목차

Ironic Baremetal Service 3
소프트웨어 아키텍처 설계 4
자원 모델 설계 6
전체 수명 주기의 상태 기계 설계 6
Inspection 베어메탈 자가 점검 단계 7
Provision 베어메탈 배포 단계 8
Clean 베어메탈 회수 단계 8
Ironic(Stein) 빠른 설치 (devstack) 9
Ironic 수동 구성 16
Provisioning 네트워크 구성 16
Cleaning 네트워크 구성 17
Deploy 이미지와 User 이미지 구축 17
Ironic Node 생성 19
Ironic Conductor의 해시 매핑 22
Ironic 드라이버 모델 24
Ironic Python Agent(IPA) 25
Ironic Console 26
Ironic의 이미지 26
참조링크 27

Ironic Baremetal Service

Ironic은 가상 머신이 아닌 베어메탈 머신을 프로비저닝하는 OpenStack 프로젝트입니다. 독립적으로 사용될 수도 있고 OpenStack Cloud의 일부로 사용될 수도 있으며, OpenStack Identity(keystone), Compute(nova), Network(neutron), Image(glance), 그리고 Object(swift) 서비스와 통합됩니다.

베어메탈 서비스는 일반적으로 사용되는 PXE와 IPMI와 같은 프로토콜과 벤더별 원격 관리 프로토콜을 통해 하드웨어를 관리합니다. 이 서비스는 클라우드 운영자에게 다양한 서버들을 통합된 인터페이스로 관리할 수 있는 기능을 제공하며, Compute 서비스에는 물리 서버를 가상 머신처럼 관리할 수 있는 인터페이스를 제공합니다. 원문: https://docs.openstack.org/ironic/latest/

베어메탈 노드는 운영 체제가 배포되지 않은 물리 서버를 의미합니다. 가상 머신과 비교했을 때, 베어메탈 노드는 더 강력한 계산 능력, 자원 독점, 그리고 보안 격리 등의 장점을 가지고 있습니다. Ironic은 사용자에게 셀프 서비스 방식의 베어메탈 관리 서비스를 제공하는 것을 목표로 하며, 독립적으로 사용할 수도 있고 OpenStack과 통합하여 사용할 수도 있습니다. 우리는 후자인 OpenStack과의 통합에 주로 집중합니다.

- `Ironic Standalone(Bifrost)`: Ironic 자체로 독립적이고 경량화된 베어메탈 관리 플랫폼을 구축합니다.
- `OpenStack`과의 통합(`Nova`): Ironic은 Nova 드라이버로서 다중 테넌트 베어메탈 클라우드를 제공합니다.

참고: `Bifrost`는 `Standalone Ironic`을 자동으로 배포하는 Ansible 플레이북 모음입니다.

Ironic은 OpenStack에 베어메탈 관리 서비스를 제공하여 사용자가 가상 머신을 관리하듯 베어메탈 노드를 관리할 수 있게 해줍니다. 베어메탈 배포는 가상 머신 배포만큼 간단하며, 사용자에게 다중 테넌트 네트워크의 베어메탈 클라우드 인프라를 제공합니다. Ironic은 주로 PXE와 IPMI 기술을 사용하여 베어메탈 노드의 대량 배포와 시스템 제어를 수행하므로, 대부분의 물리 서버 모델은 Ironic을 통해 시스템 설치와 전원 상태 관리를 할 수 있습니다. 특정 물리 서버 모델의 경우에도 Ironic의 플러그인 가능한 드라이버 아키텍처를 기반으로 맞춤형 관리 드라이버를 빠르게 개발할 수 있습니다. 표준 API, 광범위한 드라이버 지원, 경량의 공간 점유 덕분에 Ironic은 소규모 엣지 배포부터 대규모 데이터 센터까지 다양한 사용 사례에 적합하며, 고성능 클라우드 애플리케이션 및 아키텍처(예: Kubernetes와 같은 컨테이너 오케스트레이션 플랫폼)를 호스팅하기에 이상적인 실행 환경을 제공합니다.

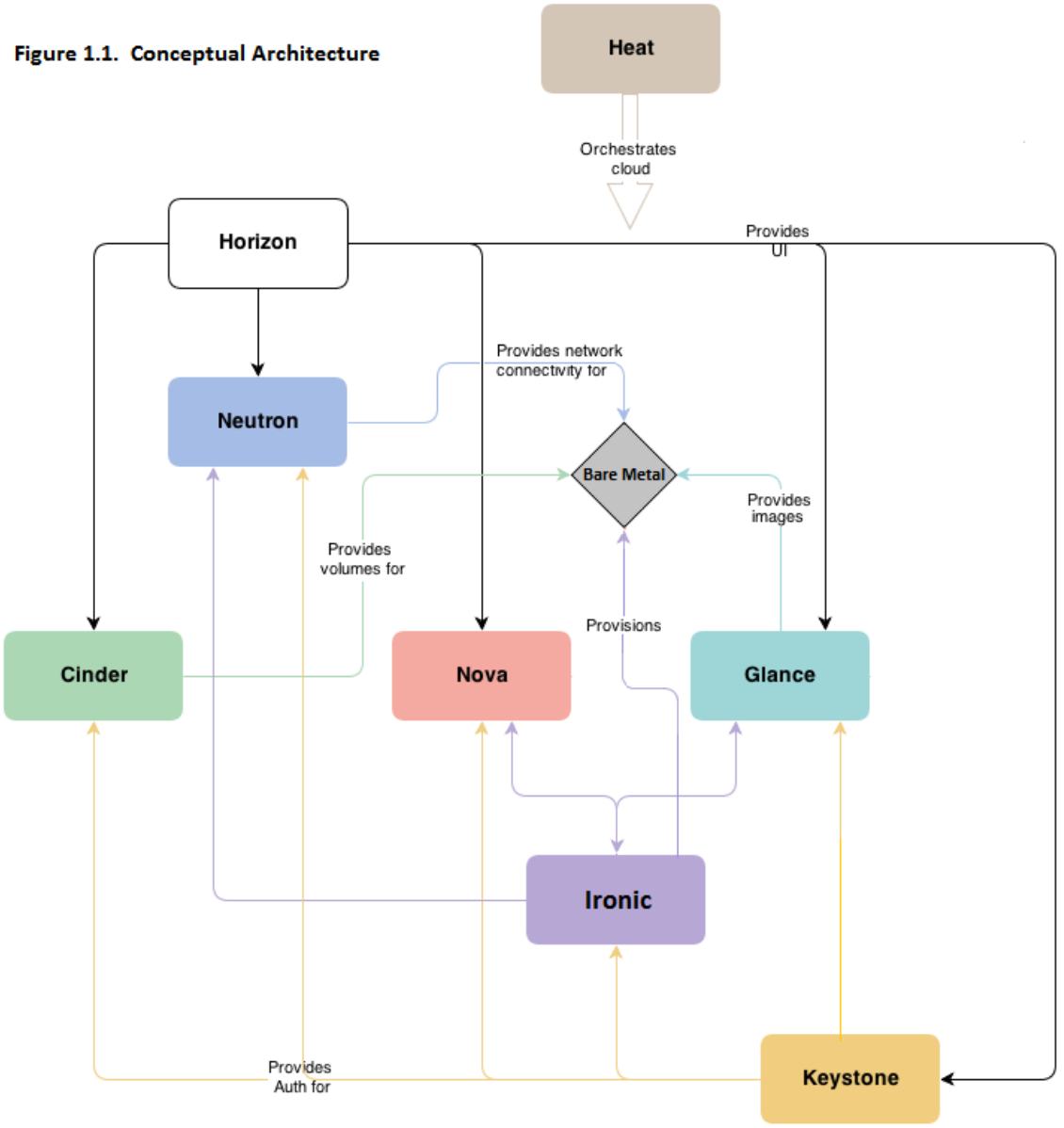
Ironic을 적용하여 해결할 수 있는 문제:

1. **베어메탈 자원의 셀프 서비스**: 사용자에게 자율적으로 베어메탈 자원을 관리할 수 있는 셀프 서비스 방식의 베어메탈 자원 서비스를 제공합니다.
2. **OpenStack 자동화 배포**: 베어메탈 설치와 설정의 자동화를 포함한 진정한 의미의 완전 자동화된 배포를 실현합니다.

Ironic과 다른 OpenStack 프로젝트의 협업:

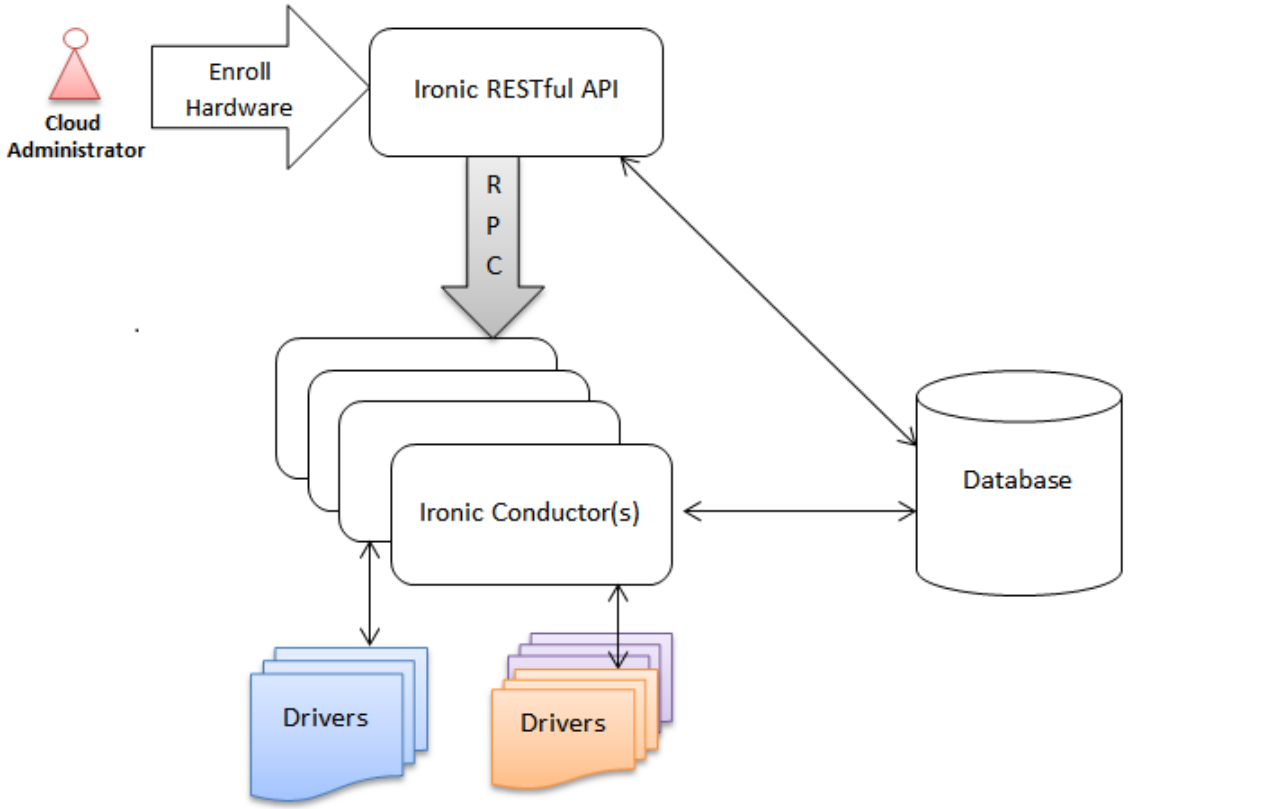
1. **Keystone**: 인증 및 권한 부여 서비스를 제공합니다.
2. **Glance**: 이미지 및 메타데이터 등록 서비스를 제공합니다.
3. **Nova**: Ironic의 스케줄링 및 관리 추상화 계층으로 작동합니다.
4. **Ironic**: 베어메탈 관리 서비스를 제공합니다.
5. **Neutron**: 다중 테넌트 네트워크 서비스를 제공합니다.
6. **Cinder**: 블록 스토리지 서비스를 제공합니다.
7. **Ceilometer**: IPMI 메트릭을 수집하는 OpenStack 텔레메트리 모듈입니다.
8. **Swift**: 구성 드라이브, 사용자 이미지, 배포 로그 및 검사 데이터를 위한 임시 저장소를 제공합니

다.



https://is-cloud.blog.csdn.net

소프트웨어 아키텍처 설계



https://is-cloud.blog.csdn.net

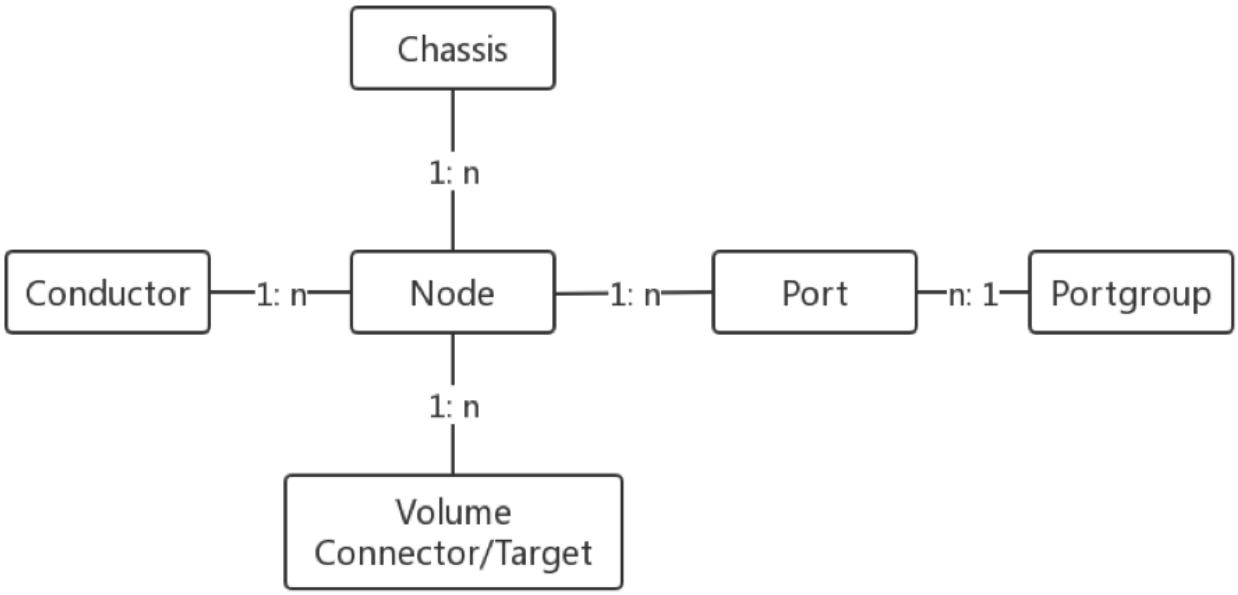
1. `Ironic API`: 북향 RESTful API를 제공합니다.
2. `Ironic Conductor`: 베어메탈 관리 서비스의 대부분 작업을 수행합니다. 예를 들어, 베어메탈 추가, 편집, 삭제, 전원 켜기/끄기, 프로비저닝, 배포, 정리 등이 포함됩니다.
3. `Drivers`: 다양한 베어메탈 모델에 맞춘 상호작용 모듈로, 각 Ironic Conductor 인스턴스는 각 모델에 맞는 드라이버 목록을 통해 베어메탈 노드를 제어합니다.
4. `Ironic Inspector`: “ 베어메탈 자가 점검 모듈 ” 로, 베어메탈 특성 정보를 자동으로 기록하며 수동 입력을 최소화합니다.
5. `Ironic Python Agent(IPA)`: RAMDisk에서 실행되는 Python 서비스로, RESTful API를 제공하여 Ironic Inspector 및 Ironic Conductor의 원격 접근을 허용하고, 내부 베어메탈의 자가 점검 및 작업을 수행합니다.
6. `RAMDisk`: “ 메모리 디스크 ” 로, 운영 체제가 설치되지 않은 베어메탈에서 Ironic Python Agent 프로그램을 실행할 수 있는 환경을 제공합니다. RAMDisk는 배포 이미지 유형에 속하며, diskimage-builder 도구로 구축됩니다.
7. `diskimage-builder`: 배포 이미지와 사용자 이미지를 제작하는 도구입니다.
8. `python-ironicclient`: Ironic CLI입니다.
9. `ironic-ui`: Horizon 대시보드 플러그인입니다.
10. `DB`: 데이터베이스입니다.
11. `MQ`: 메시지 큐입니다.

참고: 네트워크 관리는 외부 관리(out-of-band)와 내부 관리(in-band) 두 가지 방식으로 나뉩니다.

- **외부 관리(out-of-band)**: 네트워크의 관리 제어 정보와 사용자 네트워크의 데이터가 서로 다른 논리 채널을 통해 전송됩니다. 독립된 관리 채널을 사용하여 장비를 유지보수하며, 장비가 켜져 있든 꺼져 있든 시스템 관리자가 원격으로 서버와 네트워크 장비를 모니터링하고 관리할 수 있습니다.
- **내부 관리(in-band)**: 네트워크의 관리 제어 정보와 사용자 네트워크의 데이터가 동일한 논리 채널을 통해 전송됩니다. 일반 데이터 채널(예: 이더넷)을 통해 장비를 관리합니다. 내부 관리의 주요 한계는, 관리 장비가 공격받거나 손상될 경우 관리가 어려워진다는 점입니다. 네트워크 장애가

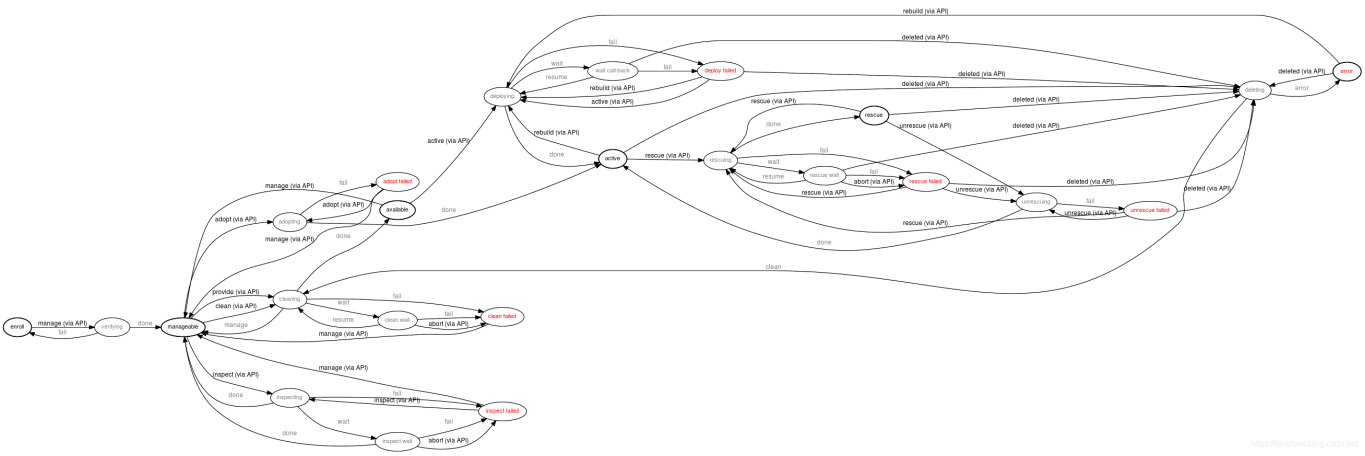
발생하면 원격으로 장비를 관리할 수 없지만, 외부 관리는 데이터 채널과 물리적으로 분리된 관리 채널을 통해 이 문제를 해결할 수 있습니다.

자원 모델 설계



1. **node**: 베어메탈의 기본 정보로, CPU, 저장소 등의 정보와 Ironic이 해당 베어메탈을 관리하는 데 사용하는 드라이버 유형 정보를 포함합니다.
2. **chassis**: 베어메탈 템플릿 정보로, 노드 관리를 위한 분류에 사용됩니다.
3. **port**: 베어메탈 네트워크 포트의 기본 정보로, MAC 주소, LLDP 등의 정보를 포함합니다.
4. **portgroup**: 베어메탈 상위 스위치와 베어메탈 네트워크 포트의 포트 그룹 구성 정보를 나타냅니다.
5. **conductor**: ironic-conductor의 상태 및 지원하는 드라이버 유형에 대한 정보를 기록합니다.
6. **volume connector/target**: 베어메탈의 블록 장치 마운트 정보를 기록합니다.

전체 수명 주기의 상태 기계 설계

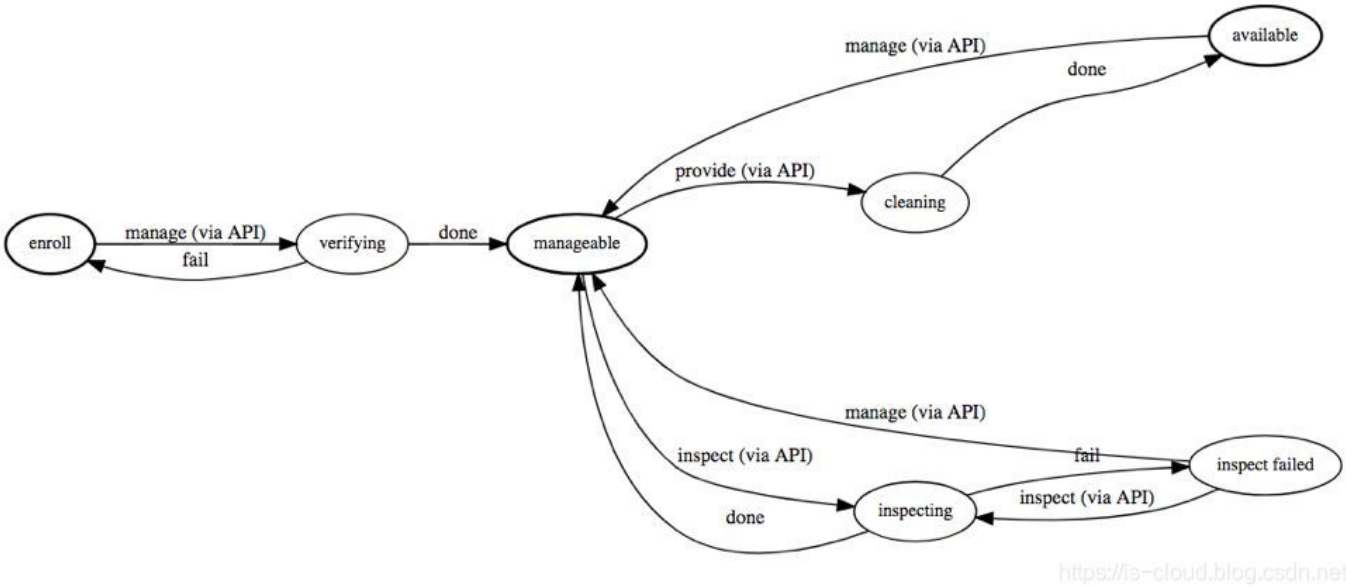


Inspection 베어메탈 자가 점검 단계

베어메탈이 하드웨어 설치와 네트워크 연결 등의 작업을 완료한 후, 관리자가 해당 베어메탈의 정보를 Ironic에 입력하여 관리 대상에 포함시킵니다. 이를 통해 후속 작업이 지원됩니다. 이 단계에서 Ironic Inspector 기능을 사용하여 베어메탈의 하드웨어 구성 정보와 상위 스위치 정보를 자동으로 수집할 수 있습니다. 즉, 베어메탈의 자가 점검을 수행하는 것입니다. 하지만 특정 드라이버 구현 방식에 따라 Ironic Inspector가 모든 작업을 완벽하게 처리하지 못할 수도 있으며, 이 경우 여전히 수동 입력이 필요할 수 있습니다.

자가 점검 단계는 주로 IPA(Ironic Python Agent)와 Ironic Inspector가 함께 수행합니다. IPA는 정보 수집 및 전송을 담당하고, Ironic Inspector는 수집된 정보를 처리합니다. IPA가 수집하는 정보에는 CPU 개수, CPU Feature Flag, 메모리 크기, 디스크 크기, 네트워크 카드 IP, MAC 주소 등이 있으며, 이 정보는 Nova Scheduler의 스케줄링 요소로 사용됩니다. SDN 네트워크가 접속된 경우에는 베어메탈 네트워크 카드에서 수신된 LLDP(링크 탐색 프로토콜) 메시지를 감시해야 합니다. LLDP 메시지는 SDN 컨트롤러에서 전송되며, Chassis ID와 Port ID를 포함하여 스위치 포트 정보를 표시합니다. 이를 통해 SDN 컨트롤러는 지정된 스위치 포트에 특정 전송 규칙을 적용할 수 있습니다.

Inspection 상태 기계: 【데이터 입력】→ 【운영 관리】→ 【데이터 검증】→ 【노드 사용 가능】.

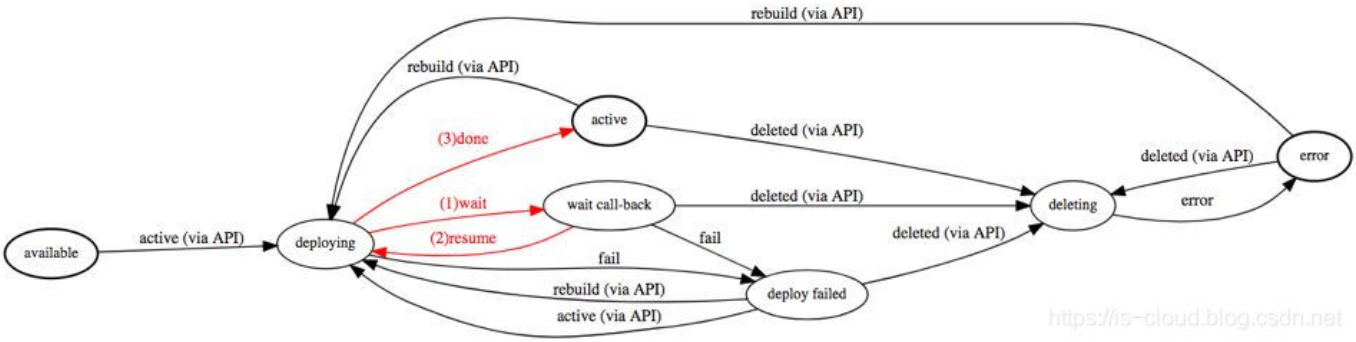


- `enroll`: 베어메탈 노드의 데이터베이스 테이블 기록을 초기화하고, 사용자의 추가 검증을 대기합니다.
- `verifying`: 사용자가 베어메탈 노드 정보를 확인한 후 manage API 요청을 발행하면, Ironic은 사용자가 입력한 정보를 기반으로 데이터 검증을 수행하여 다음 작업에 필요한 요구 사항을 충족하는지 확인합니다.
- `manageable`: 검증을 통과하면, 베어메탈 노드는 관리 가능한 상태로 전환되며, 이는 베어메탈 데이터 유지 보수의 중간 상태입니다.
- `inspecting`: Ironic이 베어메탈 노드 정보를 자동으로 수집 및 입력하는 과정으로, CPU, RAM, 디스크, NIC 등 하드웨어 정보를 자동으로 기록하여 사용자의 수동 입력을 줄이는 단계입니다.
- `available`: 사용자가 manageable 상태의 베어메탈 노드에 대해 provide API 요청을 실행하면, 해당 노드는 cleaning 상태로 전환되며, 이 단계는 깨끗한 베어메탈 노드를 제공하기 위한 것입니다. cleaning이 완료된 후 해당 노드는 공식적으로 사용 가능한 상태로 표시됩니다. 이때 베어메탈 노드의 기본 정보(CPU, RAM, 디스크, NIC 등)가 모두 입력된 상태입니다.

Provision 베어메탈 배포 단계

베어메탈이 자가 점검을 완료하고 available 상태에 도달하면, Provision 배포 단계로 진입할 수 있습니다. 이 단계에서는 사용자가 비즈니스 요구에 따라 이미지, 네트워크 등의 정보를 지정하여 베어메탈 인스턴스를 배포합니다. 클라우드 플랫폼은 자원 스케줄링, 운영체제 설치, 네트워크 구성 등의 작업을 자동으로 완료합니다. 인스턴스가 성공적으로 생성되면, 사용자는 물리 서버를 사용해 비즈니스를 운영할 수 있습니다.

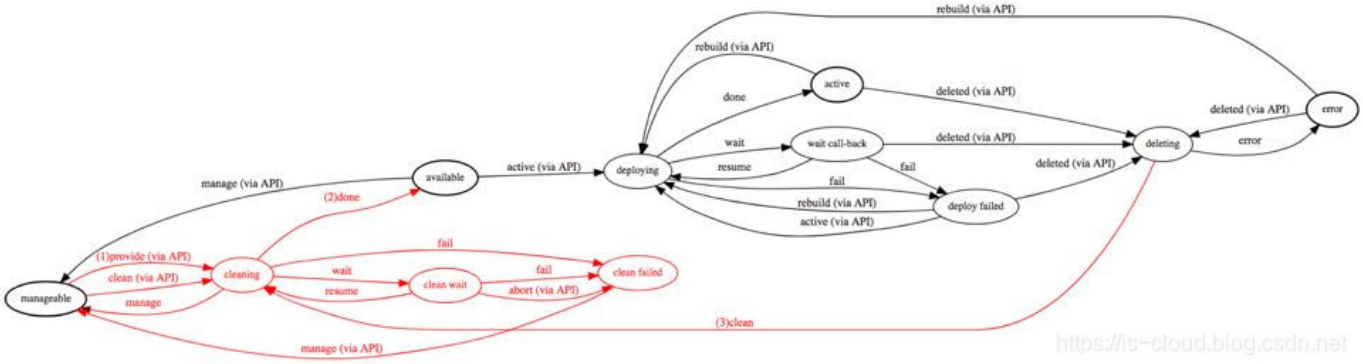
Provision 상태 기계: 【배포 템플릿 설정】→ 【배포 매개변수 전달】→ 【베어메탈 randisk 단계로 진입】→ 【ironic-python-agent가 베어메탈 노드를 인수】→ 【이미지 데이터 주입】→【운영체제 부팅】→【베어메탈 노드 활성화】.



- `available`: 사용자는 사용 가능한 상태의 베어메탈 노드에 대해 active API 요청을 발행하여 운영 체제 배포를 수행할 수 있습니다. 이 요청 시 배포의 상세 정보(예: 사용자 이미지, 인스턴스 메타데이터, 네트워크 자원 할당)를 베어메탈 데이터베이스 테이블에 지속적으로 저장해야 합니다.
- `deploying`: 배포 준비 단계로, 이때 Ironic은 자동으로 램디스크를 ironic-conductor에 캐시하고 부팅 작업을 수행합니다. 만약 이 시점에 베어메탈 노드가 켜져 있으면 재부팅을 수행합니다.
- `wait callback`: 베어메탈 노드가 부팅 또는 재부팅을 완료하기를 기다립니다. Ironic은 다양한 드라이버 유형의 정책에 따라 베어메탈 노드가 운영 체제 부트로더에 진입하도록 제어합니다. PXE가 기본 부팅 방식이며, 이 단계에서 Ironic은 베어메탈 노드의 부트로더가 램디스크로 진입하기를 기다립니다. 램디스크에서 실행되는 ironic-python-agent는 Ironic에 콜백하여 베어메탈 노드가 ironic-python-agent의 제어를 받았음을 알리고, 사용자 이미지의 주입 작업을 수행하며 진행 상황을 모니터링합니다. 사용자 이미지 주입이 완료되면 Ironic은 베어메탈 노드가 사용자 이미지로부터 부팅하도록 추가로 제어하고, 제어 계층의 데이터 유지를 완료합니다.
- `active`: 이 상태는 베어메탈 노드의 운영 체제 배포가 완료되었음을 나타내지만, 운영 체제가 완전히 로드되었음을 의미하지는 않습니다. 베어메탈 노드의 운영 체제가 로드되려면 여전히 일정 시간이 필요합니다.

Clean 베어메탈 회수 단계

Clean 단계는 Ironic이 다중 테넌트 환경에서 항상 깨끗한(구성이 통일되고 불필요한 데이터 잔여물이 없는) 베어메탈 노드를 사용자들에게 제공할 수 있도록 보장합니다. Clean 단계는 베어메탈 노드의 구성 및 데이터 정리를 위한 통일되고 확장 가능한 프로세스를 설계했습니다. 이 프로세스를 통해 사용자는 디스크 초기화, RAID 구성, BIOS 설정 등과 같은 필요한 정리 작업 및 이들의 실행 우선순위를 지정할 수 있습니다.



- Inspection 단계에서 Provide 요청을 받은 후 Cleaning을 수행하여 사용 가능한 베어메탈 노드가 깨끗한 상태임을 보장합니다.
- Provision 단계에서 Delete 요청을 받은 후 Cleaning을 수행하여 사용 가능한 베어메탈 노드가 깨끗한 상태임을 보장합니다.

Ironic(Stein) 빠른 설치 (devstack)

공식문서 :
https://docs.openstack.org/ironic/stein/contributor/dev-quickstart.html#deploying-ironic-with-devstack

devstack 다운로드

```
git clone https://git.openstack.org/openstack-dev/devstack.git -b
stable/stein
sudo ./devstack/tools/create-stack-user.sh
sudo su - stack
```

local.conf 설정

```
[[local|localrc]]
HOST_IP=192.168.1.100

# Use TryStack(99cloud) git mirror
GIT_BASE=http://git.trystack.cn
#GIT_BASE=https://git.openstack.org

# Reclone each time
RECLONE=no

# Enable Logging
DEST=/opt/stack
LOGFILE=$DEST/logs/stack.sh.log
VERBOSE=True
LOG_COLOR=True
SCREEN_LOGDIR=$DEST/logs
LOGDAYS=1
```

```
# Define images to be automatically downloaded during the DevStack built
process.
DOWNLOAD_DEFAULT_IMAGES=False
IMAGE_URLS="http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.
img"

# use TryStack git mirror
GIT_BASE=http://git.trystack.cn
NOVNC_REPO=http://git.trystack.cn/kanaka/noVNC.git
SPICE_REPO=http://git.trystack.cn/git/spice/sice-html5.git

# Apache Frontend
ENABLE_HTTPD_MOD_WSGI_SERVICES=False

# IP Version
IP_VERSION=4

# Credentials
ADMIN_PASSWORD=password
DATABASE_PASSWORD=password
RABBIT_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=password
SWIFT_HASH=password
SWIFT_TEMPURL_KEY=password

# Enable Ironic plugin
enable_plugin ironic https://git.openstack.org/openstack/ironic stable/stein

# Disable nova novnc service, ironic does not support it anyway.
disable_service n-novnc

# Enable Swift for the direct deploy interface.
enable_service s-proxy
enable_service s-object
enable_service s-container
enable_service s-account

# Cinder
VOLUME_GROUP_NAME="stack-volumes"
VOLUME_NAME_PREFIX="volume-"
VOLUME_BACKING_FILE_SIZE=100G

# Neutron
ENABLED_SERVICES+=,q-svc,q-agt,q-dhcp,q-l3,q-meta
# 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
```

```
# Swift temp URL's are required for the direct deploy interface
SWIFT_ENABLE_TEMPURLS=True

# Create 3 virtual machines to pose as Ironic's baremetal nodes.
IRONIC_VM_COUNT=3
IRONIC_BAREMETAL_BASIC_OPS=True
DEFAULT_INSTANCE_TYPE=baremetal

# Enable additional hardware types, if needed.
#IRONIC_ENABLED_HARDWARE_TYPES=ipmi,fake-hardware
# Don't forget that many hardware types require enabling of additional
# interfaces, most often power and management:
#IRONIC_ENABLED_MANAGEMENT_INTERFACES=ipmitool,fake
#IRONIC_ENABLED_POWER_INTERFACES=ipmitool,fake
# The 'ipmi' hardware type's default deploy interface is 'iscsi'.
# This would change the default to 'direct':
#IRONIC_DEFAULT_DEPLOY_INTERFACE=direct

# Change this to alter the default driver for nodes created by devstack.
# This driver should be in the enabled list above.
IRONIC_DEPLOY_DRIVER=ipmi

# The parameters below represent the minimum possible values to create
# functional nodes.
IRONIC_VM_SPECS_RAM=1280
IRONIC_VM_SPECS_DISK=10

# Size of the ephemeral partition in GB. Use 0 for no ephemeral partition.
IRONIC_VM_EPHEMERAL_DISK=0

# To build your own IPA ramdisk from source, set this to True
IRONIC_BUILD_DEPLOY_RAMDISK=False

VIRT_DRIVER=ironic

# Log all output to files
LOGFILE=/opt/stack/devstack.log
LOGDIR=/opt/stack/logs
IRONIC_VM_LOG_DIR=/opt/stack/ironic-bm-logs
```

서비스 상태 확인

```
[root@localhost ~]# openstack compute service list
+----+------------------+----------------------+----------+---------+------
-+----------------------------+
| ID | Binary           | Host                 | Zone     | Status  | State
| Updated At                 |
+----+------------------+----------------------+----------+---------+------
-+----------------------------+
|  3 | nova-scheduler   | localhost.localdomain | internal | enabled | up
```

```
| 2019-05-03T18:56:18.000000 |
|  6 | nova-consoleauth | localhost.localdomain | internal | enabled | up
| 2019-05-03T18:56:22.000000 |
|  7 | nova-conductor   | localhost.localdomain | internal | enabled | up
| 2019-05-03T18:56:14.000000 |
|  1 | nova-conductor   | localhost.localdomain | internal | enabled | up
| 2019-05-03T18:56:15.000000 |
|  3 | nova-compute     | localhost.localdomain | nova     | enabled | up
| 2019-05-03T18:56:18.000000 |
+----+------------------+-----------------------+----------+---------+------
-+----------------------------+

[root@localhost ~]# openstack network agent list
+--------------------------------------+--------------------+--------------
--------+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host
| Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+--------------
--------+-------------------+-------+-------+---------------------------+
| 52f23bda-a645-4459-bcac-686d98d23345 | Open vSwitch agent |
localhost.localdomain | None              | :-)   | UP    | neutron-
openvswitch-agent |
| 7113312f-b0b7-4ce8-ab15-428768b30855 | L3 agent           |
localhost.localdomain | nova              | :-)   | UP    | neutron-l3-agent
|
| a45fb074-3b24-4b9e-8c8a-43117f6195f2 | Metadata agent     |
localhost.localdomain | None              | :-)   | UP    | neutron-
metadata-agent    |
| f207648b-03f3-4161-872e-5210f29099c6 | DHCP agent         |
localhost.localdomain | nova              | :-)   | UP    | neutron-dhcp-
agent        |
+--------------------------------------+--------------------+--------------
--------+-------------------+-------+-------+---------------------------+

[root@localhost ~]# openstack volume service list
+------------------+---------------------------------+------+---------+---
----+----------------------------+
| Binary           | Host                            | Zone | Status  |
State | Updated At                 |
+------------------+---------------------------------+------+---------+---
----+----------------------------+
| cinder-scheduler | localhost.localdomain           | nova | enabled | up
| 2019-05-03T18:56:54.000000 |
| cinder-volume    | localhost.localdomain@lvmdriver-1 | nova | enabled | up
| 2019-05-03T18:56:53.000000 |
+------------------+---------------------------------+------+---------+---
----+----------------------------+

[root@localhost ~]# openstack baremetal driver list
+---------------------+----------------+
| Supported driver(s) | Active host(s) |
```

```
+---------------------+----------------+
| fake-hardware       | localhost      |
| ipmi                | localhost      |
+---------------------+----------------+

[root@localhost ~]# openstack baremetal node list
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
| UUID                                 | Name   | Instance UUID | Power
State | Provisioning State | Maintenance |
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
| adda54fb-1038-4634-8d82-53922e875a1f | node-0 | None          | power off
| available          | False       |
| 6952e923-11ae-4506-b010-fd7a3c4278f5 | node-1 | None          | power off
| available          | False       |
| f3b8fe69-a840-42dd-9cbf-217be8a95431 | node-2 | None          | power off
| available          | False       |
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
```

베어메탈 인스턴스 배포

```
[root@localhost ~]# openstack server create --flavor baremetal --image
cirros-0.4.0-x86_64-disk --key-name default --nic net-id=5c86f931-64da-4c69-
a0f1-e2da6d9dd082 VM1
+-------------------------------------+-------------------------------------
----------------------------+
| Field                               | Value
|
+-------------------------------------+-------------------------------------
----------------------------+
| OS-DCF:diskConfig                   | MANUAL
|
| OS-EXT-AZ:availability_zone         |
|
| OS-EXT-SRV-ATTR:host                | None
|
| OS-EXT-SRV-ATTR:hypervisor_hostname | None
|
| OS-EXT-SRV-ATTR:instance_name       |
|
| OS-EXT-STS:power_state              | NOSTATE
|
| OS-EXT-STS:task_state               | scheduling
|
| OS-EXT-STS:vm_state                 | building
|
| OS-SRV-USG:launched_at              | None
|
```

```
| OS-SRV-USG:terminated_at            | None
|
| accessIPv4                          |
|
| accessIPv6                          |
|
| addresses                           |
|
| adminPass                           | k3TgBf5Xjsqv
|
| config_drive                        |
|
| created                             | 2019-05-03T20:26:28Z
|
| flavor                              | baremetal (8f6fd22b-9bec-4b4d-
b427-7c333e47d2c2)                |
| hostId                              |
|
| id                                  | 70e9f2b1-a292-4e95-90d4-55864bb0a71d
|
| image                               | cirros-0.4.0-x86_64-disk (4ff12aca-
b762-436c-b98c-579ad2a21649) |
| key_name                            | default
|
| name                                | VM1
|
| progress                            | 0
|
| project_id                          | cbf936fc5e9d4cfcaa1dbc06cd9d2e3e
|
| properties                          |
|
| security_groups                     | name='default'
|
| status                              | BUILD
|
| updated                             | 2019-05-03T20:26:28Z
|
| user_id                             | 405fad83a4b3470faf7d6c616fe9f7f4
|
| volumes_attached                    |
|
+-------------------------------------+-------------------------------------
----------------------------+

[root@localhost ~]# openstack baremetal node list
+--------------------------------------+--------+--------------------------
-----------+-------------+--------------------+-------------+
| UUID                                 | Name   | Instance UUID
| Power State | Provisioning State | Maintenance |
+--------------------------------------+--------+--------------------------
```

```
-----------+-------------+--------------------+-------------+
| adda54fb-1038-4634-8d82-53922e875a1f | node-0 | None
| power off   | available          | False       |
| 6952e923-11ae-4506-b010-fd7a3c4278f5 | node-1 | None
| power off   | available          | False       |
| f3b8fe69-a840-42dd-9cbf-217be8a95431 | node-2 | 70e9f2b1-
a292-4e95-90d4-55864bb0a71d | power off   | deploying          | False
|
+--------------------------------------+--------+--------------------------
-----------+-------------+--------------------+-------------+

[root@localhost ~]# openstack server list --long
+--------------------------------------+------+--------+------------+-------
------+--------------------+--------------------------+---------------------
---------------+-------------+--------------------------------------+------
-------------+----------------------+------------+
| ID                                   | Name | Status | Task State | Power
State | Networks           | Image Name               | Image ID
| Flavor Name | Flavor ID                            | Availability Zone |
Host                   | Properties |
+--------------------------------------+------+--------+------------+-------
------+--------------------+--------------------------+---------------------
---------------+-------------+--------------------------------------+------
-------------+----------------------+------------+
| 70e9f2b1-a292-4e95-90d4-55864bb0a71d | VM1  | ACTIVE | None       |
Running     | private=10.0.0.40 | cirros-0.4.0-x86_64-disk | 4ff12aca-
b762-436c-b98c-579ad2a21649 | baremetal   | 8f6fd22b-9bec-4b4d-
b427-7c333e47d2c2 | nova               | localhost.localdomain |            |
+--------------------------------------+------+--------+------------+-------
------+--------------------+--------------------------+---------------------
---------------+-------------+--------------------------------------+------
-------------+----------------------+------------+

[root@localhost ~]# openstack baremetal node list
+--------------------------------------+--------+--------------------------
-----------+-------------+--------------------+-------------+
| UUID                                 | Name   | Instance UUID
| Power State | Provisioning State | Maintenance |
+--------------------------------------+--------+--------------------------
-----------+-------------+--------------------+-------------+
| adda54fb-1038-4634-8d82-53922e875a1f | node-0 | None
| power off   | available          | False       |
| 6952e923-11ae-4506-b010-fd7a3c4278f5 | node-1 | None
| power off   | available          | False       |
| f3b8fe69-a840-42dd-9cbf-217be8a95431 | node-2 | 70e9f2b1-
a292-4e95-90d4-55864bb0a71d | power on    | deploying          | False
|
+--------------------------------------+--------+--------------------------
-----------+-------------+--------------------+-------------+

[root@localhost ~]# ssh cirros@10.0.0.40
```

```
$
```

이때 Ironic은 OpenStack Nova 드라이버로 존재합니다.

```
# nova.conf

[DEFAULT]
...
ompute_driver = ironic.IronicDriver
```

Ironic 수동 구성

Provisioning 네트워크 구성

우선 Provisioning 네트워크로 사용할 물리 네트워크를 구성합니다. 이 네트워크는 DHCP와 PXE 기능을 제공하며, 베어메탈 노드 배포에 사용됩니다.

새로운 물리 네트워크 구성

```
# /etc/neutron/plugins/ml2/ml2_conf.ini

[ml2_type_flat]
flat_networks = public, physnet1

[ovs]
datapath_type = system
bridge_mappings = public:br-ex, physnet1:br-eth2
tunnel_bridge = br-tun
local_ip = 172.22.132.93
```

OvS Provider 브릿지 생성

```
$ sudo ovs-vsctl add-br br-eth2
$ sudo ovs-vsctl add-port br-eth2 eth2
```

OvS 에이전트 재시작

```
$ sudo systemctl restart devstack@q-svc.service
$ sudo systemctl restart devstack@q-agt.service
```

공급자 네트워크 생성

```

```

```
$ neutron net-create sharednet1 \
                     --shared \
                     --provider:network_type flat \
                     --provider:physical_network physnet1

$ neutron subnet-create sharednet1 172.22.132.0/24 \
                        --name sharedsubnet1 \
                        --ip-version=4 --gateway=172.22.132.254 \
                        --allocation-pool
start=172.22.132.180,end=172.22.132.200 \
                        --enable-dhcp
```

참고: DHCP를 활성화하여 베어메탈 노드의 PXE 네트워크 카드에 IP 주소를 제공하는 것을 잊지 마세요.

Cleaning 네트워크 구성

`ironic node clean` 기능을 사용할 때는 Cleaning 네트워크가 필요합니다. 여기서는 Cleaning 네트워크와 Provisioning 네트워크를 통합합니다.

구성 수정

```
# /etc/ironic/ironic.conf

[neutron]
cleaning_network = sharednet1
```

Ironic 서비스를 재시작

```
$ sudo systemctl restart devstack@ir-api.service
$ sudo systemctl restart devstack@ir-cond.service
```

Deploy 이미지와 User 이미지 구축

Deploy 이미지와 User 이미지는 Disk Image Builder 도구를 통해 생성할 수 있습니다. DevStack 배포 시 이미 Deploy 이미지는 생성되었으므로 반복 작업은 필요 없습니다.

disk-image-builder 설치

```
$ virtualenv dib
$ source dib/bin/activate
(dib) $ pip install diskimage-builder
```

User Image 생성

```
$ cat <<EOF > k8s.repo
[kubernetes]
name=Kubernetes
baseurl=http://yum.kubernetes.io/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
       https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

$ DIB_YUM_REPO_CONF=k8s.repo \
  DIB_DEV_USER_USERNAME=kyle \
  DIB_DEV_USER_PWDLESS_SUDO=yes \
  DIB_DEV_USER_PASSWORD=r00tme \
  disk-image-create \
        centos7 \
        dhcp-all-interfaces \
        devuser \
        yum \
        epel \
        baremetal \
        -o k8s.qcow2 \
        -p vim,docker,kubelet,kubeadm,kubectl,kubernetes-cni

...
Converting image using qemu-img convert
Image file k8s.qcow2 created...

$ ls
dib  k8s.d  k8s.initrd  k8s.qcow2  k8s.repo  k8s.vmlinuz
```

생성된 이미지를 Glance에 업로드

```
# Kernel
$ openstack image create k8s.kernel \
                      --public \
                      --disk-format aki \
                      --container-format aki < k8s.vmlinuz
# Initrd
$ openstack image create k8s.initrd \
                      --public \
                      --disk-format ari \
                      --container-format ari < k8s.initrd
# Qcow2
$ export MY_VMLINUZ_UUID=$(openstack image list | awk '/k8s.kernel/ { print
$2 }')
$ export MY_INITRD_UUID=$(openstack image list | awk '/k8s.initrd/ { print
```

```
$2 }')
$ openstack image create k8s \
                       --public \
                       --disk-format qcow2 \
                       --container-format bare \
                       --property kernel_id=$MY_VMLINUZ_UUID \
                       --property ramdisk_id=$MY_INITRD_UUID < k8s.qcow2
```

Ironic Node 생성

여기서 말하는 Ironic 노드는 Ironic 데몬이 실행되는 노드를 의미하는 것이 아니라, 베어메탈을 지칭한다는 점에 주의해야 합니다. 이러한 명명상의 차이점을 유념해야 합니다.

필요한 드라이버(해당 Ironic 노드와 대응되는)가 로드되었는지 확인하세요.

```
$ ironic driver-list
+---------------------+----------------+
| Supported driver(s) | Active host(s) |
+---------------------+----------------+
| agent_ipmitool      | ironic-dev     |
| fake                | ironic-dev     |
| ipmi                | ironic-dev     |
| pxe_ipmitool        | ironic-dev     |
+---------------------+----------------+
```

참고: 누락된 경우, Set up the drivers for the Bare Metal service 지침에 따라 드라이버를 추가할 수 있습니다.

Ironic 노드를 생성하고, 수동 입력 방식으로 노드를 등록하세요.

```
$ export DEPLOY_VMLINUZ_UUID=$(openstack image list | awk '/ipmitool.kernel/
{ print $2 }')
$ export DEPLOY_INITRD_UUID=$(openstack image list | awk
'/ipmitool.initramfs/ { print $2 }')
$ ironic node-create -d agent_ipmitool \
                     -n bare-node-1 \
                     -i ipmi_address=172.20.3.194 \
                     -i ipmi_username=maas \
                     -i ipmi_password=passwd \
                     -i ipmi_port=623 \
                     -i ipmi_terminal_port=9000 \
                     -i deploy_kernel=$DEPLOY_VMLINUZ_UUID \
                     -i deploy_ramdisk=$DEPLOY_INITRD_UUID
```

노드의 상세 정보를 계속 입력하세요.

```
$ export NODE_UUID=$(ironic node-list | awk '/bare-node-1/ { print $2 }')
$ ironic node-update $NODE_UUID add \
                     properties/cpus=4 \
                     properties/memory_mb=8192 \
                     properties/local_gb=100 \
                     properties/root_gb=100 \
                     properties/cpu_arch=x86_64
```

참고: 위의 정보는 추가적인 설정을 통해 Ironic Inspector를 사용하여 자동으로 입력할 수도 있습니다. 예를 들어:

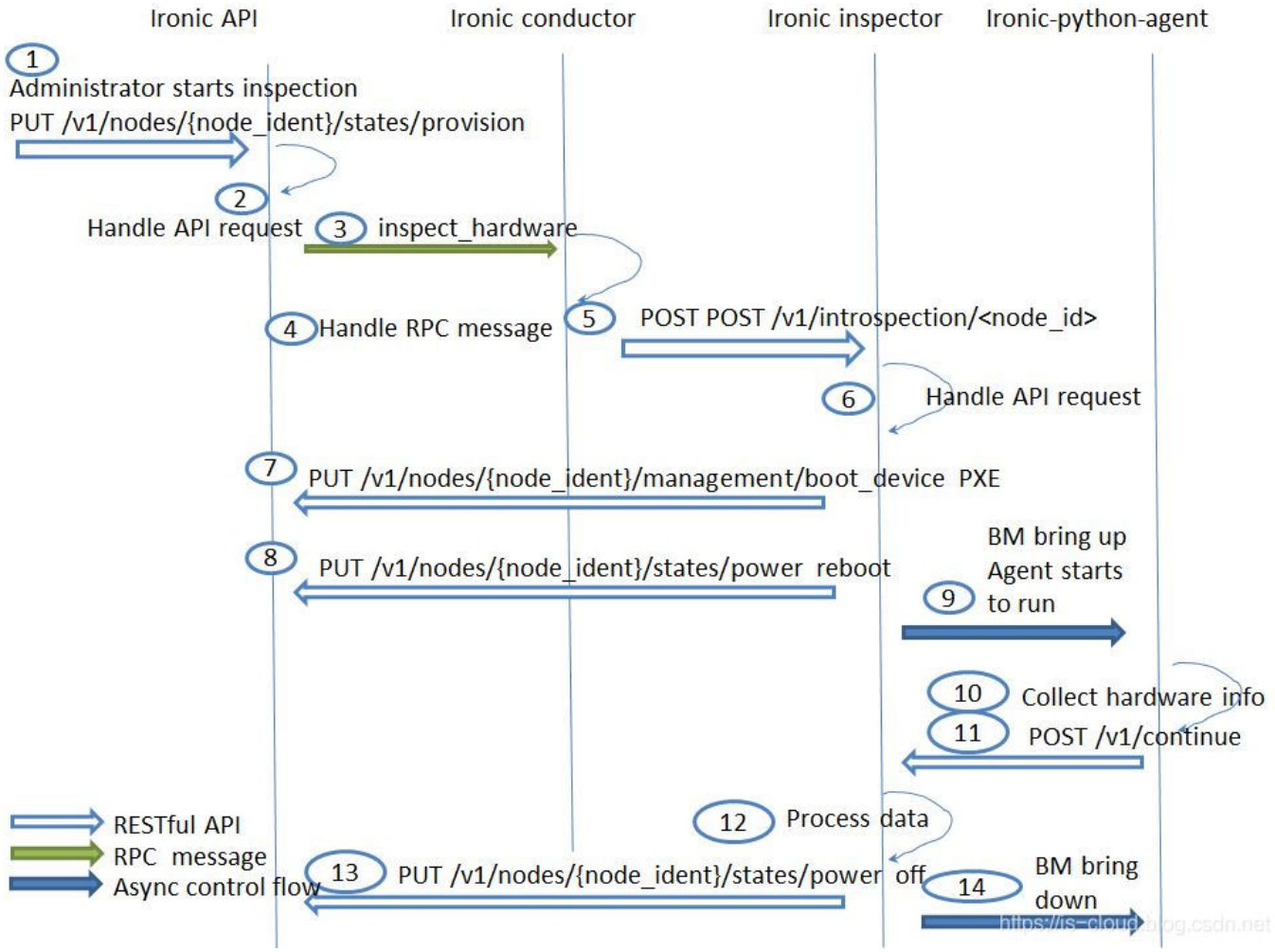
```

# /etc/ironic-inspector/dnsmasq.conf

no-daemon
port=0
interface=eth1
bind-interfaces
dhcp-range=172.22.132.200,172.22.132.210
dhcp-match=ipxe,175
dhcp-boot=tag:!ipxe,undionly.kpxe
dhcp-boot=tag:ipxe,http://172.22.132.93:3928/ironic-inspector.ipxe
dhcp-sequential-ip

$ devstack@ironic-inspector-dhcp.service
$ devstack@ironic-inspector.service
```

Inspection 프로세스



Ironic 노드의 네트워크 정보를 등록하세요.

```
$ ironic port-create -n $NODE_UUID -a NODE_PXE_NIC_MAC_ADDRESS
```

Ironic 노드에 입력한 데이터를 검증하세요.

```
$ ironic node-validate $NODE_UUID
+------------+--------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Interface  | Result | Reason                                                                                                                                                                         |
+------------+--------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| boot       | False  | Cannot validate image information for node 8e6fd86a-8eed-4e24-a510-3f5ebb0a336a because one or more parameters are missing from its instance_info. Missing are: ['ramdisk', 'kernel', 'image_source'] |
| console    | False  | Missing 'ipmi_terminal_port' parameter in node\'s driver_info.                                                                                                                 |
```

```
| deploy     | False  | Cannot validate image information for node
8e6fd86a-8eed-4e24-a510-3f5ebb0a336a because one or more parameters are
missing from its instance_info. Missing are: ['ramdisk', 'kernel',
'image_source'] |
| inspect    | True   |
|
| management | True   |
|
| network    | True   |
|
| power      | True   |
|
| raid       | True   |
|
| storage    | True   |
|
+------------+--------+-----------------------------------------------------
--------------------------------------------------------------------------
-------------------------------------------------------------------+
```

참고: Ironic이 노드 드라이버로 사용되는 경우 boot와 deploy가 False로 표시되는 것은 정상입니다.

Ironic 노드가 관리될 수 있는지 검증하세요.

```
$ ironic --ironic-api-version 1.34 node-set-provision-state $NODE_UUID
manage
$ ironic --ironic-api-version 1.34 node-set-provision-state $NODE_UUID
provide
$ ironic node-list
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
| UUID                                 | Name   | Instance UUID | Power
State | Provisioning State | Maintenance |
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
| 0c20cf7d-0a36-46f4-ac38-721ff8bfb646 | bare-0 | None          | power off
| cleaning           | False       |
+--------------------------------------+--------+---------------+-----------
--+--------------------+-------------+
```

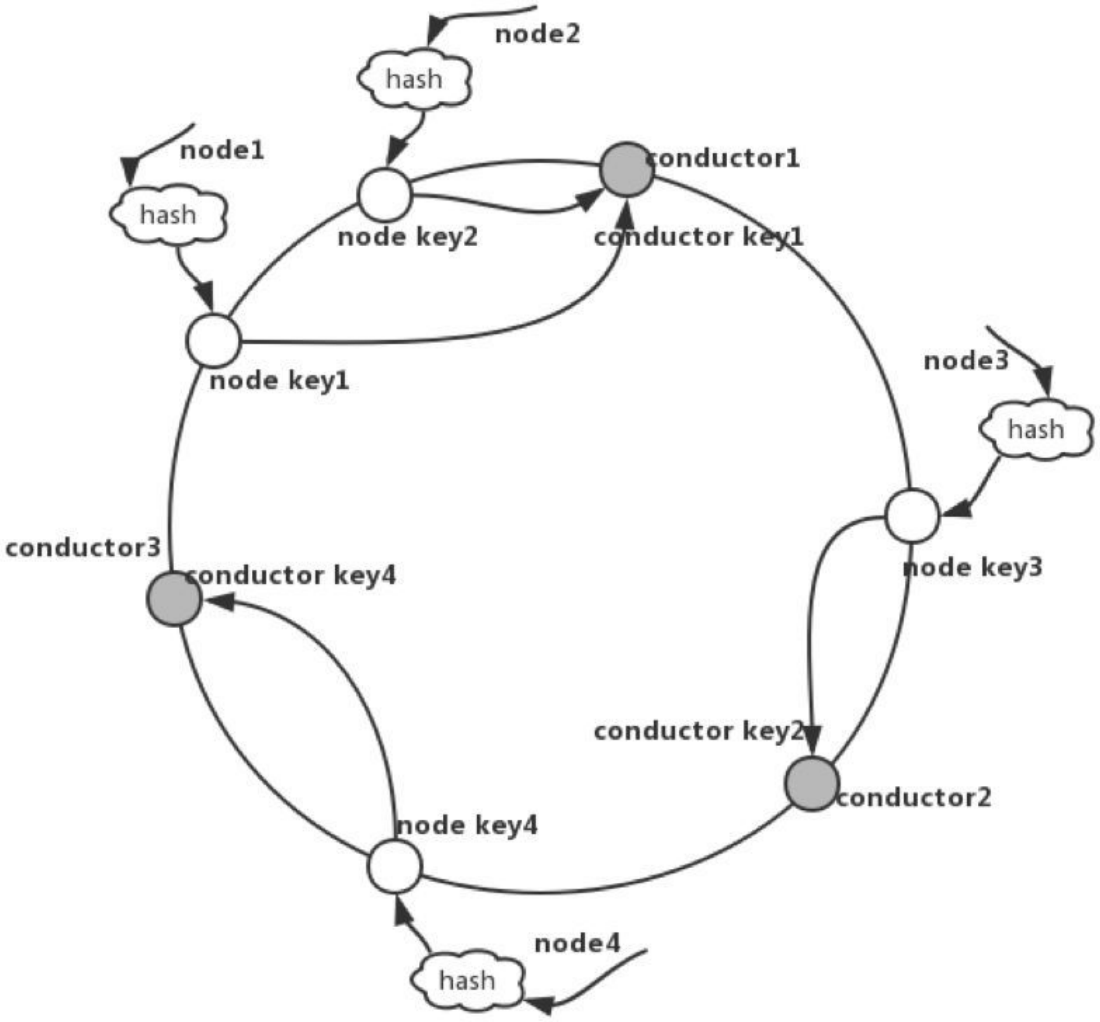
참고: Ironic 노드의 상태가 `cleaning`에서 `available`로 변경된 후에 해당 노드는 정식으로 관리에 성공하며, 그때부터 앞서 언급한 베어메탈 인스턴스 배포 작업을 계속 진행할 수 있습니다.

Ironic Conductor의 해시 매핑

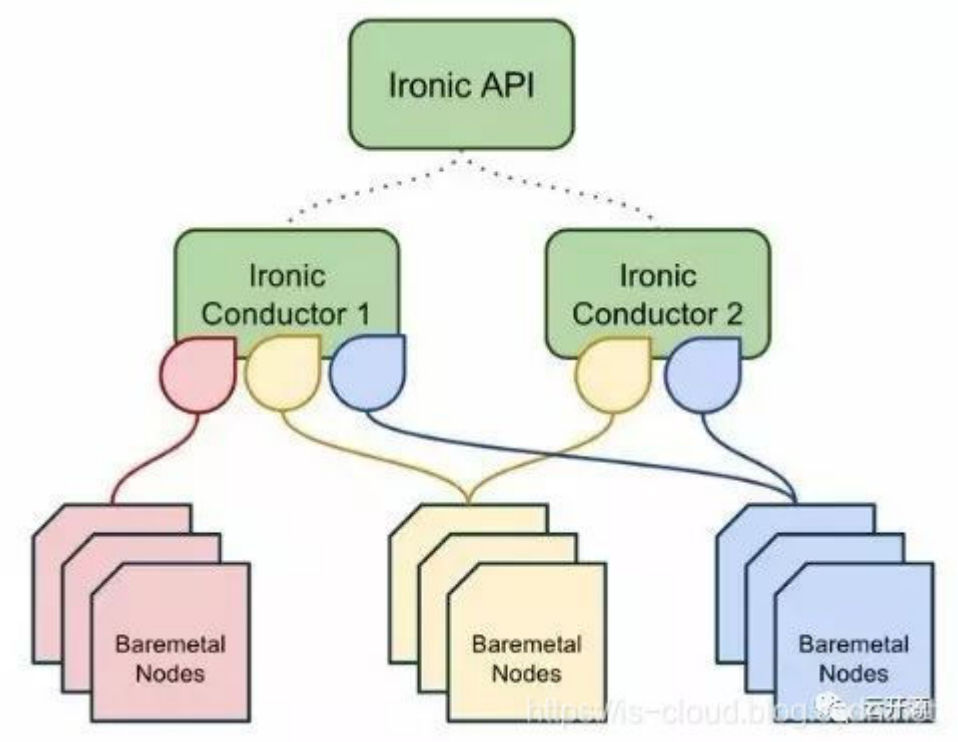
각 Conductor 인스턴스는 시작 시 데이터베이스에 등록되며, 등록된 정보에는 해당 인스턴스가 지원하는 드라이버 목록이 포함됩니다. Conductor는 주기적으로 자신의 타임스탬프를 업데이트하여 Ironic이

사용 가능한 Conductor와 드라이버를 알 수 있게 합니다. 사용자가 베어메탈 노드를 등록할 때는 해당 노드에 맞는 드라이버를 지정해야 하며, Ironic은 등록된 정보를 기반으로 해당 드라이버를 지원하는 Conductor 인스턴스에 노드를 할당합니다.

여러 Conductor와 여러 베어메탈 노드 간의 연관 관계에서 상태 관리 및 관리 충돌 방지를 위해 Ironic은 일관성 해시 알고리즘을 사용합니다. 여러 베어메탈 노드는 이 일관성 해시 알고리즘을 기반으로 하나의 Conductor 그룹에 매핑됩니다. Conductor 인스턴스가 클러스터에 추가되거나 제거되면, 베어메탈 노드가 다른 Conductor로 다시 매핑되며, 이때 여러 가지 동작(예: take-over 또는 clean-up)이 트리거될 수 있습니다.

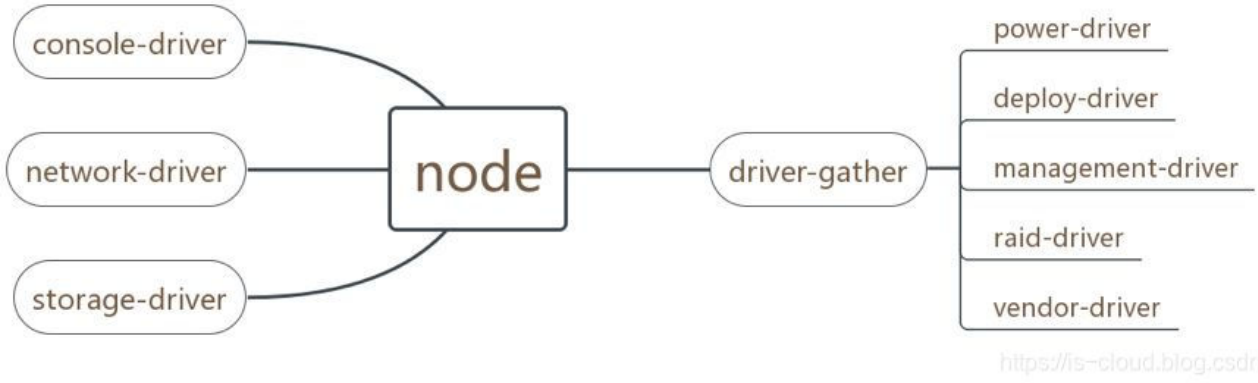


https://is-cloud.blog.csdn.net



Ironic 드라이버 모델

Ironic은 드라이버 모델을 설계할 때 기능 모듈의 높은 응집성, 느슨한 결합성, 재사용성, 그리고 조합 가능성을 최대한 고려했습니다. 하나의 드라이버 세트가 하나의 베어메탈 노드 관리를 담당하도록 설계되었습니다.



드라이버 유형은 다음과 같이 나눌 수 있습니다:

1. **power**: 전원 관리 드라이버
2. **boot**: 부트(시작) 관리 드라이버
3. **deploy**: 운영체제 배포 관리 드라이버
4. **management**: 베어메탈 관리 드라이버
5. **raid**: RAID 관리 드라이버
6. **inspect**: 베어메탈 자가 점검 드라이버
7. **console**: 콘솔 드라이버
8. **network**: 네트워크 관리 드라이버
9. **storage**: 스토리지 관리 드라이버

10. **vendor**: 제조사 드라이버로, 제조사는 고유한 기능을 확장하고 이를 Ironic을 통해 RESTful API로 노출합니다.

driver集合名称	driver名称	driver类别
agent_ipmitool	ipmi	power
	pxe	boot
	agent	deploy

Ironic Python Agent(IPA)

Ironic Python Agent, 줄여서 IPA는 물리적 매체로 베어메탈 노드에서 실행되며, 소프트웨어 매체는 ramdisk입니다. IPA는 베어메탈 노드를 제어하고 구성하며, 디스크 초기화 및 이미지 주입 등의 작업을 수행합니다. 이는 커스터마이즈된 리눅스 커널을 부팅하고 IPA를 실행하며 Ironic Conductor와 연결되는 initramfs 이미지를 통해 이루어집니다.

- **ironic-agent.kernel**: 커스터마이즈된 리눅스 커널
- **ironic-agent.initramfs**: IPA를 실행하고 Ironic Conductor와 연결되는 이미지

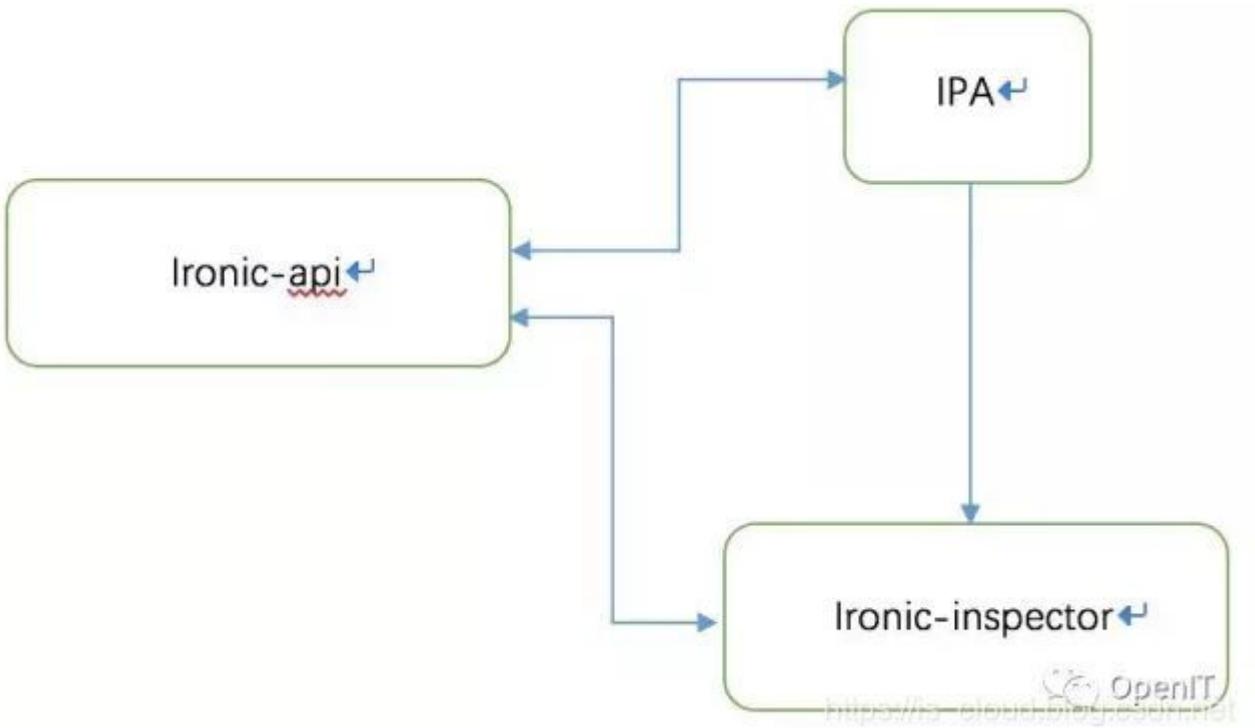
IPA 기능 목록:

- 디스크 포맷
- 디스크 파티션
- 운영체제 부트로더
- 펌웨어 업그레이드
- RAID 구성

IPA는 다음 명령을 통해 생성할 수 있습니다:

```
disk-image-create -c ironic-agent ubuntu dynamic-login stable-interface-
names proliant-tools -o ironic-agent
```

작업 흐름: PXE를 통해 시스템을 부팅한 후 IPA에 진입합니다. IPA는 수집된 정보를 ironic-inspector로 전송하며, inspector는 이 정보를 기반으로 IPMI의 IP/MAC 주소를 얻어 ironic-api와 통신하여 노드를 등록합니다. 이후 ironic-api와 IPA는 통신을 통해 agent_* 드라이버를 사용하여 Clean 단계로 진입하거나 배포 작업을 진행할 수 있습니다.



Ironic Console

Ironic은 두 가지 콘솔 유형을 지원합니다:

1. **Shellinabox**
2. **Socat**

Shellinabox는 터미널 출력을 Ajax로 구현된 HTTP 서비스로 변환하여, 브라우저를 통해 직접 접근할 수 있으며 터미널과 유사한 인터페이스를 제공합니다.

Socat은 Shellinabox와 유사하게 파이프 역할을 하지만, 터미널 스트림을 TCP 연결로 리디렉션합니다. Shellinabox는 비교적 초기 방식으로, Ironic Conductor 노드에서만 웹 서비스를 실행할 수 있다는 제한이 있어 접근 범위가 제한적입니다. 이를 해결하기 위해 커뮤니티는 Socat을 사용하여 또 다른 방식을 구현했습니다. Socat은 TCP 연결을 제공하며 Nova의 Serial Console과 연동할 수 있습니다.

이 두 방식을 사용하려면 Ironic Conductor 노드에 적절한 도구를 설치해야 합니다. Socat은 `yum` 리포지토리에서 찾을 수 있으며, Shellinabox는 표준 리포지토리에 없기 때문에 EPEL 리포지토리에서 다운로드해야 합니다. Shellinabox는 외부 종속성이 없으므로 RPM 패키지를 직접 다운로드하여 설치할 수 있습니다.

Ironic 드라이버 중에서 Socat으로 끝나는 드라이버는 Socat을 지원합니다(예: `agent_ipmitool_socat`). 나머지는 Shellinabox 방식을 사용합니다. 터미널 방식을 선택할 때는 노드를 생성할 때 명확히 결정해야 합니다. 이 두 방식은 모두 양방향 통신을 지원하며, 터미널 출력을 확인하고 키보드 입력도 가능합니다.

Ironic의 이미지

Ironic이 베어메탈 노드를 배포할 때는 두 가지 유형의 이미지가 필요합니다. 이 이미지는 모두 Disk

Image Builder 도구를 통해 구축할 수 있습니다:

- **Deploy Images**
- **User Images**

Deploy Images는 배포 단계에서 중요한 모듈인 IPA(Ironic Python Agent)를 포함합니다. IPA는 배포 단계에서 tgt+iSCSI 서비스를 제공하며, 물리 머신 발견 단계에서 목표 호스트의 물리 정보를 수집하고 보고하는 역할을 합니다. 아래 명령을 사용해 deploy image를 구축하면 두 개의 파일이 생성됩니다:

- `ironic-deploy.vmlinuz (ironic-deploy.kernel)`
- `ironic-deploy.initramfs`

```
disk-image-create ironic-agent centos7 -o ironic-deploy
```

User Images는 실제 사용자 목적의 운영체제 이미지입니다. 아래 명령을 사용하여 user image를 구축한 후에는 다음 세 개의 파일이 생성됩니다:

- `my-image.qcow2`
- `my-image.vmlinuz`
- `my-image.initrd`

이 파일들은 운영체제의 부팅과 시작에 사용됩니다.

```
image-create centos7 baremetal dhcp-all-interfaces grub2 -o my-image
```

참조링크

- https://www.cnblogs.com/jmilkfan-fanguiju/p/11825068.html

From:
https://atl.kr/dokuwiki/ - **AllThatLinux!**

Permanent link:
https://atl.kr/dokuwiki/doku.php/ironic_baremetal_service

Last update: **2024/10/11 06:11**