* button under *Container Host* field and select *Docker Standalone Host*
- b. Configure *Docker Standalone Host* Component

Configure Application: Docker Standalone Host

Application Name *
Docker Standalone Host

Assign Floating IP

Custom Docker registry URL
Optional

Docker Standalone Host
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

Custom Docker registry URL: URL of docker repository mirror to use. Leave empty for Docker default

Next

Configure Application: Docker Standalone Host

Instance image *
Ubuntu 14.04 x64 (pre-installed Docker and mur)

Instance flavor
m1.small

Key Pair
No keypair

Availability zone
nova

Hostname
Optional

Docker Standalone Host
Specify some instance parameters on which the application would be created

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

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

Key Pair: Select a 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 the application would be installed.

Back Create

5. For IronWorker on Kubernetes
 - a. Click *Add Application* button under *Container Host* field and select *Kubernetes Pod*
 - b. Configure *Kubernetes Pod* Component

Configure Application: Kubernetes Pod

Pod Name *

Labels

Kubernetes cluster *

Replicas (0 = disabled) *

Kubernetes Pod

Apache License, Version 2.0

Pod Name: Name of the pod to create. This name must be unique throughout the cluster. The name should be up to maximum length of 253 characters and consist of lower case alphanumeric characters, hyphens, and dots.

Labels: Comma separated list of labels. Allows easy selecting in the future. Valid label keys have two segments - prefix and name - separated by a slash. The name segment is required and must be a DNS label 63 characters or less, all lowercase, beginning and ending with an alphanumeric character, with dashes and alphanumerics between. The prefix and slash are optional. If specified, the prefix must be a DNS subdomain. Valid label values must be shorter than 64 characters, accepted characters are ([A-Za-z0-9_]) but the first character must be ([A-Za-z0-9]).

Kubernetes cluster: Kubernetes service

Replicas (0 = disabled): Number of cluster Replicas. Setting to '0' prevents Replication Controller creation

c. Click *Add Application* button under *Kubernetes cluster* field press

d. Configure *Kubernetes Cluster* Component

Configure Application: Kubernetes Cluster

Cluster Name *
KubernetesCluster

Initial/current number of Kubernetes nodes *
2

Maximum number of Kubernetes nodes *
3

Assign floating IP to Kubernetes nodes

Kubernetes node hostname pattern
kube-#

Expose cAdvisor UI

Initial/current number of gateway nodes *
1

Maximum number of gateway nodes *
2

Assign floating IP to gateway nodes

Gateway hostname pattern
gateway-#

Custom Docker registry URL
Optional

Docker registry mirror URL
Optional

Google registry key
Optional

Kubernetes Cluster

Apache License, Version 2.0

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

Initial/current number of Kubernetes nodes: Select number of Kubernetes nodes

Maximum number of Kubernetes nodes: Select maximum number of Kubernetes nodes

Assign floating IP to Kubernetes nodes: Check to assign floating IP to Kubernetes nodes

Kubernetes node hostname pattern: For your convenience instance hostname can be specified. Enter a name or leave blank for random name generation.

Expose cAdvisor UI: Opens external access to cAdvisor interface

Initial/current number of gateway nodes: External traffic will be routed through gateway nodes. Increasing gateways count allows to set up complex and HA clusters.

Maximum number of gateway nodes: Maximum number of gateway nodes. Taken into account when performing scalability actions.

Assign floating IP to gateway nodes: Check to assign floating IP to gateway nodes

Gateway hostname pattern: Check to assign floating IP to gateway nodes

Custom Docker registry URL: Host IP or domain name of custom Docker registry to use. Leave empty to use Docker default.

Docker registry mirror URL: URL of Docker registry mirror to use. Leave empty to not use one.

Google registry key: Contents of JSON key file. Used to authenticate to the Google Container Registry

Next

6. Click *Deploy the Environment*

5.4 Limitations

- Only HTTP endpoints for **API_URL** are supported. HTTPS will be supported in upcoming patch.
- Make sure you have enough memory on instances you spawn. Each IronWorker instance requires $16 + \text{CONCURRENCY} * \text{MEMORY_PER_JOB}$ megabytes.

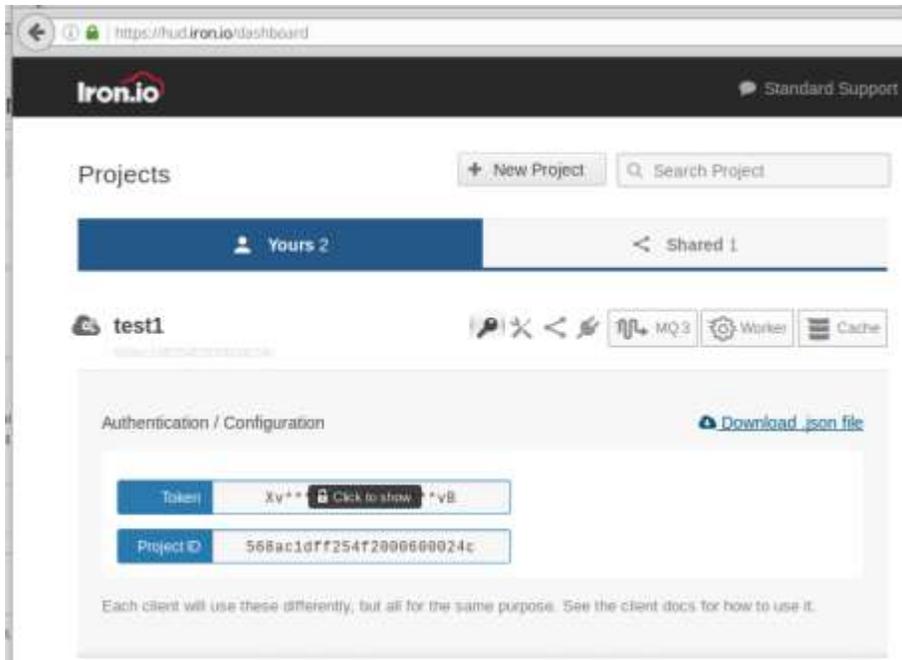
5.5 Testing

5.5.1 Test cases

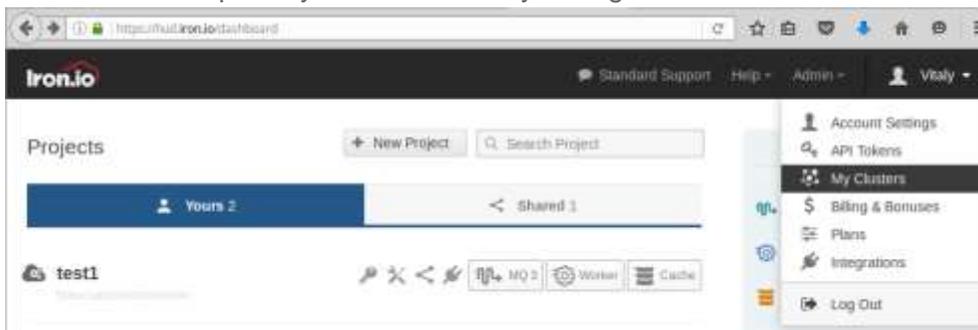
Testing the IronWorker installation was done manually. You need an Iron.io IronWorker account to pass the test case successfully.

Prepare the environment

1. Login into your Iron.io IronWorker account and collect the **PROJECT_ID** and **TOKEN** on the <https://hud.iron.io/dashboard> page (click on the *key* button aside your project)



2. Make sure you've created a cluster, if not, please create a new one
 - a. Contact Iron customer support (support@iron.io) to ensure that custom clusters are enabled for your account.
 - b. Open My Clusters under your login button



- c. Click *New Cluster* and provide a name, memory and disk space per runner, then click *Create New Cluster*

For more information about Iron CLI please visit
<http://dev.iron.io/worker/cli>

5.5.2 Test Results

Case 1 output

Note: ID's in your test run will differ

```
-----> Configuring client
          Project 'test1' with id='568ac1dff254f2000600024c'
-----> Queueing task 'hello-world'
          Queued task with id='57b33f067582400006cb1b80'
          Check
https://hud.iron.io/tq/projects/568ac1dff254f2000600024c/jobs/57b33f067582400006cb1b80 for more
info
-----> Waiting for task57b33f067582400006cb1b80
-----> Done
-----> Printing Log:
```

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker Hub account:

```
https://hub.docker.com
```

For more examples and ideas, visit:

```
https://docs.docker.com/engine/userguide/
```

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