

OpenStack Ironic API

Team Members

Sindhusha Yadavalli(201305518)

Geetika Chauhan(201306520)

Sriharsha Vogeti(201164157)

Sai Praneeth (201125083)

Contents

1. Problem Statement
2. Project Description
 - 2.1 OpenStack Description
 - 2.2 Ironic Component of OpenStack
 - 2.2.1 Background
 - 2.2.2 Introduction
 - 2.2.3 Logical Architecture
 - 2.2.4 Sub-Components of Ironic
3. Key Technologies for BM provision
4. Target Users
5. User Interaction with System
6. Interaction Diagram
7. Block Diagram of the system
8. Detailed Design of System
9. Project Approach
10. Project Implementation

1.Problem Statement

Providing baremetal provisioning(install physical server automatically) function and managing those physical servers in the same way as virtual machine Using Ironic API of OpenStack

2.Project Description

2.1 Brief Description of OpenStack:

OpenStack is a free and OSS(open-source software) cloud computing platform where users can get virtual machine whenever they want, with their required specifications.The technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a data center—which users manage through a web-based dashboard, command-line tools, or a RESTful API.The main functional components involved in this technology are as follows:

Component	Purpose
Nova	Compute
Swift	Object Storage
Cinder	Block Storage
Neutron	Networking
Keystone	Identity Service

Glance	Image Service
Horizon	DashBoard
Ceilometer	Telemetry
Heat	Orchestration
Trove	Database
Ironic	Baremetal Provisioning
Sahara	Elastic Map Reduce
Zaqar	Multiple Tenant Cloud Messaging

2.2 Ironic:

Ironic component of Openstack helps for providing “Baremetal Provisioning” (install physical server automatically) function.

2.2.1 Background

The main function of OpenStack is just providing virtual machines to users. what if a user need physical server not simply the virtual machine? and this concept of providing physical machines (as same way as virtual machines) to users is called “Baremetal Provisioning”. The component of Openstack that helps to achieve this functionality of “Baremetal Provisioning” is “Ironic”.

Advantages of Baremetal Provisioning

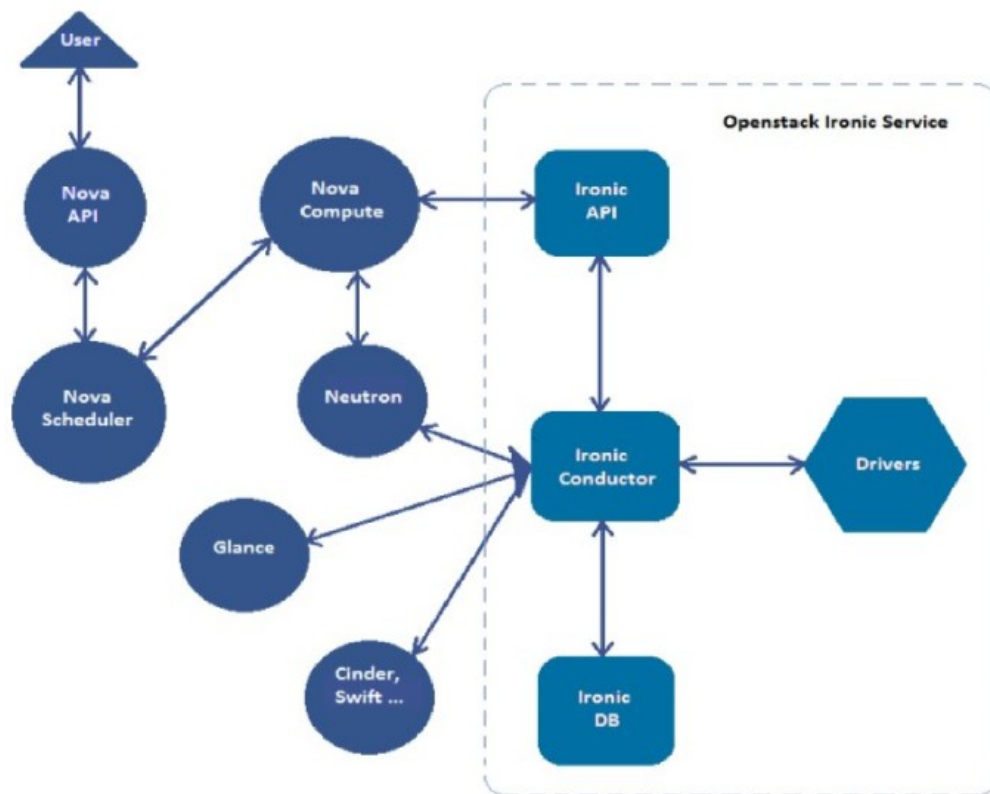
- High-performance computing clusters
- High-stress computing clusters
- For database hosting (some databases run poorly in a hypervisor) eg: social games
- Customers hesitating about using virtual machines for production system can go with Baremetal Provisioning.
- Can handle Computing tasks that require access to hardware devices which can't be virtualized .
- Can copy many Physical Servers

2.2.2 Introduction

Ironic is an OpenStack project which provisions physical hardware as opposed to virtual machines. Ironic provides several reference drivers which leverage common technologies like PXE and IPMI, to cover a wide range of hardware. Ironic’s pluggable driver architecture also allows vendor-specific drivers to be added for improved performance or functionality not provided by reference drivers.

If one thinks of traditional hypervisor functionality (e.g., creating a VM, enumerating virtual devices, managing the power state, loading an OS onto the VM, and so on), then Ironic may be thought of as a hypervisor API gluing together multiple drivers, each of which implement some portion of that functionality with respect to physical hardware.

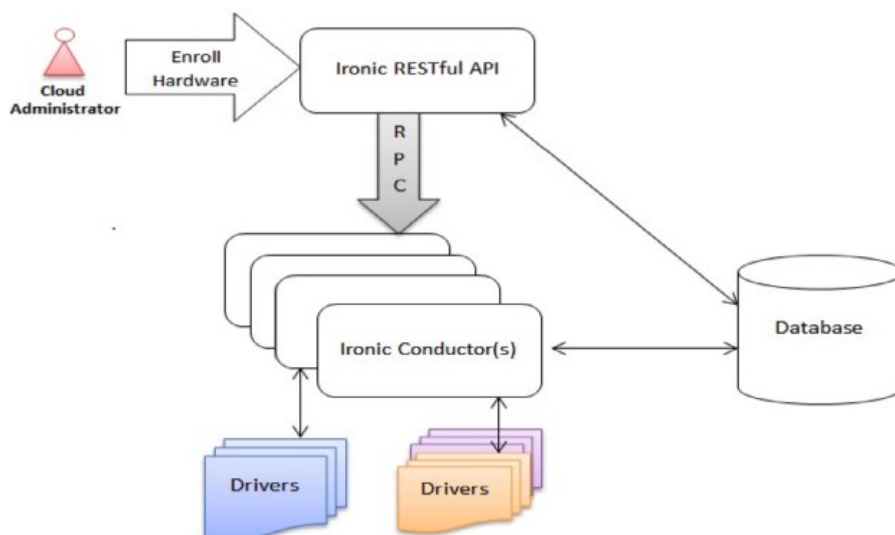
2.2.3 Logical Architecture



2.2.4 Sub-Components of Ironi Component of OpenStack

The Ironi service is composed of the following components:

- A RESTful API service, by which operators and other services may interact with the managed bare metal servers.
- A Conductor service, which does the bulk of the work. Functionality is exposed via the API service. The Conductor and API services communicate via RPC.
- A Message Queue
- A Database for storing the state of the Conductor and Drivers.



3.Key Technologies for BM Provision

PXE : Preboot Execution Environment is part of Wired For Management (WfM) specification developed by Intel and Microsoft. Booting computer via a network.

NBP : Network Boot Strap Program is equivalent to GRUB (GRand Unified Bootloader) or LILO (Linux Loader) – loaders which are traditionally used in local booting. Like the boot program in hard drive environment, the NBP is responsible for loading the OS kernel into memory so that OS can be bootstrapped over a network.

IPMI : Intelligent Platform Management Interface is a standardized computer system interface used by system administrators for out-of-band management of computer systems and monitoring of their operation.

DHCP : Using PXE, the BIOS uses DHCP to obtain an IP address for the network interface and to locate the server that stores NBP.

TFTP : Trivial File Transfer Protocol is a simple file transfer protocol that is generally used for automated transfer of configuration or boot files between machines in a local environment. In a PXE environment, TFTP is used to download NBP over network using information from DHCP server.

iSCSI : Internet Small Computer System Interface, an Internet Protocol (IP)-based storage networking standard for linking data storage facilities. By carrying SCSI commands over IP networks, iSCSI is used to facilitate data transfers over intranets and to manage storage over long distances.

4.Target User

- The target users for this project are the cloud service providers who offer cloud services to clients and enterprises to build their private cloud systems.
- It can be used by the people who want traditional hypervisor functionality like creating a VM, enumerating virtual devices, managing the power state, loading an OS onto the VM etc.
- It can also be used by Service Providers and enterprises as it makes provision of physical servers as easy as virtual machines in cloud, which in turn will open up new avenues.

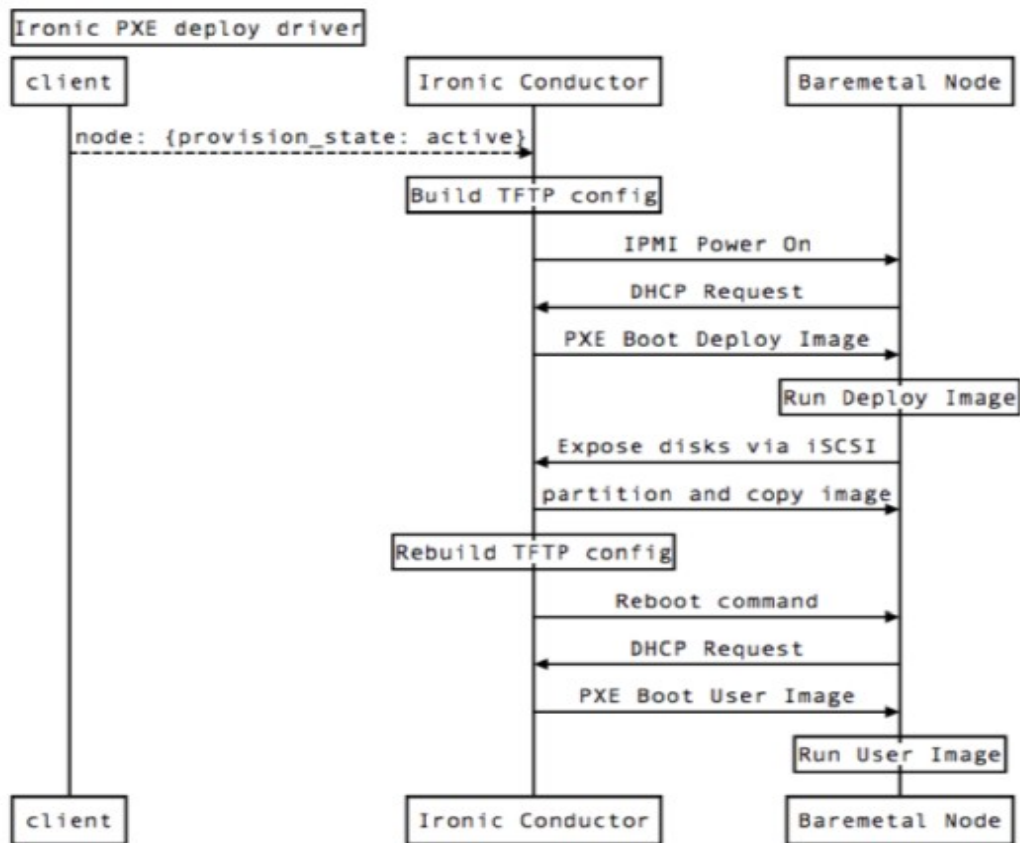
5.User interaction with the system

User interacts using commands provided by REST API
Ironic Commands

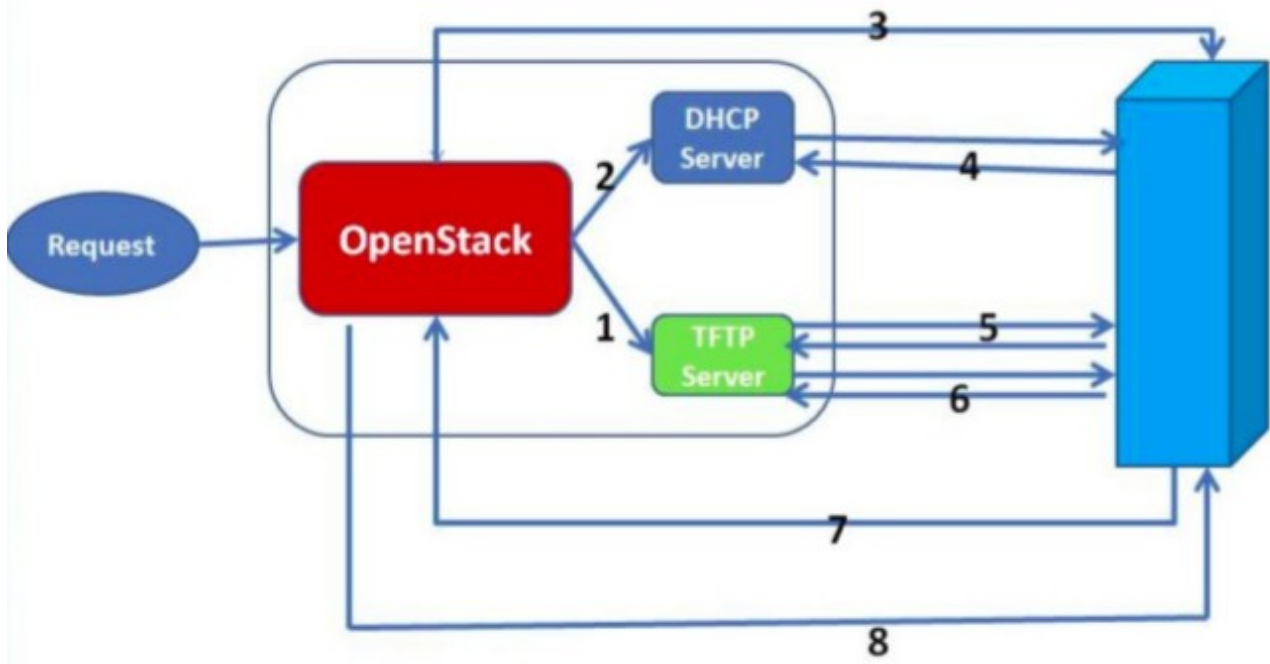
- node-create - Create a new node.
- node-delete - Delete a node.
- node-get-console - return the connection information about the console.
- node-list - list nodes.
- node-set-console-mode - Enable or disable the console access.
- node-set-power-state - Power the node on or off.
- node-set-provision-state - Provision or tear down a node.
- node-show - Show a node.
- node-update - Update a node.
- node-validate - Validate the node driver interfaces.
- port-create - Create a new port.

- port-delete - Delete a port.
- port-list - List ports.
- port-show - Show a port.
- port-update - Update a port.
- chassis-create - Create a new chassis.
- chassis-delete - Delete a chassis.
- chassis-list - List chassis.
- chassis-node-list - List the nodes contained in the chassis.
- chassis-show - Show a chassis.
- chassis-update - Update a chassis.
- driver-list - List drivers.
- driver-show - Show a driver.

6.Interaction Diagram



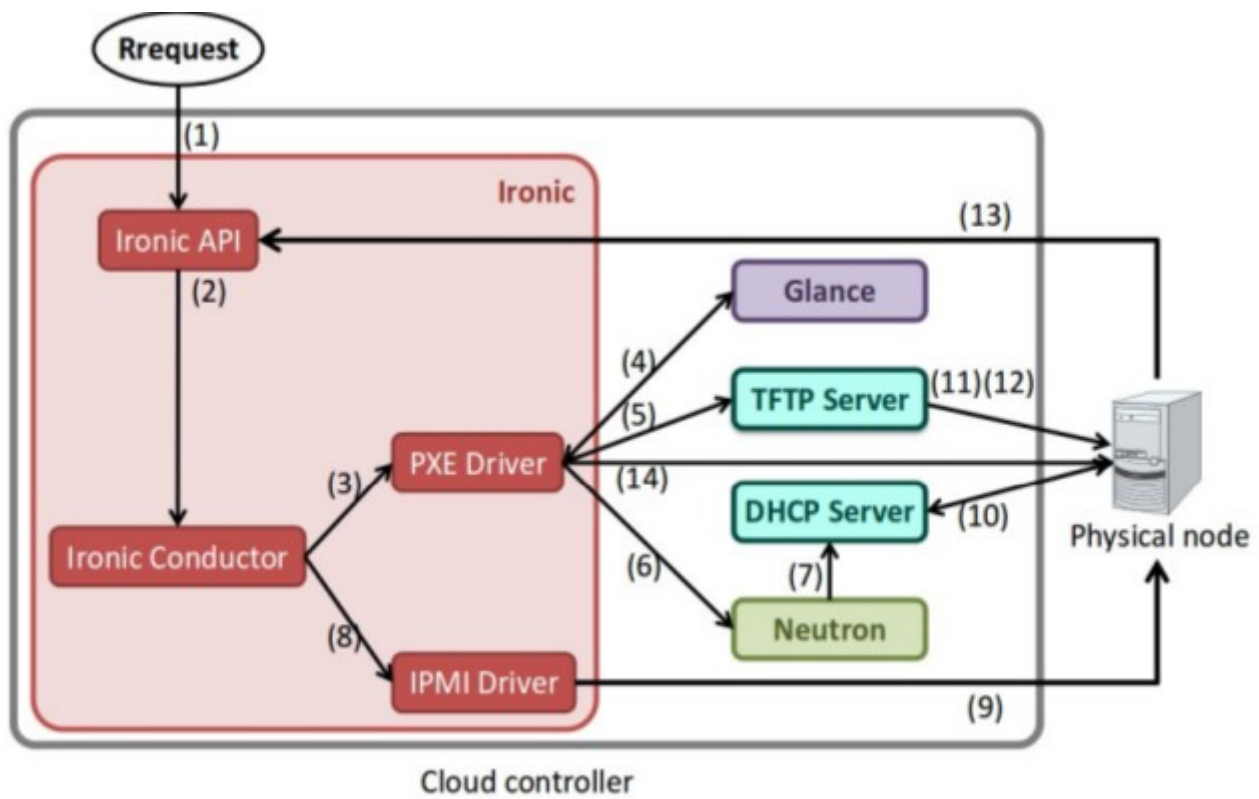
7. Block Diagram of the system



Explanation (Baremetal provisioning method) :

1. OpenStack Stores image to TFTP server
2. OpenStack sets DHCP configurations
3. OpenStack powers on physical node
4. Booted physical node unicast DHCP request
5. Physical node unicast bootloader (pxelinux.0) request to the TFTP server which DHCP server informed to physical node
6. Physical node boots from bootloader which was gotten in(5), then unicast PXE configuration file request.
7. Physical node posts its own iSCSI connection information(e.g. IP address, port) to OpenStack API.
8. OpenStack performs iSCSI connection to physical node, then creates partition, copies image file. Finally, OpenStack reboots physical node.

8.Detailed Design



Explanation:

1. Sends request of creating physical node to Ironical API (Usually from Nova)
2. Ironical API sends request of creating physical node to Ironical Conductor
3. Ironical Conductor sends request of creating physical node to PXE Driver
4. PXE Driver requests image from Glance
5. PXE Driver copies the image which is gotten from Glance to TFTP server
6. PXE Driver requests setting DHCP configuration from Neutron
7. Neutron sets DHCP configurations.
8. Ironical Conductor sends request of power on the physical node to IPMI Driver
9. IPMI Driver powers on physical node
10. Booted physical node unicasts DHCP request
11. Physical node unicasts bootloader (pxelinux.0) request to the TFTP server which DHCP server informs to physical node
12. Physical node boots from bootloader which was gotten in (11), then unicasts PXE configuration file request.
13. Physical node posts its own iSCSI connection information (e.g. IP address, port) to OpenStack API
14. OpenStack performs iSCSI connection to physical node, then creates partition, copies image file. Finally, OpenStack reboots physical node

9.Project Approach

It is clear from the above explanation to create BareMetal we need to have a System of IPMI Support . Because of lack of IPMI supported system, here we are going to create fake baremetal nodes for ironic using ssh instead of using IPMI.

10.Project Implementation

Step-1:

clone the repo using command:

```
git clone https://github.com/openstack-dev/devstack.git devstack
```

Step-2:

create local.conf in devstack/local.conf and copy the following contents:

```
[[local|localrc]]
```

```
# Credentials
```

```
DATABASE_PASSWORD=secrete
```

```
ADMIN_PASSWORD=secrete
```

```
SERVICE_PASSWORD=secrete
```

```
SERVICE_TOKEN=secrete
```

```
RABBIT_PASSWORD=secrete
```

```
# Enable Ironic API and Ironic Conductor
```

```
enable_service ironic
```

```
enable_service ir-api
```

```
enable_service ir-cond
```

```
# Enable Neutron which is required by Ironic and disable nova-network.
```

```
ENABLED_SERVICES=rabbit,mysql,key
```

```
ENABLED_SERVICES+=,n-api,n-crt,n-obj,n-cpu,n-cond,n-sch,n-novnc,n-cauth
```

```
ENABLED_SERVICES+=,neutron,q-svc,q-agt,q-dhcp,q-l3,q-meta,q-lbaas
```

```
ENABLED_SERVICES+=,g-api,g-reg
```

```
ENABLED_SERVICES+=,cinder,c-api,c-vol,c-sch,c-bak
```

```
ENABLED_SERVICES+=,ironic,ir-api,ir-cond
```

```
ENABLED_SERVICES+=,heat,h-api,h-api-cfn,h-api-cw,h-eng
```

```
ENABLED_SERVICES+=,horizon
```

```
# Create 3 virtual machines to pose as Ironic's baremetal nodes.
```

```
IRONIC_VM_COUNT=3
```

```
IRONIC_VM_SSH_PORT=22
```

```
IRONIC_BAREMETAL_BASIC_OPS=True
```

```
# The parameters below represent the minimum possible values to create
```

```
# functional nodes.
```

```
IRONIC_VM_SPECS_RAM=1024
```

```
IRONIC_VM_SPECS_DISK=10
```



```
# Size of the ephemeral partition in GB. Use 0 for no ephemeral partition.
IRONIC_VM_EPHEMERAL_DISK=0
VIRT_DRIVER=ironic
# By default, DevStack creates a 10.0.0.0/24 network for instances.
# If this overlaps with the hosts network, you may adjust with the
# following.
NETWORK_GATEWAY=10.1.0.1
FIXED_RANGE=10.1.0.0/24
FIXED_NETWORK_SIZE=256
# Log all output to files
LOGFILE=$HOME/devstack.log
SCREEN_LOGDIR=$HOME/logs
IRONIC_VM_LOG_DIR=$HOME/ironic-bm-logs
```

Step-3:

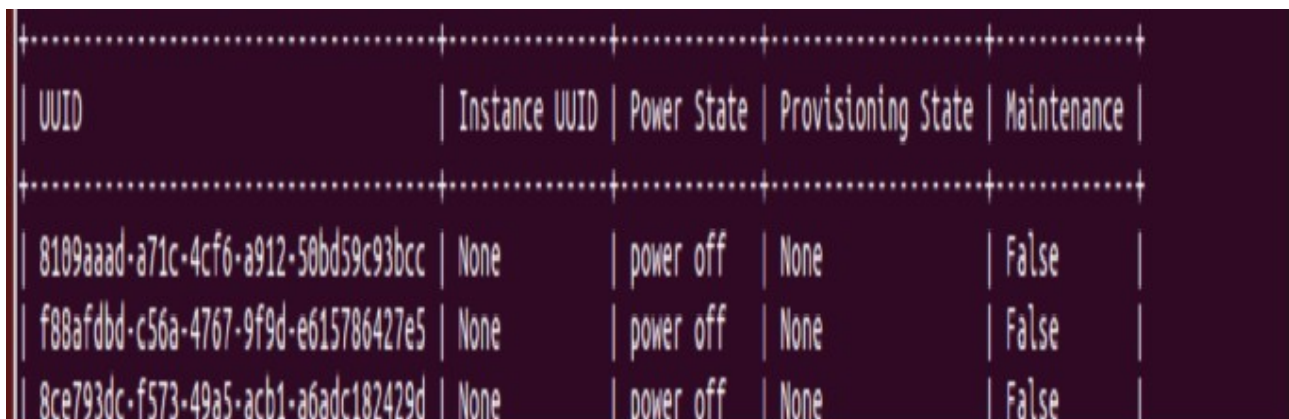
Now your devstack is ready for setup, simply run
./stack.sh

Step-4:

After devstack finishes running, you should see something similar to:
Horizon is now available at <http://10.0.0.1/>
Keystone is serving at <http://10.0.0.1:5000/v2.0/>
Examples on using novaclient command line is in exercise.sh
The default users are: admin and demo
The password: secrete
This is your host ip: 10.0.0.1

Step-5:

Created fake-nodes can be observed using command:
ironic node-list



UUID	Instance UUID	Power State	Provisioning State	Maintenance
8109aaad-a71c-4cf6-a912-50bd59c93bcc	None	power off	None	False
f88afdbd-c56a-4767-9f9d-e615786427e5	None	power off	None	False
8ce793dc-f573-49a5-acb1-a6adc182429d	None	power off	None	False

With all power states as “power-off” since no instance was created

Step-6:

nova boot --nic net-id=<net-id> --flavor baremetal --image <image-id> --key_name test2 my-first

Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	-
OS-EXT-SRV-ATTR:hypervisor_hostname	-
OS-EXT-SRV-ATTR:instance_name	instance-00000005
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
adminPass	ziY9VATr2V7z
config_drive	
created	2014-11-29T20:47:42Z
flavor	flavor2 (c4d10009-f6ed-4400-ba88-3314372ebf0c)
hostid	
id	70c85b39-2ead-40b7-a31f-9c59c02c0194
image	Fedora-x86_64-20-20140618-sda (79e5d937-0d3e-48b1-a670-e6cbf6ebdc29)
key_name	test2
metadata	{}
name	my-second
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	BUILD
tenant_id	9759b2f940c047068b085d70deb5a4fe
updated	2014-11-29T20:47:42Z
user_id	2060c176c3094965ae61d5584b8074cd

Step-7:

Now ironic nodes will go through the following stages:

(i)

UUID	Instance UUID	Power State	Provisioning State	Maintenance
101cf153-87fa-4c9f-a16d-4d1785f628ff	1921bf7d-8374-480f-9acb-63abcbd68ee9	power off	deploying	False
18b93bc3-2a00-4f91-954e-1db901e6517a	None	power off	None	False
7a92313b-d1ce-4af7-bdc7-c7d90ed32a98	None	power off	None	False

(ii)

UUID	Instance UUID	Power State	Provisioning State	Maintenance
101cf153-87fa-4c9f-a16d-4d1785f628ff	1921bf7d-8374-480f-9acb-63abcbd68ee9	power on	wait call-back	False
18b93bc3-2a00-4f91-954e-1db901e6517a	None	power off	None	False
7a92313b-d1ce-4af7-bdc7-c7d90ed32a98	None	power off	None	False

(iii)

UUID	Instance UUID	Power State	Provisioning State	Maintenance
101cf153-87fa-4c9f-a16d-4d1785f628ff	90f4310d-1c3a-466b-809a-82e2ac8d70c9	power on	active	False
18b93bc3-2a00-4f91-954e-1db901e6517a	None	power off	None	False
7a92313b-d1ce-4af7-bdc7-c7d90ed32a98	None	power off	None	False

Step-8:

And now we can see the instance is Running with command:

nova list

ID	Name	Status	Task State	Power State	Networks
90f4310d-1c3a-466b-809a-82e2ac8d70c9	my-first	ACTIVE	-	Running	private=10.1.0.9

sindhusha@sindhusha-Inspiron-N4050:~/devstack\$

Step-9:

ssh to created instance from external network using “Floating IP”

(i)check available floating-ip using command:

nova floating-ip-list

(ii)create floating-ip:

nova floating-ip-create

(iii)Assign created floating-ip to instance

nova add floating-ip my-first 172.24.4.3

```
@sindhusha-Inspiron-N4050: ~/devstack
```

sindhusha@sindhusha-Inspiron-N4050:~/devstack\$ nova list					
ID	Name	Status	Task State	Power State	Networks
729fbdee-8e43-4ad4-9e91-00752d44a2b8	my-first	ACTIVE	-	Running	private=10.1.0.4, 172.24.4.3

Now with that floating-ip can ping from external network
and Can ssh to that system using:

ssh -i test.pem ciros@172.24.4.3

Step-10:

If in case want to create ironic node on other system, we need to create ironic node by ssh to required system. Which involves following steps:

1.Chassis Create

ironic chassis-create

2.node-create

```
ironic node-create --chassis_uuid 98834cf1-6ed3-4fd2-98c8-f1c98184bd54 --driver pxe_ssh -i  
pxe_deploy_kernel=1a229813-bb82-4a0c-8057-47f7ed559a4f -i pxe_deploy_ramdisk=c866ad86-  
9846-41b5-9766-4089788b5950 -i ssh_virt_type=virsh -i ssh_address=10.1.98.197 -i ssh_port=22  
-i ssh_username="sindhusha" -p cpus=1 -p memory_mb=1024 -p local_gb=20 -p  
cpu_arch=x86_64
```

3.port-create

```
ironic port-create --address 56:84:7a:fe:97:99 --node_uuid 0bdabd94-7667-4b19-8de4-  
27fe5cf0d516
```

4.Instance info updation

```
ironic node-update 72ba92fb-1afa-496e-ae95-15ba3f3a3c8f add  
instance_info/image_source=54514ec1-a95a-4f12-ab7b-d6d9abf15f33  
ironic node-update 72ba92fb-1afa-496e-ae95-15ba3f3a3c8f add instance_info/root_gb=1
```