ironic-staging-drivers Documentation

Release 0.17.0

Ironic Staging Drivers Developers

May 10, 2022

Contents

Contents:

CHAPTER 1

Ironic Staging Drivers

The Ironic Staging Drivers is used to hold out-of-tree Ironic drivers which doesn't have means to provide a 3rd Party CI at this point in time which is required by Ironic.

The intention of this project is to provide a common place for useful drivers resolving the "hundreds of different download sites" problem.

1.1 What the Ironic Staging Drivers is not

- The Ironic Staging Drivers is **not** a project under Ironic's governance, meaning that the Ironic core group is **not responsible** for the code in this project (even though, some individuals that work in this project also hold core status in the Ironic project).
- This project is **not** a place to dump code and run away hoping that someone else will take care of it for you. Drivers included in this project should be maintained and have their bugs fixed quickly. Therefore, driver owners are going to be asked to "babysit" their driver.

1.2 How to contribute

We want to make sure that the Ironic Staging Drivers project is a welcoming and friendly place to contribute code. Therefore, we want to avoid bureaucratic processes as much as possible. If you want to propose a driver to be included in the repository: Just submit the code!

1.2.1 How do I submit the code?

1. Before we can accept your patches, you'll have to agree to a contributor license agreement.
2. Learn about how to use our Gerrit review system.
3. Get the code:

```
git clone https://opendev.org/x/ironic-staging-drivers
```

4. Make your changes and write a nice commit message explaining the change in details.
5. Submit the code!

1.3 Useful links

- Free software: Apache license
- Documentation: http://ironic-staging-drivers.readthedocs.io/en/latest/
- Source: https://opendev.org/x/ironic-staging-drivers
- Bugs: https://storyboard.openstack.org/#!/project/950

CHAPTER 2

Available drivers

2.1 Wake-On-Lan driver

2.1.1 Overview

Wake-On-Lan is a standard that allows a computer to be powered on by a network message. This is widely available and doesn't require any fancy hardware to work with[1].

The Wake-On-Lan driver is a **testing** driver not meant for production. And useful for users that wants to try Ironic with real bare metal instead of virtual machines.

It's important to note that Wake-On-Lan is only capable of powering on the machine. When power off is called the driver won't take any action and will just log a message, the power off require manual intervention to be performed.

Also, since Wake-On-Lan does not offer any means to determine the current power state of the machine, the driver relies on the power state set in the Ironic database. Any calls to the API to get the power state of the node will return the value from the Ironic's database.

2.1.2 Requirements

- Wake-On-Lan should be enabled in the BIOS

2.1.3 Configuring and Enabling

1. Add `staging-wol` to the list of `enabled_hardware_types` in */etc/ironic/ironic.conf*. Also enable the `staging-wol` power interface and the `fake` management interface. For example:

```
[DEFAULT]
enabled_hardware_types = staging-wol,ipmi
```

(continues on next page)

[1] Wake-On-Lan - https://en.wikipedia.org/wiki/Wake-on-LAN

(continued from previous page)

```
enabled_management_interfaces = fake,ipmitool
enabled_power_interfaces = staging-wol,ipmitool
```

2. Restart the Ironic conductor service:

```
service ironic-conductor restart
```

2.1.4 Registering a node

Nodes configured for Wake-On-Lan driver should have the `driver` property set to `staging-wol`.

The node should have at least one port registered with it because the Wake-On-Lan driver will use the MAC address of the ports to create the magic packet[2].

The following configuration values are optional and can be added to the node's `driver_info` as needed to match the network configuration:

- `wol_host`: The broadcast IP address; defaults to **255.255.255.255**.
- `wol_port`: The destination port; defaults to **9**.

Note: Say the `ironic-conductor` is connected to more than one network and the node you are trying to wake up is in the `192.0.2.0/24` range. The `wol_host` configuration should be set to **192.0.2.255** (the broadcast IP) so the packets will get routed correctly.

The following sequence of commands can be used to enroll a node with the Wake-On-Lan driver.

1. Create node:

```
openstack baremetal node create --driver staging-wol \
    --driver-info wol_host=<broadcast ip> \
    --driver-info wol_port=<destination port>
```

The above command `ironic node-create` will return UUID of the node, which is the value of *$NODE* in the following command.

2. Associate port with the node created:

```
openstack baremetal port create --node $NODE <MAC address>
```

Additional requirements

- Boot device order should be set to "PXE, DISK" in the BIOS setup
- BIOS must try next boot device if PXE boot failed
- Cleaning should be disabled, see[3].
- Node should be powered off before start of deploy

[2] Magic packet - https://en.wikipedia.org/wiki/Wake-on-LAN#Sending_the_magic_packet
[3] Ironic node cleaning - https://docs.openstack.org/ironic/latest/admin/cleaning.html

2.1.5 References

2.2 AMT drivers

2.2.1 Overview

AMT (Active Management Technology) drivers extend Ironic's range to the desktop. AMT/vPro is widely used in desktops to remotely control their power, similar to IPMI in servers.

AMT drivers use WS-MAN protocol to interact with AMT clients. They work on AMT 7.0/8.0/9.0. AMT 7.0 was released in 2010, so AMT drivers should work on most PCs with vPro.

The `staging-amt` hardware type uses AMT for power and boot device management.

2.2.2 Set up your environment

A detailed reference is available in Intel Active Management Technology.

A short guide follows below:

- Set up AMT Client
  - Choose a system which supports Intel AMT / vPro. Desktop and laptop systems that support this can often be identified by looking at the "Intel" tag for the word `vPro`.
  - During boot, press Ctrl+P to enter Intel MEBx management.
  - Reset password – default is `admin`. The new password must contain at least one upper case letter, one lower case letter, one digit and one special character, and be at least eight characters.
  - Go to Intel AMT Configuration:
    - Enable all features under SOL/IDER/KVM section
    - Select User Consent and choose None (No password is needed)
    - Select Network Setup section and set IP
    - Activate Network Access
  - MEBx Exit
  - Restart and enable PXE boot in bios
- Install `openwsman` on servers where `ironic-conductor` is running:
  - Fedora/RHEL: `openwsman-python`.
  - Ubuntu: `python-openwsman`'s most recent version is 2.4.3 which is enough.
  - Or build it yourself from: https://github.com/Openwsman/openwsman
- Enable the `staging-amt` hardware type, power, deploy and management interfaces, for example:

```
[DEFAULT]
enabled_hardware_types = staging-amt,ipmi
enabled_deploy_interfaces = direct
enabled_management_interfaces = staging-amt,ipmitool
enabled_power_interfaces = staging-amt,ipmitool
```

  and restart the `ironic-conductor` process:

```
service ironic-conductor restart
```

- Enroll an AMT node
- **Specify these driver_info properties for the node: `amt_password`,** `amt_address`, and `amt_username`
- Boot an instance

Note: It is recommended that nodes using the pxe_amt* driver be deployed with the local boot option. This is because the AMT firmware currently has no support for setting a persistent boot device. Nodes deployed without the local boot option could fail to boot if they are restarted outside of Ironic's control (I.E. rebooted by a local user) because the node will not attempt to PXE / network boot the kernel, using local boot solves this known issue.

2.3 iBoot driver

2.3.1 Overview

The iBoot power driver enables you to take advantage of power cycle management of nodes using Dataprobe iBoot devices over the DxP protocol.

The `staging-iboot` hardware type uses iBoot to manage power of the nodes.

Requirements

- `python-iboot` library should be installed - https://github.com/darkip/python-iboot

Tested platforms

- iBoot-G2[1]

Configuring and enabling

1. Add `staging-iboot` to `enabled_hardware_types` and `enabled_power_interfaces` in */etc/ironic/ironic.conf*. Also enable the `fake` management interface. For example:

```
[DEFAULT]
enabled_hardware_types = staging-iboot,ipmi
enabled_management_interfaces = fake,ipmitool
enabled_power_interfaces = staging-iboot,ipmitool
```

2. Restart the Ironic conductor service:

```
service ironic-conductor restart
```

[1] iBoot-G2 official documentation - http://dataprobe.com/support_iboot-g2.html

Registering a node

Nodes configured for the iBoot driver should have the `driver` property set to `staging-iboot`.

The following configuration values are also required in `driver_info`:

- `iboot_address`: The IP address of the iBoot PDU.
- `iboot_username`: User name used for authentication.
- `iboot_password`: Password used for authentication.

In addition, there are optional properties in `driver_info`:

- `iboot_port`: iBoot PDU port. Defaults to 9100.
- `iboot_relay_id`: iBoot PDU relay ID. This option is useful in order to support multiple nodes attached to a single PDU. Defaults to 1.

The following sequence of commands can be used to enroll a node with the iBoot driver.

1. Create node:

```
openstack baremetal node create --driver staging-iboot \
    --driver-info iboot_username=<username> \
    --driver-info iboot_password=<password> \
    --driver-info iboot_address=<address>
```

2.3.2 References

2.4 Libvirt drivers

2.4.1 Overview

This driver implements Power/Management interfaces for virtual baremetal hardware and is based on Libvirt[1] library and its Python interface. Thus it is suited **for testing environments only**.

It performs considerably better that Ironic's SSH driver, especially when there are many virtual baremetal nodes placed on hypervizor[2]. It also supports additional connection transports, including TCP with SASL authentication that can be considered as secure alternative to SSH.

Known drawbacks in comparison to Ironic's SSH driver are:

- no support for user+password SSH authentication
- some use cases possible with SSH driver are not supported
  - e.g. managing VirtualBox VMs on a Windows host from Linux guest

2.4.2 Setting up the environment

1. Install Ironic
2. Install ironic-staging-drivers
3. Install `libvirt-python`

[1] https://libvirt.org
[2] https://github.com/pshchelo/ironic_libvirt_vs_virsh

- When installing from PyPI, you'd need development version of `libvirt` package from your distribution (e.g. `libvirt-dev` in Ubuntu, `libvirt-devel` in Fedora) and all the usual Python packages required to compile C-extensions in your system (on DevStack, those are already installed when nova-compute is enabled).

4. Add `staging-libvirt` to the list of `enabled_hardware_types` in ironic.conf, configure the power and management interfaces, for example:

```
[DEFAULT]
enabled_hardware_types = staging-libvirt
enabled_management_interfaces = staging-libvirt
enabled_power_interfaces = staging-libvirt
```

   Then restart the ironic-conductor service.

5. Create or update existing virtual baremetal nodes to use one of libvirt-based drivers enabled in the previous step.
6. Update node properties with driver-specific fields if needed. (see *Node driver_info*). Default values are suitable for single-node DevStack.
7. Deploy the node.

Node driver_info

libvirt_uri (optional) Libvirt URI to connect to. Default is `qemu+unix:///system`.

ssh_key_filename (optional) File name of private SSH key when using `qemu+ssh://` transport. The file must have appropriate permissions for the user running ironic-conductor service. Default is to use default SSH keys for that user. Note that for private keys with password those must be pre-loaded into `ssh-agent`.

sasl_username username to authenticate as. Required when using TCP transport with SASL authentication.

sasl_password password to use for SASL authentication. Required when using TCP transport with SASL authentication.

2.4.3 References

2.5 Intel NodeManager drivers

2.5.1 Overview

This driver implements support of Intel NodeManager platform via ironic vendor interface methods. Those methods are implemented as sending raw bytes over IPMI. Hardware with Intel NodeManager 1.5 or above is required, feature must be enabled via Flash Image Tool. The driver detects internal addresses of NodeManager device automatically. The main term for NodeManager is `policy`, which can be power, thermal or boot time. Each policy identified by `policy_id` (integer number from 0 to 255). Maximum numbers of policies which can be set at the same time limited by platform. For more detailed information see full specification[1].

The `staging-nm` hardware types extends the `ipmi` hardware type with support for the `staging-nm` vendor interface.

2.5.2 Supported vendor passthru methods

In all examples below request/response are JSON bodies in the HTTP request or response.

[1] http://www.intel.com/content/www/us/en/power-management/intelligent-power-node-manager-3-0-specification.html

get_nm_version

HTTP method GET

Description Get Intel Node Manager version.

Example of response:

```
{"firmware": "1.2", "ipmi": "3.0", "nm": "3.0", "patch": "7"}
```

get_nm_capabilities

HTTP method GET

Description Get Intel Node Manager capabilities.

Example of request:

```
{"domain_id": "platform", "policy_trigger": "none",
 "power_domain": "primary"}
```

Example of response:

```
{"domain_id": "platform", "max_correction_time": 100000,
 "max_limit_value": 4096, "max_policies": 16,
 "max_reporting_period": 32768, "min_correction_time": 10,
 "min_limit_value": 100, "min_reporting_period": 100,
 "power_domain": "primary"}
```

control_nm_policy

HTTP method PUT

Description Enable or disable Intel Node Manager policy control.

Example of request:

```
{"scope": "policy", "enable": false, "policy_id": 10}
```

set_nm_policy

HTTP method PUT

Description Set Intel Node Manager policy. This method creates new policy if provided `policy_id` is not present or changes current policy.

Example of request:

```
{"domain_id": "platform", "enable": true, "policy_id": 10,
 "policy_trigger": "none", "action": "alert", "power_domain": "primary",
 "target_limit": 200, "reporting_period": 20000}
```

get_nm_policy

HTTP method GET

Description Get Intel Node Manager policy.

Example of request:

```
{"domain_id": "platform", "policy_id": 11}
```

Example of response:

```
{"action": "alert", "correction_time": 10000, "cpu_power_correction": "auto",
 "created_by_nm": true, "domain_id": "platform", "enabled": true,
 "global_enabled": true, "per_domain_enabled": true,
 "policy_trigger": "none", "power_domain": "primary", "power_policy": false,
 "reporting_period": 20000, "storage": "persistent", "target_limit": 250,
 "trigger_limit": 300}
```

remove_nm_policy

HTTP method DELETE

Description Remove Intel Node Manager policy.

Example of request:

```
{"domain_id": "platform", "policy_id": 11}
```

set_nm_policy_suspend

HTTP method PUT

Description Set Intel Node Manager policy suspend periods.

Example of request:

```
{"domain_id": "platform", "policy_id": 10,
 "periods": [{"start": 10, "stop": 60, "days": ["monday", "tuesday"]}]}
```

For information about time periods calculation please read NodeManager specification.

get_nm_policy_suspend

HTTP method GET

Description Get Intel Node Manager policy suspend periods.

Example of request:

```
{"domain_id": "platform", "policy_id": 13}
```

Example of response:

```
{"domain_id": "platform", "policy_id": 13,
 "periods": [{"start": 20, "stop": 100, "days": ["monday", "tuesday"]},
             {"start": 30, "stop": 150, "days": ["friday", "sunday"]}]}
```

remove_nm_policy_suspend

HTTP method DELETE

Description Remove Intel Node Manager policy suspend periods.

Example of request:

```
{"domain_id": "platform", "policy_id": 13}
```

get_nm_statistics

HTTP method GET

Description Get Intel Node Manager statistics.

Example of request:

```
{"scope": "global", "domain_id": "platform", "parameter_name": "power"}
```

Example of response:

```
{"activation_state": true, "administrative_enabled": true,
 "average_value": 200, "current_value": 202, "domain_id": "platform",
 "maximum_value": 240, "measurement_state": true, "minimum_value": 150,
 "operational_state": true, "reporting_period": 2125,
 "timestamp": "2016-02-03T20:13:52"}
```

reset_nm_statistics

HTTP method DELETE

Description Reset Intel Node Manager statistics.

Example of request:

```
{"scope": "global", "domain_id": "platform"}
```

2.5.3 References

CHAPTER 3

Releasing ironic-staging-drivers

This section is relevant to the maintainers of ironic-staging-drivers. You have to be a member of the ironic-staging-drivers-release group to do releases.

1. Verify that the CI works via a dummy patch.
2. Create a **signed** tag locally:

   ```
   git tag -s -m "Release <version>" <version>
   ```

3. Push the new tag to gerrit:

   ```
   git push gerrit <version>
   ```

4. Wait for the new release to appear on PyPI, contact the infra team in case of any issues.
5. If a stable branch is needed, go to the *branches* section of the ironic-staging-drivers settings in gerrit and add a new branch from the newly created tag.
6. If a stable branch has been created, submit a change for it that:
   1. updates `.gitreview` with a new `defaultbranch`,
   2. updates `extra-requirements.txt` with a link to ironic stable,
   3. updates `tox.ini` to use upper constraints from the corresponding release.

   See the stable/xena patch for an example.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search