Ironic Inspector Documentation

Release 10.4.3.dev4

OpenStack Foundation

Mar 18, 2024

CONTENTS

CHAPTER ONE

INTRODUCTION

This is an auxiliary service for discovering hardware properties for a node managed by Ironic. Hardware introspection or hardware properties discovery is a process of getting hardware parameters required for scheduling from a bare metal node, given its power management credentials (e.g. IPMI address, user name and password).

- Free software: Apache license
- Source: https://opendev.org/openstack/ironic-inspector/
- Bugs: https://storyboard.openstack.org/#!/project/944
- Downloads: https://tarballs.openstack.org/ironic-inspector/
- Documentation: https://docs.openstack.org/ironic-inspector/latest/
- Python client library and CLI tool: python-ironic-inspector-client (documentation).

Note: **ironic-inspector** was called *ironic-discoverd* before version 2.0.0.

CHAPTER TWO

RELEASE NOTES

For information on any current or prior version, see the release notes.

CHAPTER THREE

USING IRONIC INSPECTOR

3.1 Install Guide

Install from PyPI (you may want to use virtualenv to isolate your environment):

```
pip install ironic-inspector
```

Also there is a DevStack plugin for **ironic-inspector** - see *How To Contribute* for the current status.

Finally, some distributions (e.g. Fedora) provide **ironic-inspector** packaged, some of them - under its old name *ironic-discoverd*.

There are several projects you can use to set up **ironic-inspector** in production. puppet-ironic provides Puppet manifests, while bifrost provides an Ansible-based standalone installer. Refer to *Configuration* if you plan on installing **ironic-inspector** manually.

Note: Please beware of *possible DNS issues* when installing **ironic-inspector** on Ubuntu.

3.1.1 Sample Configuration Files

To generate a sample configuration file, run the following command from the top level of the code tree:

```
tox -egenconfig
```

For a pre-generated sample configuration file, see *Ironic Inspector Configuration Options*.

To generate a sample policy file, run the following command from the top level of the code tree:

```
tox -egenpolicy
```

For a pre-generated sample configuration file, see *Ironic Inspector Policy*.

3.1.2 Installation options

Starting with Train release, ironic-inspector can run in a non-standalone mode, which means ironic-inspector API and ironic-inspector conductor are separated services, they can be installed on the same host or different hosts.

Following are some considerations when you run ironic-inspector in non-standalone mode:

- Additional packages may be required depending on the tooz backend used in the installation. For example, `etcd3gw` is required if the backend driver is configured to use `etcd3+http://`, `pymemcache` is required to use `memcached://`. Some distributions may provide packages like `python3-etcd3gw` or `python3-memcache`. Supported drivers are listed at Tooz drivers.
- For ironic-inspector running in non-standalone mode, PXE configuration is only required on the node where ironic-inspector conductor service is deployed.
- Switch to a database backend other than sqlite.

3.1.3 Configuration

Copy the sample configuration files to some permanent place (e.g. `/etc/ironic-inspector/inspector.conf`). Fill in these minimum configuration values:

- The `standalone` in the `DEFAULT` section - This determines whether ironic-inspector services are intended to be deployed separately.
- The `keystone_authtoken` section - credentials to use when checking user authentication.
- The `ironic` section - credentials to use when accessing **ironic** API. When **ironic** is deployed standalone with no authentication, specify the following:

  ```
  [ironic]
  auth_type=none
  ```

  When **ironic** is deployed standalone with HTTP Basic authentication, valid credentials are also required:

  ```
  [ironic]
  auth_type=http_basic
  username=myName
  password=myPassword
  ```

- `connection` in the `database` section - SQLAlchemy connection string for the database. By default ironic-inspector uses sqlite as the database backend, if you are running ironic-inspector in a non-standalone mode, please change to other database backends.
- `dnsmasq_interface` in the `iptables` section - interface on which `dnsmasq` (or another DHCP service) listens for PXE boot requests (defaults to `br-ctlplane` which is a sane default for **tripleo**-based installations but is unlikely to work for other cases).
- if you wish to use the `dnsmasq` PXE/DHCP filter driver rather than the default `iptables` driver, see the *dnsmasq PXE filter* description.
- `store_data` in the `processing` section defines where introspection data is stored and takes one of three values:

none introspection data is not stored (the default)

database introspection data is stored in the database (recommended for standalone deployments)

swift introspection data is stored in the Object Store service (recommended for full openstack deployments)

Note: It is possible to create third party storage backends using the `ironic_inspector.introspection_data.store` entry point.

See comments inside *the sample configuration* for other possible configuration options.

Note: Configuration file contains a password and thus should be owned by `root` and should have access rights like `0600`.

Here is an example *inspector.conf* (adapted from a gate run):

```
[DEFAULT]
debug = false
rootwrap_config = /etc/ironic-inspector/rootwrap.conf

[database]
connection = mysql+pymysql://root:<PASSWORD>@127.0.0.1/ironic_inspector?
↪charset=utf8

[pxe_filter]
driver=iptables

[iptables]
dnsmasq_interface = br-ctlplane

[ironic]
os_region = RegionOne
project_name = service
password = <PASSWORD>
username = ironic-inspector
auth_url = http://127.0.0.1/identity
auth_type = password

[keystone_authtoken]
www_authenticate_uri = http://127.0.0.1/identity
project_name = service
password = <PASSWORD>
username = ironic-inspector
auth_url = http://127.0.0.1/identity_v2_admin
auth_type = password

[processing]
ramdisk_logs_dir = /var/log/ironic-inspector/ramdisk
store_data = swift

[swift]
os_region = RegionOne
```

(continues on next page)

(continued from previous page)

```
project_name = service
password = <PASSWORD>
username = ironic-inspector
auth_url = http://127.0.0.1/identity
auth_type = password
```

Note: Set `debug = true` if you want to see complete logs.

ironic-inspector requires root rights for managing `iptables`. It gets them by running `ironic-inspector-rootwrap` utility with `sudo`. To allow it, copy file `rootwrap.conf` and directory `rootwrap.d` to the configuration directory (e.g. `/etc/ironic-inspector/`) and create file `/etc/sudoers.d/ironic-inspector-rootwrap` with the following content:

```
Defaults:stack !requiretty
stack ALL=(root) NOPASSWD: /usr/bin/ironic-inspector-rootwrap /etc/ironic-
↪inspector/rootwrap.conf *
```

Danger: Be very careful about typos in `/etc/sudoers.d/ironic-inspector-rootwrap` as any typo will break sudo for **ALL** users on the system. Especially, make sure there is a new line at the end of this file.

Note: `rootwrap.conf` and all files in `rootwrap.d` must be writeable only by root.

Note: If you store `rootwrap.d` in a different location, make sure to update the *filters_path* option in `rootwrap.conf` to reflect the change.

If your `rootwrap.conf` is in a different location, then you need to update the *rootwrap_config* option in `ironic-inspector.conf` to point to that location.

Replace `stack` with whatever user youll be using to run **ironic-inspector**.

Configuring IPA

ironic-python-agent is a ramdisk developed for **ironic** and support for **ironic-inspector** was added during the Liberty cycle. This is the default ramdisk starting with the Mitaka release.

Note: You need at least 2 GiB of RAM on the machines to use IPA built with diskimage-builder and at least 384 MiB to use the *TinyIPA*.

To build an **ironic-python-agent** ramdisk, use ironic-python-agent-builder. Alternatively, you can download a prebuild image.

For local testing and CI purposes you can use a TinyIPA image.

Configuring PXE

For the PXE boot environment, youll need:

- TFTP server running and accessible (see below for using *dnsmasq*). Ensure `pxelinux.0` is present in the TFTP root.

  Copy `ironic-python-agent.kernel` and `ironic-python-agent.initramfs` to the TFTP root as well.

- Next, setup `$TFTPROOT/pxelinux.cfg/default` as follows:

  ```
  default introspect

  label introspect
  kernel ironic-python-agent.kernel
  append initrd=ironic-python-agent.initramfs ipa-inspection-callback-
  ↪url=http://{IP}:5050/v1/continue systemd.journald.forward_to_
  ↪console=yes

  ipappend 3
  ```

  Replace `{IP}` with IP of the machine (do not use loopback interface, it will be accessed by ramdisk on a booting machine).

  Note: While `systemd.journald.forward_to_console=yes` is not actually required, it will substantially simplify debugging if something goes wrong. You can also enable IPA debug logging by appending `ipa-debug=1`.

  IPA is pluggable: you can insert introspection plugins called *collectors* into it. For example, to enable a very handy `logs` collector (sending ramdisk logs to **ironic-inspector**), modify the `append` line in `$TFTPROOT/pxelinux.cfg/default`:

  ```
  append initrd=ironic-python-agent.initramfs ipa-inspection-callback-
  ↪url=http://{IP}:5050/v1/continue ipa-inspection-collectors=default,
  ↪logs systemd.journald.forward_to_console=yes
  ```

  Note: You probably want to always keep the `default` collector, as it provides the basic information required for introspection.

- You need PXE boot server (e.g. *dnsmasq*) running on **the same** machine as **ironic-inspector**. Dont do any firewall configuration: **ironic-inspector** will handle it for you. In **ironic-inspector** configuration file set `dnsmasq_interface` to the interface your PXE boot server listens on. Here is an example *dnsmasq.conf*:

  ```
  port=0
  interface={INTERFACE}
  bind-interfaces
  dhcp-range={DHCP IP RANGE, e.g. 192.168.0.50,192.168.0.150}
  enable-tftp
  tftp-root={TFTP ROOT, e.g. /tftpboot}
  dhcp-boot=pxelinux.0
  dhcp-sequential-ip
  ```

Note: `dhcp-sequential-ip` is used because otherwise a lot of nodes booting simultaneously cause conflicts - the same IP address is suggested to several nodes.

Configuring iPXE

iPXE allows better scaling as it primarily uses the HTTP protocol instead of slow and unreliable TFTP. You still need a TFTP server as a fallback for nodes not supporting iPXE. To use iPXE, youll need:

- TFTP server running and accessible (see above for using *dnsmasq*). Ensure `undionly.kpxe` is present in the TFTP root. If any of your nodes boot with UEFI, youll also need `ipxe.efi` there.
- You also need an HTTP server capable of serving static files. Copy `ironic-python-agent.kernel` and `ironic-python-agent.initramfs` there.
- Create a file called `inspector.ipxe` in the HTTP root (you can name and place it differently, just dont forget to adjust the *dnsmasq.conf* example below):

```
#!ipxe

:retry_dhcp
dhcp || goto retry_dhcp

:retry_boot
imgfree
kernel --timeout 30000 http://{IP}:8088/ironic-python-agent.kernel␣
↪ipa-inspection-callback-url=http://{IP}>:5050/v1/continue systemd.
↪journald.forward_to_console=yes BOOTIF=${mac} initrd=agent.ramdisk␣
↪|| goto retry_boot
initrd --timeout 30000 http://{IP}:8088/ironic-python-agent.ramdisk␣
↪|| goto retry_boot
boot
```

Note: Older versions of the iPXE ROM tend to misbehave on unreliable network connection, thus we use the timeout option with retries.

Just like with PXE, you can customize the list of collectors by appending the `ipa-inspection-collectors` kernel option. For example:

```
ipa-inspection-collectors=default,logs,extra_hardware
```

- Just as with PXE, youll need a PXE boot server. The configuration, however, will be different. Here is an example *dnsmasq.conf*:

```
port=0
interface={INTERFACE}
bind-interfaces
dhcp-range={DHCP IP RANGE, e.g. 192.168.0.50,192.168.0.150}
enable-tftp
tftp-root={TFTP ROOT, e.g. /tftpboot}
dhcp-sequential-ip
dhcp-match=ipxe,175
```

(continues on next page)

(continued from previous page)

```
dhcp-match=set:efi,option:client-arch,7
dhcp-match=set:efi,option:client-arch,9
dhcp-match=set:efi,option:client-arch,11
# dhcpv6.option: Client System Architecture Type (61)
dhcp-match=set:efi6,option6:61,0007
dhcp-match=set:efi6,option6:61,0009
dhcp-match=set:efi6,option6:61,0011
dhcp-userclass=set:ipxe6,iPXE
# Client is already running iPXE; move to next stage of chainloading
dhcp-boot=tag:ipxe,http://{IP}:8088/inspector.ipxe
# Client is PXE booting over EFI without iPXE ROM,
# send EFI version of iPXE chainloader
dhcp-boot=tag:efi,tag:!ipxe,ipxe.efi
dhcp-option=tag:efi6,tag:!ipxe6,option6:bootfile-url,tftp://{IP}/ipxe.
↪efi
# Client is running PXE over BIOS; send BIOS version of iPXE␣
↪chainloader
dhcp-boot=undionly.kpxe,localhost.localdomain,{IP}
```

First, we configure the same common parameters as with PXE. Then we define `ipxe` and `efi` tags for IPv4 and `ipxe6` and `efi6` for IPv6. Nodes already supporting iPXE are ordered to download and execute `inspector.ipxe`. Nodes without iPXE booted with UEFI will get `ipxe.efi` firmware to execute, while the remaining will get `undionly.kpxe`.

Configuring PXE for aarch64

For aarch64 Bare Metals, the PXE boot environment is basically the same as x86_64, youll need:

- TFTP server running and accessible (see below for using *dnsmasq*). Ensure `grubaa64.efi` is present in the TFTP root. The firmware can be retrieved from the installation distributions for aarch64.
- Copy `ironic-agent.kernel` and `ironic-agent.initramfs` to the TFTP root as well. Note that the ramdisk needs to be pre-built on an aarch64 machine with tools like `ironic-python-agent-builder`, see https://docs.openstack.org/ironic-python-agent-builder/latest/admin/dib.html for how to build ramdisk for aarch64.
- Next, setup `$TFTPROOT/EFI/BOOT/grub.cfg` as follows:

```
set default="1"
set timeout=5

menuentry 'Introspection for aarch64' {
    linux ironic-agent.kernel text showopts selinux=0 ipa-inspection-
↪callback-url=http://{IP}:5050/v1/continueăipa-inspection-
↪collectors=defaultăipa-collect-lldp=1ăsystemd.journald.forward_to_
↪console=no
    initrd ironic-agent.initramfs
}
```

  Replace `{IP}` with IP of the machine (do not use loopback interface, it will be accessed by ramdisk on a booting machine).

- Update DHCP options for aarch64, here is an example *dnsmasq.conf*:

```
port=0
interface={INTERFACE}
bind-interfaces
dhcp-range={DHCP IP RANGE, e.g. 192.168.0.50,192.168.0.150}
enable-tftp
dhcp-match=aarch64, option:client-arch, 11 # aarch64
dhcp-boot=tag:aarch64, grubaa64.efi
tftp-root={TFTP ROOT, e.g. /tftpboot}
dhcp-sequential-ip
```

Configuring PXE for Multi-arch

If the environment consists of bare metals with different architectures, normally different ramdisks are required for each architecture. The grub built-in variable grub_cpu could be used to locate the correct config file for each of them.

For example, setup `$TFTPROOT/EFI/BOOT/grub.cfg` as following:

```
set default=master
set timeout=5
set hidden_timeout_quiet=false

menuentry "master"  {
configfile /tftpboot/grub-${grub_cpu}.cfg
}
```

Prepare specific grub config for each existing architectures, e.g. `grub-arm64.cfg` for ARM64 and `grub-x86_64.cfg` for x86_64.

Update dnsmasq configuration to contain options for supported architectures.

3.1.4 Managing the ironic-inspector Database

ironic-inspector provides a command line client for managing its database. This client can be used for upgrading, and downgrading the database using alembic migrations.

If this is your first time running **ironic-inspector** to migrate the database, simply run:

```
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf␣
↪upgrade
```

If you have previously run a version of **ironic-inspector** earlier than 2.2.0, the safest thing is to delete the existing SQLite database and run `upgrade` as shown above. However, if you want to save the existing database, to ensure your database will work with the migrations, youll need to run an extra step before upgrading the database. You only need to do this the first time running version 2.2.0 or later.

If you are upgrading from **ironic-inspector** version 2.1.0 or lower:

```
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf␣
↪stamp --revision 578f84f38d
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf␣
↪upgrade
```

If you are upgrading from a git master install of the **ironic-inspector** after *rules* were introduced:

```
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf␣
↪stamp --revision d588418040d
ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf␣
↪upgrade
```

Other available commands can be discovered by running:

```
ironic-inspector-dbsync --help
```

3.1.5 Running

Running in standalone mode

Execute:

```
ironic-inspector --config-file /etc/ironic-inspector/inspector.conf
```

Running in non-standalone mode

API service can be started in development mode with:

```
ironic-inspector-api-wsgi -p 5050 -- --config-file /etc/ironic-inspector/
↪inspector.conf
```

For production, the ironic-inspector API service should be hosted under a web service. Below is a sample configuration for Apache with module mod_wsgi:

```
Listen 5050

<VirtualHost *:5050>
    WSGIDaemonProcess ironic-inspector user=stack group=stack threads=10␣
↪display-name=%{GROUP}
    WSGIScriptAlias / /usr/local/bin/ironic-inspector-api-wsgi

    SetEnv APACHE_RUN_USER stack
    SetEnv APACHE_RUN_GROUP stack
    WSGIProcessGroup ironic-inspector

    ErrorLog /var/log/apache2/ironic_inspector_error.log
    LogLevel info
    CustomLog /var/log/apache2/ironic_inspector_access.log combined

    <Directory /opt/stack/ironic-inspector/ironic_inspector/cmd>
        WSGIProcessGroup ironic-inspector
        WSGIApplicationGroup %{GLOBAL}
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

You can refer to ironic installation document for more guides.

ironic-inspector conductor can be started with:

```
ironic-inspector-conductor --config-file /etc/ironic-inspector/inspector.
↪conf
```

3.2 Configuration Guide

The ironic-inspector service operation is defined by a configuration file. The overview of configuration file options follow.

3.2.1 ironic-inspector.conf

DEFAULT

rpc_conn_pool_size

Type integer

Default `30`

Minimum Value 1

Size of RPC connection pool.

Table 1: Deprecated Variations

Group	Name
DEFAULT	rpc_conn_pool_size

conn_pool_min_size

Type integer

Default `2`

The pool size limit for connections expiration policy

conn_pool_ttl

Type integer

Default `1200`

The time-to-live in sec of idle connections in the pool

executor_thread_pool_size

Type integer

Default `64`

Size of executor thread pool when executor is threading or eventlet.

Table 2: Deprecated Variations

Group	Name
DEFAULT	rpc_thread_pool_size

rpc_response_timeout

Type integer

Default `60`

Seconds to wait for a response from a call.

`transport_url`

Type string

Default `rabbit://`

The network address and optional user credentials for connecting to the messaging backend, in URL format. The expected format is:

driver://[user:pass@]host:port[,[userN:passN@]hostN:portN]/virtual_host?query

Example: rabbit://rabbitmq:password@127.0.0.1:5672//

For full details on the fields in the URL see the documentation of oslo_messaging.TransportURL at https://docs.openstack.org/oslo.messaging/latest/reference/transport.html

`control_exchange`

Type string

Default `openstack`

The default exchange under which topics are scoped. May be overridden by an exchange name specified in the transport_url option.

`rpc_ping_enabled`

Type boolean

Default `False`

Add an endpoint to answer to ping calls. Endpoint is named oslo_rpc_server_ping

`backdoor_port`

Type string

Default `<None>`

Enable eventlet backdoor. Acceptable values are 0, <port>, and <start>:<end>, where 0 results in listening on a random tcp port number; <port> results in listening on the specified port number (and not enabling backdoor if that port is in use); and <start>:<end> results in listening on the smallest unused port number within the specified range of port numbers. The chosen port is displayed in the services log file.

`backdoor_socket`

Type string

Default `<None>`

Enable eventlet backdoor, using the provided path as a unix socket that can receive connections. This option is mutually exclusive with backdoor_port in that only one should be provided. If both are provided then the existence of this option overrides the usage of that option. Inside the path {pid} will be replaced with the PID of the current process.

`log_options`

Type boolean

Default `True`

Enables or disables logging values of all registered options when starting a service (at DEBUG level).

`graceful_shutdown_timeout`

Type integer

Default `60`

Specify a timeout after which a gracefully shutdown server will exit. Zero value means endless wait.

`api_paste_config`

Type string

Default `api-paste.ini`

File name for the paste.deploy config for api service

`wsgi_log_format`

Type string

Default `%(client_ip)s "%(request_line)s" status: %(status_code)s len: %(body_length)s time: %(wall_seconds).7f`

A python format string that is used as the template to generate log lines. The following values can beformatted into it: client_ip, date_time, request_line, status_code, body_length, wall_seconds.

`tcp_keepidle`

Type integer

Default `600`

Sets the value of TCP_KEEPIDLE in seconds for each server socket. Not supported on OS X.

`wsgi_default_pool_size`

Type integer

Default `100`

Size of the pool of greenthreads used by wsgi

`max_header_line`

Type integer

Default `16384`

Maximum line size of message headers to be accepted. max_header_line may need to be increased when using large tokens (typically those generated when keystone is configured to use PKI tokens with big service catalogs).

`wsgi_keep_alive`

Type boolean

Default `True`

If False, closes the client socket connection explicitly.

client_socket_timeout

Type integer

Default `900`

Timeout for client connections socket operations. If an incoming connection is idle for this number of seconds it will be closed. A value of 0 means wait forever.

wsgi_server_debug

Type boolean

Default `False`

True if the server should send exception tracebacks to the clients on 500 errors. If False, the server will respond with empty bodies.

debug

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

log_config_append

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 3: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

log_date_format

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

log_file

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 4: Deprecated Variations

Group	Name
DEFAULT	logfile

`log_dir`

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 5: Deprecated Variations

Group	Name
DEFAULT	logdir

`watch_log_file`

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

`use_syslog`

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

`use_journal`

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

`syslog_log_facility`

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

`use_json`

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

`use_stderr`

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

`use_eventlog`

Type boolean

Default `False`

Log output to Windows Event Log.

`log_rotate_interval`

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is setto interval.

`log_rotate_interval_type`

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

`max_logfile_count`

Type integer

Default `30`

Maximum number of rotated log files.

`max_logfile_size_mb`

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

`log_rotation_type`

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

`logging_context_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

`logging_default_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

`logging_debug_format_suffix`

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

`logging_exception_prefix`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

`logging_user_identity_format`

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

`default_log_levels`

Type list

Default `['sqlalchemy=WARNING', 'iso8601=WARNING', 'requests=WARNING', 'urllib3.connectionpool=WARNING',`

```
'keystonemiddleware=WARNING', 'keystoneauth=WARNING',
'ironicclient=WARNING', 'amqp=WARNING',
'amqplib=WARNING', 'oslo.messaging=WARNING',
'oslo_messaging=WARNING']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

`publish_errors`

Type boolean

Default `False`

Enables or disables publication of error events.

`instance_format`

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

`instance_uuid_format`

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

`rate_limit_interval`

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

`rate_limit_burst`

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

`rate_limit_except_level`

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

`fatal_deprecations`

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

listen_address

> **Type** string
>
> **Default** `::`

IP to listen on.

listen_port

> **Type** port number
>
> **Default** `5050`
>
> **Minimum Value** 0
>
> **Maximum Value** 65535

Port to listen on.

host

> **Type** string
>
> **Default** `localhost`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Name of this node. This can be an opaque identifier. It is not necessarily a hostname, FQDN, or IP address. However, the node name must be valid within an AMQP key, and if using ZeroMQ, a valid hostname, FQDN, or IP address.

auth_strategy

> **Type** string
>
> **Default** `keystone`
>
> **Valid Values** noauth, keystone, http_basic

Authentication method used on the ironic-inspector API. noauth, keystone or http_basic are valid options. noauth will disable all authentication.

Possible values

noauth no authentication

keystone use the Identity service for authentication

http_basic HTTP basic authentication

http_basic_auth_user_file

> **Type** string
>
> **Default** `/etc/ironic-inspector/htpasswd`

Path to Apache format user authentication file used when auth_strategy=http_basic

timeout

> **Type** integer

Default `3600`

Maximum Value 315576000

Timeout after which introspection is considered failed, set to 0 to disable.

`clean_up_period`

Type integer

Default `60`

Minimum Value 0

Amount of time in seconds, after which repeat clean up of timed out nodes and old nodes status information. WARNING: If set to a value of 0, then the periodic task is disabled and inspector will not sync with ironic to complete the internal clean-up process. Not advisable if the deployment uses a PXE filter, and will result in the ironic-inspector ceasing periodic cleanup activities.

`leader_election_interval`

Type integer

Default `10`

Interval (in seconds) between leader elections.

`use_ssl`

Type boolean

Default `False`

SSL Enabled/Disabled

`max_concurrency`

Type integer

Default `1000`

Minimum Value 2

The green thread pool size.

`introspection_delay`

Type integer

Default `5`

Delay (in seconds) between two introspections. Only applies when boot is managed by ironic-inspector (i.e. manage_boot==True).

`ipmi_address_fields`

Type list

Default `['redfish_address', 'ilo_address', 'drac_host', 'drac_address']`

Ironic driver_info fields that are equivalent to ipmi_address.

`rootwrap_config`

Type string

Default `/etc/ironic-inspector/rootwrap.conf`

Path to the rootwrap configuration file to use for running commands as root

`api_max_limit`

Type integer

Default `1000`

Minimum Value 1

Limit the number of elements an API list-call returns

`can_manage_boot`

Type boolean

Default `True`

Whether the current installation of ironic-inspector can manage PXE booting of nodes. If set to False, the API will reject introspection requests with manage_boot missing or set to True.

`enable_mdns`

Type boolean

Default `False`

Whether to enable publishing the ironic-inspector API endpoint via multicast DNS.

`standalone`

Type boolean

Default `True`

Whether to run ironic-inspector as a standalone service. Its EXPERIMENTAL to set to False.

capabilities

`boot_mode`

Type boolean

Default `False`

Whether to store the boot mode (BIOS or UEFI).

`cpu_flags`

Type dict

Default `{'vmx': 'cpu_vt', 'svm': 'cpu_vt', 'aes': 'cpu_aes', 'pse': 'cpu_hugepages', 'pdpe1gb': 'cpu_hugepages_1g', 'smx': 'cpu_txt'}`

Mapping between a CPU flag and a capability to set if this flag is present.

coordination

`backend_url`

Type string

Default `memcached://localhost:11211`

The backend URL to use for distributed coordination. EXPERIMENTAL.

cors

`allowed_origin`

Type list

Default `<None>`

Indicate whether this resource may be shared with the domain received in the requests origin header. Format: <protocol>://<host>[:<port>], no trailing slash. Example: https://horizon.example.com

`allow_credentials`

Type boolean

Default `True`

Indicate that the actual request can include user credentials

`expose_headers`

Type list

Default `[]`

Indicate which headers are safe to expose to the API. Defaults to HTTP Simple Headers.

`max_age`

Type integer

Default `3600`

Maximum cache age of CORS preflight requests.

`allow_methods`

Type list

Default `['GET', 'POST', 'PUT', 'HEAD', 'PATCH', 'DELETE', 'OPTIONS']`

Indicate which methods can be used during the actual request.

`allow_headers`

Type list

Default `['X-Auth-Token', 'X-OpenStack-Ironic-Inspector-API-Minimum-Version 'X-OpenStack-Ironic-Inspector-API-Maximum-Version', 'X-OpenStack-Ironic-Inspector-API-Version']`

Indicate which header field names may be used during the actual request.

database

sqlite_synchronous

Type boolean

Default `True`

If True, SQLite uses synchronous mode.

Table 6: Deprecated Variations

Group	Name
DEFAULT	sqlite_synchronous

backend

Type string

Default `sqlalchemy`

The back end to use for the database.

Table 7: Deprecated Variations

Group	Name
DEFAULT	db_backend

connection

Type string

Default `<None>`

The SQLAlchemy connection string to use to connect to the database.

Table 8: Deprecated Variations

Group	Name
DEFAULT	sql_connection
DATABASE	sql_connection
sql	connection

slave_connection

Type string

Default `<None>`

The SQLAlchemy connection string to use to connect to the slave database.

mysql_sql_mode

Type string

Default `TRADITIONAL`

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

mysql_enable_ndb

Type boolean

Default `False`

If True, transparently enables support for handling MySQL Cluster (NDB).

connection_recycle_time

Type integer

Default `3600`

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

Table 9: Deprecated Variations

Group	Name
DATABASE	idle_timeout
database	idle_timeout
DEFAULT	sql_idle_timeout
DATABASE	sql_idle_timeout
sql	idle_timeout

max_pool_size

Type integer

Default `5`

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

Table 10: Deprecated Variations

Group	Name
DEFAULT	sql_max_pool_size
DATABASE	sql_max_pool_size

max_retries

Type integer

Default `10`

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

Table 11: Deprecated Variations

Group	Name
DEFAULT	sql_max_retries
DATABASE	sql_max_retries

retry_interval

Type integer

Default `10`

Interval between retries of opening a SQL connection.

Table 12: Deprecated Variations

Group	Name
DEFAULT	sql_retry_interval
DATABASE	reconnect_interval

max_overflow

Type integer

Default `50`

If set, use this value for max_overflow with SQLAlchemy.

Table 13: Deprecated Variations

Group	Name
DEFAULT	sql_max_overflow
DATABASE	sqlalchemy_max_overflow

connection_debug

Type integer

Default `0`

Minimum Value 0

Maximum Value 100

Verbosity of SQL debugging information: 0=None, 100=Everything.

Table 14: Deprecated Variations

Group	Name
DEFAULT	sql_connection_debug

connection_trace

Type boolean

Default `False`

Add Python stack traces to SQL as comment strings.

Table 15: Deprecated Variations

Group	Name
DEFAULT	sql_connection_trace

pool_timeout

Type integer

Default `<None>`

If set, use this value for pool_timeout with SQLAlchemy.

Table 16: Deprecated Variations

Group	Name
DATABASE	sqlalchemy_pool_timeout

use_db_reconnect

Type boolean

Default `False`

Enable the experimental use of database reconnect on connection lost.

db_retry_interval

Type integer

Default `1`

Seconds between retries of a database transaction.

db_inc_retry_interval

Type boolean

Default `True`

If True, increases the interval between retries of a database operation up to db_max_retry_interval.

db_max_retry_interval

Type integer

Default `10`

If db_inc_retry_interval is set, the maximum seconds between retries of a database operation.

db_max_retries

Type integer

Default `20`

Maximum retries in case of connection error or deadlock error before error is raised. Set to -1 to specify an infinite retry count.

connection_parameters

Type string

Default `''`

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&

discovery

enroll_node_driver

Type string

Default `fake-hardware`

The name of the Ironic driver used by the enroll hook when creating a new node in Ironic.

enroll_node_fields

Type dict

Default `{}`

Additional fields to set on newly discovered nodes.

enabled_bmc_address_version

Type list

Default `['4', '6']`

IP version of BMC address that will be used when enrolling a new node in Ironic. Defaults to 4,6. Could be 4 (use v4 address only), 4,6 (v4 address have higher priority and if both addresses found v6 version is ignored), 6,4 (v6 is desired but fall back to v4 address for BMCs having v4 address, opposite to 4,6), 6 (use v6 address only and ignore v4 version).

dnsmasq_pxe_filter

dhcp_hostsdir

Type string

Default `/var/lib/ironic-inspector/dhcp-hostsdir`

The MAC address cache directory, exposed to dnsmasq.This directory is expected to be in exclusive control of the driver.

purge_dhcp_hostsdir

Type boolean

Default `True`

Purge the hostsdir upon driver initialization. Setting to false should only be performed when the deployment of inspector is such that there are multiple processes executing inside of the same host and namespace. In this case, the Operator is responsible for setting up a custom cleaning facility.

dnsmasq_start_command

Type string

Default `''`

A (shell) command line to start the dnsmasq service upon filter initialization. Default: dont start.

dnsmasq_stop_command

Type string

Default `''`

A (shell) command line to stop the dnsmasq service upon inspector (error) exit. Default: dont stop.

extra_hardware

`strict`

Type boolean

Default `False`

If True, refuse to parse extra data if at least one record is too short. Additionally, remove the incoming data even if parsing failed.

iptables

`dnsmasq_interface`

Type string

Default `br-ctlplane`

Interface on which dnsmasq listens, the default is for VMs.

`firewall_chain`

Type string

Default `ironic-inspector`

iptables chain name to use.

`ethoib_interfaces`

Type list

Default `[]`

List of Ethernet Over InfiniBand interfaces on the Inspector host which are used for physical access to the DHCP network. Multiple interfaces would be attached to a bond or bridge specified in dnsmasq_interface. The MACs of the InfiniBand nodes which are not in desired state are going to be blacklisted based on the list of neighbor MACs on these interfaces.

`ip_version`

Type string

Default `4`

Valid Values 4, 6

The IP version that will be used for iptables filter. Defaults to 4.

Possible values

4 IPv4

6 IPv6

ironic

`auth_url`

Type unknown type

Default `<None>`

Authentication URL

`auth_type`

Type unknown type

Default `<None>`

Authentication type to load

Table 17: Deprecated Variations

Group	Name
ironic	auth_plugin

`cafile`

Type string

Default `<None>`

PEM encoded Certificate Authority to use when verifying HTTPs connections.

`certfile`

Type string

Default `<None>`

PEM encoded client certificate cert file

`collect_timing`

Type boolean

Default `False`

Collect per-API call timing information.

`connect_retries`

Type integer

Default `<None>`

The maximum number of retries that should be attempted for connection errors.

`connect_retry_delay`

Type floating point

Default `<None>`

Delay (in seconds) between two retries for connection errors. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

`default_domain_id`

Type unknown type

Default `<None>`

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`default_domain_name`

Type unknown type

Default `<None>`

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`domain_id`

Type unknown type

Default `<None>`

Domain ID to scope to

`domain_name`

Type unknown type

Default `<None>`

Domain name to scope to

`endpoint_override`

Type string

Default `<None>`

Always use this endpoint URL for requests for this client. NOTE: The unversioned endpoint should be specified here; to request a particular API version, use the *version*, *min-version*, and/or *max-version* options.

`insecure`

Type boolean

Default `False`

Verify HTTPS connections.

`keyfile`

Type string

Default `<None>`

PEM encoded client certificate key file

max_version

Type string

Default `<None>`

The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

max_retries

Type integer

Default `30`

Maximum number of retries in case of conflict error (HTTP 409).

min_version

Type string

Default `<None>`

The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

password

Type unknown type

Default `<None>`

Users password

project_domain_id

Type unknown type

Default `<None>`

Domain ID containing project

project_domain_name

Type unknown type

Default `<None>`

Domain name containing project

project_id

Type unknown type

Default `<None>`

Project ID to scope to

Table 18: Deprecated Variations

Group	Name
ironic	tenant-id
ironic	tenant_id

`project_name`

Type unknown type

Default `<None>`

Project name to scope to

Table 19: Deprecated Variations

Group	Name
ironic	tenant-name
ironic	tenant_name

`region_name`

Type string

Default `<None>`

The default region_name for endpoint URL discovery.

`retry_interval`

Type integer

Default `2`

Interval between retries in case of conflict error (HTTP 409).

`service_name`

Type string

Default `<None>`

The default service_name for endpoint URL discovery.

`service_type`

Type string

Default `baremetal`

The default service_type for endpoint URL discovery.

`split_loggers`

Type boolean

Default `False`

Log requests to multiple loggers.

`status_code_retries`

Type integer

Default `<None>`

The maximum number of retries that should be attempted for retriable HTTP status codes.

`status_code_retry_delay`

Type floating point

Default `<None>`

Delay (in seconds) between two retries for retriable status codes. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

system_scope

Type unknown type

Default `<None>`

Scope for system operations

tenant_id

Type unknown type

Default `<None>`

Tenant ID

tenant_name

Type unknown type

Default `<None>`

Tenant Name

timeout

Type integer

Default `<None>`

Timeout value for http requests

trust_id

Type unknown type

Default `<None>`

Trust ID

user_domain_id

Type unknown type

Default `<None>`

Users domain id

user_domain_name

Type unknown type

Default `<None>`

Users domain name

user_id

Type unknown type

Default `<None>`

User id

username

Type unknown type

Default `<None>`

Username

Table 20: Deprecated Variations

Group	Name
ironic	user-name
ironic	user_name

valid_interfaces

Type list

Default `['internal', 'public']`

List of interfaces, in order of preference, for endpoint URL.

version

Type string

Default `<None>`

Minimum Major API version within a given Major API version for endpoint URL discovery. Mutually exclusive with min_version and max_version

keystone_authtoken

www_authenticate_uri

Type string

Default `<None>`

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 21: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

auth_uri

Type string

Default `<None>`

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

> **Warning:** This option is deprecated for removal since Queens. Its value may be silently ignored in the future.
>
> **Reason** The auth_uri option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

`auth_version`

Type string

Default `<None>`

API version of the Identity API endpoint.

`interface`

Type string

Default `internal`

Interface to use for the Identity API endpoint. Valid values are public, internal (default) or admin.

`delay_auth_decision`

Type boolean

Default `False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

`http_connect_timeout`

Type integer

Default `<None>`

Request timeout value for communicating with Identity API server.

`http_request_max_retries`

Type integer

Default `3`

How many times are we trying to reconnect when communicating with Identity API Server.

`cache`

Type string

Default `<None>`

Request environment key where the Swift cache object is stored. When auth_token middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

certfile

Type string

Default `<None>`

Required if identity server requires client certificate

keyfile

Type string

Default `<None>`

Required if identity server requires client certificate

cafile

Type string

Default `<None>`

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

insecure

Type boolean

Default `False`

Verify HTTPS connections.

region_name

Type string

Default `<None>`

The region in which the identity server can be found.

memcached_servers

Type list

Default `<None>`

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 22: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

token_cache_time

Type integer

Default `300`

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

memcache_security_strategy

Type string

Default `None`

Valid Values None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, auth_token will raise an exception on initialization.

`memcache_secret_key`

Type string

Default `<None>`

(Optional, mandatory if memcache_security_strategy is defined) This string is used for key derivation.

`memcache_pool_dead_retry`

Type integer

Default `300`

(Optional) Number of seconds memcached server is considered dead before it is tried again.

`memcache_pool_maxsize`

Type integer

Default `10`

(Optional) Maximum total number of open connections to every memcached server.

`memcache_pool_socket_timeout`

Type integer

Default `3`

(Optional) Socket timeout in seconds for communicating with a memcached server.

`memcache_pool_unused_timeout`

Type integer

Default `60`

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

`memcache_pool_conn_get_timeout`

Type integer

Default `10`

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

`memcache_use_advanced_pool`

Type boolean

Default `False`

(Optional) Use the advanced (eventlet safe) memcached client pool. The advanced pool will only work under python 2.x.

`include_service_catalog`

Type boolean

Default `True`

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

`enforce_token_bind`

Type string

Default `permissive`

Used to control the use and type of token binding. Can be set to: disabled to not check token binding. permissive (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. strict like permissive but if the bind type is unknown the token will be rejected. required any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

`service_token_roles`

Type list

Default `['service']`

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

`service_token_roles_required`

Type boolean

Default `False`

For backwards compatibility reasons we must let valid service tokens pass that dont pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

`service_type`

Type string

Default `<None>`

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

`auth_type`

Type unknown type

Default `<None>`

Authentication type to load

Table 23: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

auth_section

Type unknown type

Default `<None>`

Config Section from which to load plugin specific options

mdns

registration_attempts

Type integer

Default `5`

Minimum Value 1

Number of attempts to register a service. Currently has to be larger than 1 because of race conditions in the zeroconf library.

lookup_attempts

Type integer

Default `3`

Minimum Value 1

Number of attempts to lookup a service.

params

Type unknown type

Default `{}`

Additional parameters to pass for the registered service.

interfaces

Type list

Default `<None>`

List of IP addresses of interfaces to use for mDNS. Defaults to all interfaces on the system.

oslo_messaging_amqp

`container_name`

Type string

Default `<None>`

Name for the AMQP container. must be globally unique. Defaults to a generated UUID

Table 24: Deprecated Variations

Group	Name
amqp1	container_name

`idle_timeout`

Type integer

Default `0`

Timeout for inactive connections (in seconds)

Table 25: Deprecated Variations

Group	Name
amqp1	idle_timeout

`trace`

Type boolean

Default `False`

Debug: dump AMQP frames to stdout

Table 26: Deprecated Variations

Group	Name
amqp1	trace

`ssl`

Type boolean

Default `False`

Attempt to connect via SSL. If no other ssl-related parameters are given, it will use the systems CA-bundle to verify the servers certificate.

`ssl_ca_file`

Type string

Default `''`

CA certificate PEM file used to verify the servers certificate

Table 27: Deprecated Variations

Group	Name
amqp1	ssl_ca_file

ssl_cert_file

Type string

Default `''`

Self-identifying certificate PEM file for client authentication

Table 28: Deprecated Variations

Group	Name
amqp1	ssl_cert_file

ssl_key_file

Type string

Default `''`

Private key PEM file used to sign ssl_cert_file certificate (optional)

Table 29: Deprecated Variations

Group	Name
amqp1	ssl_key_file

ssl_key_password

Type string

Default `<None>`

Password for decrypting ssl_key_file (if encrypted)

Table 30: Deprecated Variations

Group	Name
amqp1	ssl_key_password

ssl_verify_vhost

Type boolean

Default `False`

By default SSL checks that the name in the servers certificate matches the hostname in the transport_url. In some configurations it may be preferable to use the virtual hostname instead, for example if the server uses the Server Name Indication TLS extension (rfc6066) to provide a certificate per virtual host. Set ssl_verify_vhost to True if the servers SSL certificate uses the virtual host name instead of the DNS name.

sasl_mechanisms

Type string

Default `''`

Space separated list of acceptable SASL mechanisms

Table 31: Deprecated Variations

Group	Name
amqp1	sasl_mechanisms

sasl_config_dir

Type string

Default `''`

Path to directory that contains the SASL configuration

Table 32: Deprecated Variations

Group	Name
amqp1	sasl_config_dir

sasl_config_name

Type string

Default `''`

Name of configuration file (without .conf suffix)

Table 33: Deprecated Variations

Group	Name
amqp1	sasl_config_name

sasl_default_realm

Type string

Default `''`

SASL realm to use if no realm present in username

connection_retry_interval

Type integer

Default `1`

Minimum Value 1

Seconds to pause before attempting to re-connect.

connection_retry_backoff

Type integer

Default `2`

Minimum Value 0

Increase the connection_retry_interval by this many seconds after each unsuccessful failover attempt.

`connection_retry_interval_max`

Type integer

Default `30`

Minimum Value 1

Maximum limit for connection_retry_interval + connection_retry_backoff

`link_retry_delay`

Type integer

Default `10`

Minimum Value 1

Time to pause between re-connecting an AMQP 1.0 link that failed due to a recoverable error.

`default_reply_retry`

Type integer

Default `0`

Minimum Value -1

The maximum number of attempts to re-send a reply message which failed due to a recoverable error.

`default_reply_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for an rpc reply message delivery.

`default_send_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for an rpc cast or call message delivery. Only used when caller does not provide a timeout expiry.

`default_notify_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for a sent notification message delivery. Only used when caller does not provide a timeout expiry.

`default_sender_link_timeout`

Type integer

Default `600`

Minimum Value 1

The duration to schedule a purge of idle sender links. Detach link after expiry.

addressing_mode

Type string

Default `dynamic`

Indicates the addressing mode used by the driver. Permitted values: legacy - use legacy non-routable addressing routable - use routable addresses dynamic - use legacy addresses if the message bus does not support routing otherwise use routable addressing

pseudo_vhost

Type boolean

Default `True`

Enable virtual host support for those message buses that do not natively support virtual hosting (such as qpidd). When set to true the virtual host name will be added to all message bus addresses, effectively creating a private subnet per virtual host. Set to False if the message bus supports virtual hosting using the hostname field in the AMQP 1.0 Open performative as the name of the virtual host.

server_request_prefix

Type string

Default `exclusive`

address prefix used when sending to a specific server

Table 34: Deprecated Variations

Group	Name
amqp1	server_request_prefix

broadcast_prefix

Type string

Default `broadcast`

address prefix used when broadcasting to all servers

Table 35: Deprecated Variations

Group	Name
amqp1	broadcast_prefix

group_request_prefix

Type string

Default `unicast`

address prefix when sending to any server in group

Table 36: Deprecated Variations

Group	Name
amqp1	group_request_prefix

`rpc_address_prefix`

Type string

Default `openstack.org/om/rpc`

Address prefix for all generated RPC addresses

`notify_address_prefix`

Type string

Default `openstack.org/om/notify`

Address prefix for all generated Notification addresses

`multicast_address`

Type string

Default `multicast`

Appended to the address prefix when sending a fanout message. Used by the message bus to identify fanout messages.

`unicast_address`

Type string

Default `unicast`

Appended to the address prefix when sending to a particular RPC/Notification server. Used by the message bus to identify messages sent to a single destination.

`anycast_address`

Type string

Default `anycast`

Appended to the address prefix when sending to a group of consumers. Used by the message bus to identify messages that should be delivered in a round-robin fashion across consumers.

`default_notification_exchange`

Type string

Default `<None>`

Exchange name used in notification addresses. Exchange name resolution precedence: Target.exchange if set else default_notification_exchange if set else control_exchange if set else notify

`default_rpc_exchange`

Type string

Default `<None>`

Exchange name used in RPC addresses. Exchange name resolution precedence: Target.exchange if set else default_rpc_exchange if set else control_exchange if set else rpc

`reply_link_credit`

Type integer

Default `200`

Minimum Value 1

Window size for incoming RPC Reply messages.

`rpc_server_credit`

Type integer

Default `100`

Minimum Value 1

Window size for incoming RPC Request messages

`notify_server_credit`

Type integer

Default `100`

Minimum Value 1

Window size for incoming Notification messages

`pre_settled`

Type multi-valued

Default `rpc-cast`

Default `rpc-reply`

Send messages of this type pre-settled. Pre-settled messages will not receive acknowledgement from the peer. Note well: pre-settled messages may be silently discarded if the delivery fails. Permitted values: rpc-call - send RPC Calls pre-settled rpc-reply- send RPC Replies pre-settled rpc-cast - Send RPC Casts pre-settled notify - Send Notifications pre-settled

oslo_messaging_kafka

`kafka_max_fetch_bytes`

Type integer

Default `1048576`

Max fetch bytes of Kafka consumer

`kafka_consumer_timeout`

Type floating point

Default `1.0`

Default timeout(s) for Kafka consumers

`pool_size`

Type integer

Default `10`

Pool Size for Kafka Consumers

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

conn_pool_min_size

Type integer

Default `2`

The pool size limit for connections expiration policy

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

conn_pool_ttl

Type integer

Default `1200`

The time-to-live in sec of idle connections in the pool

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

consumer_group

Type string

Default `oslo_messaging_consumer`

Group id for Kafka consumer. Consumers in one group will coordinate message consumption

producer_batch_timeout

Type floating point

Default `0.0`

Upper bound on the delay for KafkaProducer batching in seconds

producer_batch_size

Type integer

Default `16384`

Size of batch for the producer async send

`compression_codec`

Type string

Default `none`

Valid Values none, gzip, snappy, lz4, zstd

The compression codec for all data generated by the producer. If not set, compression will not be used. Note that the allowed values of this depend on the kafka version

`enable_auto_commit`

Type boolean

Default `False`

Enable asynchronous consumer commits

`max_poll_records`

Type integer

Default `500`

The maximum number of records returned in a poll call

`security_protocol`

Type string

Default `PLAINTEXT`

Valid Values PLAINTEXT, SASL_PLAINTEXT, SSL, SASL_SSL

Protocol used to communicate with brokers

`sasl_mechanism`

Type string

Default `PLAIN`

Mechanism when security protocol is SASL

`ssl_cafile`

Type string

Default `''`

CA certificate PEM file used to verify the server certificate

`ssl_client_cert_file`

Type string

Default `''`

Client certificate PEM file used for authentication.

`ssl_client_key_file`

Type string

Default `''`

Client key PEM file used for authentication.

ssl_client_key_password

Type string

Default `''`

Client key password file used for authentication.

oslo_messaging_notifications

driver

Type multi-valued

Default `''`

The Drivers(s) to handle sending notifications. Possible values are messaging, messagingv2, routing, log, test, noop

Table 37: Deprecated Variations

Group	Name
DEFAULT	notification_driver

transport_url

Type string

Default `<None>`

A URL representing the messaging driver to use for notifications. If not set, we fall back to the same configuration used for RPC.

Table 38: Deprecated Variations

Group	Name
DEFAULT	notification_transport_url

topics

Type list

Default `['notifications']`

AMQP topic used for OpenStack notifications.

Table 39: Deprecated Variations

Group	Name
rpc_notifier2	topics
DEFAULT	notification_topics

retry

Type integer

Default `-1`

The maximum number of attempts to re-send a notification message which failed to be delivered due to a recoverable error. 0 - No retry, -1 - indefinite

oslo_messaging_rabbit

`amqp_durable_queues`

Type boolean

Default `False`

Use durable queues in AMQP.

`amqp_auto_delete`

Type boolean

Default `False`

Auto-delete queues in AMQP.

Table 40: Deprecated Variations

Group	Name
DEFAULT	amqp_auto_delete

`ssl`

Type boolean

Default `False`

Connect over SSL.

Table 41: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_use_ssl

`ssl_version`

Type string

Default `''`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

Table 42: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_version

`ssl_key_file`

Type string

Default `''`

SSL key file (valid only if SSL enabled).

Table 43: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_keyfile

ssl_cert_file

Type string

Default `''`

SSL cert file (valid only if SSL enabled).

Table 44: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_certfile

ssl_ca_file

Type string

Default `''`

SSL certification authority file (valid only if SSL enabled).

Table 45: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_ca_certs

heartbeat_in_pthread

Type boolean

Default `False`

EXPERIMENTAL: Run the health check heartbeat thread through a native python thread. By default if this option isnt provided the health check heartbeat will inherit the execution model from the parent process. By example if the parent process have monkey patched the stdlib by using eventlet/greenlet then the heartbeat will be run through a green thread.

kombu_reconnect_delay

Type floating point

Default `1.0`

How long to wait before reconnecting in response to an AMQP consumer cancel notification.

Table 46: Deprecated Variations

Group	Name
DEFAULT	kombu_reconnect_delay

kombu_compression

Type string

Default `<None>`

EXPERIMENTAL: Possible values are: gzip, bz2. If not set compression will not be used. This option may not be available in future versions.

`kombu_missing_consumer_retry_timeout`

Type integer

Default `60`

How long to wait a missing client before abandoning to send it its replies. This value should not be longer than rpc_response_timeout.

Table 47: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_reconnect_timeout

`kombu_failover_strategy`

Type string

Default `round-robin`

Valid Values round-robin, shuffle

Determines how the next RabbitMQ node is chosen in case the one we are currently connected to becomes unavailable. Takes effect only if more than one RabbitMQ node is provided in config.

`rabbit_login_method`

Type string

Default `AMQPLAIN`

Valid Values PLAIN, AMQPLAIN, RABBIT-CR-DEMO

The RabbitMQ login method.

Table 48: Deprecated Variations

Group	Name
DEFAULT	rabbit_login_method

`rabbit_retry_interval`

Type integer

Default `1`

How frequently to retry connecting with RabbitMQ.

`rabbit_retry_backoff`

Type integer

Default `2`

How long to backoff for between retries when connecting to RabbitMQ.

Table 49: Deprecated Variations

Group	Name
DEFAULT	rabbit_retry_backoff

rabbit_interval_max

Type integer

Default `30`

Maximum interval of RabbitMQ connection retries. Default is 30 seconds.

rabbit_ha_queues

Type boolean

Default `False`

Try to use HA queues in RabbitMQ (x-ha-policy: all). If you change this option, you must wipe the RabbitMQ database. In RabbitMQ 3.0, queue mirroring is no longer controlled by the x-ha-policy argument when declaring a queue. If you just want to make sure that all queues (except those with auto-generated names) are mirrored across all nodes, run: rabbitmqctl set_policy HA ^(?!amq.).* {ha-mode: all}

Table 50: Deprecated Variations

Group	Name
DEFAULT	rabbit_ha_queues

rabbit_transient_queues_ttl

Type integer

Default `1800`

Minimum Value 1

Positive integer representing duration in seconds for queue TTL (x-expires). Queues which are unused for the duration of the TTL are automatically deleted. The parameter affects only reply and fanout queues.

rabbit_qos_prefetch_count

Type integer

Default `0`

Specifies the number of messages to prefetch. Setting to zero allows unlimited messages.

heartbeat_timeout_threshold

Type integer

Default `60`

Number of seconds after which the Rabbit broker is considered down if heartbeats keep-alive fails (0 disables heartbeat).

heartbeat_rate

Type integer

Default `2`

How often times during the heartbeat_timeout_threshold we check the heartbeat.

`direct_mandatory_flag`

Type boolean

Default `True`

(DEPRECATED) Enable/Disable the RabbitMQ mandatory flag for direct send. The direct send is used as reply, so the MessageUndeliverable exception is raised in case the client queue does not exist.MessageUndeliverable exception will be used to loop for a timeout to lets a chance to sender to recover.This flag is deprecated and it will not be possible to deactivate this functionality anymore

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

Reason Mandatory flag no longer deactivable.

`enable_cancel_on_failover`

Type boolean

Default `False`

Enable x-cancel-on-ha-failover flag so that rabbitmq server will cancel and notify consumerswhen queue is down

oslo_policy

`enforce_scope`

Type boolean

Default `False`

This option controls whether or not to enforce scope when evaluating policies. If `True`, the scope of the token used in the request is compared to the `scope_types` of the policy being enforced. If the scopes do not match, an `InvalidScope` exception will be raised. If `False`, a message will be logged informing operators that policies are being invoked with mismatching scope.

`enforce_new_defaults`

Type boolean

Default `False`

This option controls whether or not to use old deprecated defaults when evaluating policies. If `True`, the old deprecated defaults are not going to be evaluated. This means if any existing token is allowed for old defaults but is disallowed for new defaults, it will be disallowed. It is encouraged to enable this flag along with the `enforce_scope` flag so that you can get the benefits of new defaults and `scope_type` together

`policy_file`

Type string

Default `policy.json`

The relative or absolute path of a file that maps roles to permissions for a given service. Relative paths must be specified in relation to the configuration file setting this option.

Table 51: Deprecated Variations

Group	Name
DEFAULT	policy_file

`policy_default_rule`

Type string

Default `default`

Default rule. Enforced when a requested rule is not found.

Table 52: Deprecated Variations

Group	Name
DEFAULT	policy_default_rule

`policy_dirs`

Type multi-valued

Default `policy.d`

Directories where policy configuration files are stored. They can be relative to any directory in the search path defined by the config_dir option, or absolute paths. The file defined by policy_file must exist for these directories to be searched. Missing or empty directories are ignored.

Table 53: Deprecated Variations

Group	Name
DEFAULT	policy_dirs

`remote_content_type`

Type string

Default `application/x-www-form-urlencoded`

Valid Values application/x-www-form-urlencoded, application/json

Content Type to send and receive data for REST based policy check

`remote_ssl_verify_server_crt`

Type boolean

Default `False`

server identity verification for REST based policy check

`remote_ssl_ca_crt_file`

Type string

Default `<None>`

Absolute path to ca cert file for REST based policy check

remote_ssl_client_crt_file

Type string

Default `<None>`

Absolute path to client cert for REST based policy check

remote_ssl_client_key_file

Type string

Default `<None>`

Absolute path client key file REST based policy check

pci_devices

alias

Type multi-valued

Default `''`

An alias for PCI device identified by vendor_id and product_id fields. Format: {vendor_id: 1234, product_id: 5678, name: pci_dev1}

port_physnet

cidr_map

Type list

Default `10.10.10.0/24:physnet_a,2001:db8::/64:physnet_b`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Mapping of IP subnet CIDR to physical network. When the physnet_cidr_map processing hook is enabled the physical_network property of baremetal ports is populated based on this mapping.

processing

add_ports

Type string

Default `pxe`

Valid Values all, active, pxe, disabled

Which MAC addresses to add as ports during introspection. Possible values: all (all MAC addresses), active (MAC addresses of NIC with IP addresses), pxe (only MAC address of NIC node PXE booted from, falls back to active if PXE MAC is not supplied by the ramdisk).

keep_ports

Type string

Default `all`

Valid Values all, present, added

Which ports (already present on a node) to keep after introspection. Possible values: all (do not delete anything), present (keep ports which MACs were present in introspection data), added (keep only MACs that we added during introspection).

overwrite_existing

Type boolean

Default `True`

Whether to overwrite existing values in node database. Disable this option to make introspection a non-destructive operation.

default_processing_hooks

Type string

Default `ramdisk_error,root_disk_selection,scheduler,validate_interfaces,capabilities,pci_devices`

Comma-separated list of default hooks for processing pipeline. Hook scheduler updates the node with the minimum properties required by the Nova scheduler. Hook validate_interfaces ensures that valid NIC data was provided by the ramdisk. Do not exclude these two unless you really know what youre doing.

processing_hooks

Type string

Default `$default_processing_hooks`

Comma-separated list of enabled hooks for processing pipeline. The default for this is $default_processing_hooks, hooks can be added before or after the defaults like this: prehook,$default_processing_hooks,posthook.

ramdisk_logs_dir

Type string

Default `<None>`

If set, logs from ramdisk will be stored in this directory.

always_store_ramdisk_logs

Type boolean

Default `False`

Whether to store ramdisk logs even if it did not return an error message (dependent upon ramdisk_logs_dir option being set).

node_not_found_hook

Type string

Default `<None>`

The name of the hook to run when inspector receives inspection information from a node it isnt already aware of. This hook is ignored by default.

store_data

Type string

Default `none`

The storage backend for storing introspection data. Possible values are: none, database and swift. If set to none, introspection data will not be stored.

disk_partitioning_spacing

Type boolean

Default `True`

Whether to leave 1 GiB of disk size untouched for partitioning. Only has effect when used with the IPA as a ramdisk, for older ramdisk local_gb is calculated on the ramdisk side.

ramdisk_logs_filename_format

Type string

Default `{uuid}_{dt:%Y%m%d-%H%M%S.%f}.tar.gz`

File name template for storing ramdisk logs. The following replacements can be used: {uuid} - node UUID or unknown, {bmc} - node BMC address or unknown, {dt} - current UTC date and time, {mac} - PXE booting MAC or unknown.

power_off

Type boolean

Default `True`

Whether to power off a node after introspection.Nodes in active or rescue states which submit introspection data will be left on if the feature is enabled via the permit_active_introspection configuration option.

permit_active_introspection

Type boolean

Default `False`

Whether to process nodes that are in running states.

update_pxe_enabled

Type boolean

Default `True`

Whether to update the pxe_enabled value according to the introspection data. This option has no effect if [processing]overwrite_existing is set to False

pxe_filter

driver

Type string

Default `iptables`

PXE boot filter driver to use, possible filters are: iptables, dnsmasq and noop. Set noop to disable the firewall filtering.

sync_period

Type integer

Default `15`

Minimum Value 0

Amount of time in seconds, after which repeat periodic update of the filter.

deny_unknown_macs

Type boolean

Default `False`

By default inspector will open the DHCP server for any node when introspection is active. Opening DHCP for unknown MAC addresses when introspection is active allow for users to add nodes with no ports to ironic and have ironic-inspector enroll ports based on node introspection results. NOTE: If this option is True, nodes must have at least one enrolled port prior to introspection.

service_catalog

auth_url

Type unknown type

Default `<None>`

Authentication URL

auth_type

Type unknown type

Default `<None>`

Authentication type to load

Table 54: Deprecated Variations

Group	Name
service_catalog	auth_plugin

cafile

Type string

Default `<None>`

PEM encoded Certificate Authority to use when verifying HTTPs connections.

certfile

Type string

Default `<None>`

PEM encoded client certificate cert file

collect_timing

Type boolean

Default `False`

Collect per-API call timing information.

connect_retries

Type integer

Default `<None>`

The maximum number of retries that should be attempted for connection errors.

connect_retry_delay

Type floating point

Default `<None>`

Delay (in seconds) between two retries for connection errors. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

default_domain_id

Type unknown type

Default `<None>`

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

default_domain_name

Type unknown type

Default `<None>`

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

domain_id

Type unknown type

Default `<None>`

Domain ID to scope to

domain_name

Type unknown type

Default `<None>`

Domain name to scope to

`endpoint_override`

> **Type** string
>
> **Default** `<None>`

Always use this endpoint URL for requests for this client. NOTE: The unversioned endpoint should be specified here; to request a particular API version, use the *version*, *min-version*, and/or *max-version* options.

`insecure`

> **Type** boolean
>
> **Default** `False`

Verify HTTPS connections.

`keyfile`

> **Type** string
>
> **Default** `<None>`

PEM encoded client certificate key file

`max_version`

> **Type** string
>
> **Default** `<None>`

The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

`min_version`

> **Type** string
>
> **Default** `<None>`

The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

`password`

> **Type** unknown type
>
> **Default** `<None>`

Users password

`project_domain_id`

> **Type** unknown type
>
> **Default** `<None>`

Domain ID containing project

`project_domain_name`

> **Type** unknown type
>
> **Default** `<None>`

Domain name containing project

project_id

Type unknown type

Default `<None>`

Project ID to scope to

Table 55: Deprecated Variations

Group	Name
service_catalog	tenant-id
service_catalog	tenant_id

project_name

Type unknown type

Default `<None>`

Project name to scope to

Table 56: Deprecated Variations

Group	Name
service_catalog	tenant-name
service_catalog	tenant_name

region_name

Type string

Default `<None>`

The default region_name for endpoint URL discovery.

service_name

Type string

Default `<None>`

The default service_name for endpoint URL discovery.

service_type

Type string

Default `baremetal-introspection`

The default service_type for endpoint URL discovery.

split_loggers

Type boolean

Default `False`

Log requests to multiple loggers.

status_code_retries

Type integer

Default `<None>`

The maximum number of retries that should be attempted for retriable HTTP status codes.

`status_code_retry_delay`

Type floating point

Default `<None>`

Delay (in seconds) between two retries for retriable status codes. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

`system_scope`

Type unknown type

Default `<None>`

Scope for system operations

`tenant_id`

Type unknown type

Default `<None>`

Tenant ID

`tenant_name`

Type unknown type

Default `<None>`

Tenant Name

`timeout`

Type integer

Default `<None>`

Timeout value for http requests

`trust_id`

Type unknown type

Default `<None>`

Trust ID

`user_domain_id`

Type unknown type

Default `<None>`

Users domain id

`user_domain_name`

Type unknown type

Default `<None>`

Users domain name

user_id

Type unknown type

Default `<None>`

User id

username

Type unknown type

Default `<None>`

Username

Table 57: Deprecated Variations

Group	Name
service_catalog	user-name
service_catalog	user_name

valid_interfaces

Type list

Default `['internal', 'public']`

List of interfaces, in order of preference, for endpoint URL.

version

Type string

Default `<None>`

Minimum Major API version within a given Major API version for endpoint URL discovery. Mutually exclusive with min_version and max_version

ssl

ca_file

Type string

Default `<None>`

CA certificate file to use to verify connecting clients.

Table 58: Deprecated Variations

Group	Name
DEFAULT	ssl_ca_file

cert_file

Type string

Default `<None>`

Certificate file to use when starting the server securely.

Table 59: Deprecated Variations

Group	Name
DEFAULT	ssl_cert_file

key_file

Type string

Default `<None>`

Private key file to use when starting the server securely.

Table 60: Deprecated Variations

Group	Name
DEFAULT	ssl_key_file

version

Type string

Default `<None>`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

ciphers

Type string

Default `<None>`

Sets the list of available ciphers. value should be a string in the OpenSSL cipher list format.

swift

auth_url

Type unknown type

Default `<None>`

Authentication URL

auth_type

Type unknown type

Default `<None>`

Authentication type to load

Table 61: Deprecated Variations

Group	Name
swift	auth_plugin

cafile

Type string

Default `<None>`

PEM encoded Certificate Authority to use when verifying HTTPs connections.

certfile

Type string

Default `<None>`

PEM encoded client certificate cert file

collect_timing

Type boolean

Default `False`

Collect per-API call timing information.

connect_retries

Type integer

Default `<None>`

The maximum number of retries that should be attempted for connection errors.

connect_retry_delay

Type floating point

Default `<None>`

Delay (in seconds) between two retries for connection errors. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

container

Type string

Default `ironic-inspector`

Default Swift container to use when creating objects.

default_domain_id

Type unknown type

Default `<None>`

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

default_domain_name

Type unknown type

Default `<None>`

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

delete_after

Type integer

Default `0`

Number of seconds that the Swift object will last before being deleted. (set to 0 to never delete the object).

domain_id

Type unknown type

Default `<None>`

Domain ID to scope to

domain_name

Type unknown type

Default `<None>`

Domain name to scope to

endpoint_override

Type string

Default `<None>`

Always use this endpoint URL for requests for this client. NOTE: The unversioned endpoint should be specified here; to request a particular API version, use the *version*, *min-version*, and/or *max-version* options.

insecure

Type boolean

Default `False`

Verify HTTPS connections.

keyfile

Type string

Default `<None>`

PEM encoded client certificate key file

max_version

Type string

Default `<None>`

The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

min_version

Type string

Default `<None>`

The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

password

Type unknown type

Default `<None>`

Users password

project_domain_id

Type unknown type

Default `<None>`

Domain ID containing project

project_domain_name

Type unknown type

Default `<None>`

Domain name containing project

project_id

Type unknown type

Default `<None>`

Project ID to scope to

Table 62: Deprecated Variations

Group	Name
swift	tenant-id
swift	tenant_id

project_name

Type unknown type

Default `<None>`

Project name to scope to

Table 63: Deprecated Variations

Group	Name
swift	tenant-name
swift	tenant_name

region_name

Type string

Default `<None>`

The default region_name for endpoint URL discovery.

service_name

Type string

Default `<None>`

The default service_name for endpoint URL discovery.

service_type

Type string

Default `object-store`

The default service_type for endpoint URL discovery.

split_loggers

Type boolean

Default `False`

Log requests to multiple loggers.

status_code_retries

Type integer

Default `<None>`

The maximum number of retries that should be attempted for retriable HTTP status codes.

status_code_retry_delay

Type floating point

Default `<None>`

Delay (in seconds) between two retries for retriable status codes. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

system_scope

Type unknown type

Default `<None>`

Scope for system operations

tenant_id

Type unknown type

Default `<None>`

Tenant ID

tenant_name

Type unknown type

Default `<None>`

Tenant Name

timeout

Type integer

Default `<None>`

Timeout value for http requests

trust_id

Type unknown type

Default `<None>`

Trust ID

user_domain_id

Type unknown type

Default `<None>`

Users domain id

user_domain_name

Type unknown type

Default `<None>`

Users domain name

user_id

Type unknown type

Default `<None>`

User id

username

Type unknown type

Default `<None>`

Username

Table 64: Deprecated Variations

Group	Name
swift	user-name
swift	user_name

valid_interfaces

Type list

Default `['internal', 'public']`

List of interfaces, in order of preference, for endpoint URL.

version

Type string

Default `<None>`

Minimum Major API version within a given Major API version for endpoint URL discovery. Mutually exclusive with min_version and max_version

3.2.2 Policies

The following is an overview of all available policies in **ironic inspector**. For a sample configuration file, refer to *Ironic Inspector Policy*.

ironic_inspector.api

`is_admin`

> **Default** `role:admin or role:administrator or role:baremetal_admin`

Full read/write API access

`is_observer`

> **Default** `role:baremetal_observer`

Read-only API access

`public_api`

> **Default** `is_public_api:True`

Internal flag for public API routes

`default`

> **Default** `!`

Default API access policy

`introspection`

> **Default** `rule:public_api`
>
> **Operations**
>
> - **GET** `/`

Access the API root for available versions information

`introspection:version`

> **Default** `rule:public_api`
>
> **Operations**
>
> - **GET** `/{version}`

Access the versioned API root for version information

`introspection:continue`

> **Default** `rule:public_api`
>
> **Operations**
>
> - **POST** `/continue`

Ramdisk callback to continue introspection

`introspection:status`

> **Default** `rule:is_admin or rule:is_observer`

Operations

- **GET** `/introspection`
- **GET** `/introspection/{node_id}`

Get introspection status

`introspection:start`

Default `rule:is_admin`

Operations

- **POST** `/introspection/{node_id}`

Start introspection

`introspection:abort`

Default `rule:is_admin`

Operations

- **POST** `/introspection/{node_id}/abort`

Abort introspection

`introspection:data`

Default `rule:is_admin`

Operations

- **GET** `/introspection/{node_id}/data`

Get introspection data

`introspection:reapply`

Default `rule:is_admin`

Operations

- **POST** `/introspection/{node_id}/data/unprocessed`

Reapply introspection on stored data

`introspection:rule:get`

Default `rule:is_admin`

Operations

- **GET** `/rules`
- **GET** `/rules/{rule_id}`

Get introspection rule(s)

`introspection:rule:delete`

Default `rule:is_admin`

Operations

- **DELETE** `/rules`

- **DELETE** `/rules/{rule_id}`

Delete introspection rule(s)

`introspection:rule:create`

Default `rule:is_admin`

Operations

- **POST** `/rules`

Create introspection rule

3.3 User Guide

3.3.1 How Ironic Inspector Works

How Ironic Inspector Works

Workflow

Usual hardware introspection flow is as follows:

- Operator enrolls nodes into Ironic e.g. via openstack baremetal CLI command. Power management credentials should be provided to Ironic at this step.
- Nodes are put in the correct state for introspection as described in *node states*.
- Operator sends nodes on introspection using **ironic-inspector** API or CLI (see *usage*).
- On receiving node UUID **ironic-inspector**:
  - validates node power credentials, current power and provisioning states,
  - allows access to PXE boot service for the nodes,
  - issues reboot command for the nodes, so that they boot the ramdisk.
- The ramdisk collects the required information and posts it back to **ironic-inspector**.
- On receiving data from the ramdisk, **ironic-inspector**:
  - validates received data,
  - finds the node in Ironic database using its BMC address (MAC address in case of SSH driver),
  - fills missing node properties with received data and creates missing ports.

Note: **ironic-inspector** is responsible to create Ironic ports for some or all NICs found on the node. **ironic-inspector** is also capable of deleting ports that should not be present. There are two important configuration options that affect this behavior: `add_ports` and `keep_ports` (please refer to *the sample configuration file* for a detailed explanation).

Default values as of **ironic-inspector** 1.1.0 are `add_ports=pxe`, `keep_ports=all`, which means that only one port will be added, which is associated with NIC the ramdisk PXE booted from. No ports will be deleted. This setting ensures that deploying on introspected nodes will succeed despite Ironic bug 1405131.

Ironic inspection feature by default requires different settings: `add_ports=all`, `keep_ports=present`, which means that ports will be created for all detected NICs, and all other ports will be deleted. Refer to the Ironic inspection documentation for details.

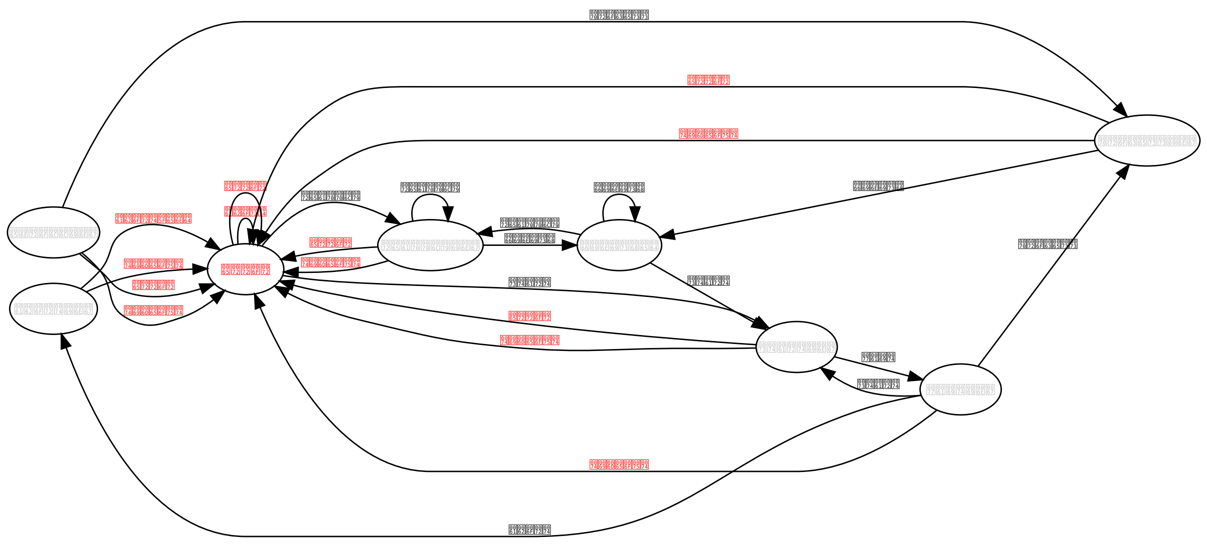
Ironic inspector can also be configured to not create any ports. This is done by setting `add_ports=disabled`. If setting `add_ports` to disabled the `keep_ports` option should be also set to `all`. This will ensure no manually added ports will be deleted.

- Separate API (see *usage* and API reference) can be used to query introspection results for a given node.
- Nodes are put in the correct state for deploying as described in *node states*.

Starting DHCP server and configuring PXE boot environment is not part of this package and should be done separately.

State machine diagram

The diagram below shows the introspection states that an **ironic-inspector** FSM goes through during the node introspection, discovery and reprocessing. The diagram also shows events that trigger state transitions.



3.3.2 How to use Ironic Inspector

Usage

Refer to the API reference for information on the HTTP API. Refer to the client documentation for information on how to use CLI and Python library.

Using from Ironic API

Ironic Kilo introduced support for hardware introspection under name of inspection. **ironic-inspector** introspection is supported for some generic drivers, please refer to Ironic inspection documentation for details.

Node States

- The nodes should be moved to `MANAGEABLE` provision state before introspection (requires *python-ironicclient* of version 0.5.0 or newer):

  ```
  openstack baremetal node manage <node>
  ```

- The introspection can be triggered by using the following command:

  ```
  openstack baremetal node inspect <node>
  ```

- After successful introspection and before deploying nodes should be made available to Nova, by moving them to `AVAILABLE` state:

  ```
  openstack baremetal node provide <node>
  ```

  Note: Due to how Nova interacts with Ironic driver, you should wait 1 minute before Nova becomes aware of available nodes after issuing this command. Use `nova hypervisor-stats` command output to check it.

Introspection Rules

Inspector supports a simple JSON-based DSL to define rules to run during introspection. Inspector provides an API to manage such rules, and will run them automatically after running all processing hooks.

A rule consists of conditions to check, and actions to run. If conditions evaluate to true on the introspection data, then actions are run on a node.

Please refer to the command below to import introspection rule:

```
openstack baremetal introspection rule import <json file>
```

Available conditions and actions are defined by plugins, and can be extended, see *How To Contribute* for details. See the API reference for specific calls to define introspection rules.

Conditions

A condition is represented by an object with fields:

`op` the type of comparison operation, default available operators include:

- `eq`, `le`, `ge`, `ne`, `lt`, `gt` - basic comparison operators;
- `in-net` - checks that an IP address is in a given network;
- `matches` - requires a full match against a given regular expression;
- `contains` - requires a value to contain a given regular expression;
- `is-empty` - checks that field is an empty string, list, dict or None value.

`field` a JSON path to the field in the introspection data to use in comparison.

Starting with the Mitaka release, you can also apply conditions to ironic node field. Prefix field with schema (`data://` or `node://`) to distinguish between values from introspection data and node. Both schemes use JSON path:

```
{"field": "node://property.path", "op": "eq", "value": "val"}
{"field": "data://introspection.path", "op": "eq", "value": "val"}
```

if scheme (node or data) is missing, condition compares data with introspection data.

`invert` boolean value, whether to invert the result of the comparison.

`multiple` how to treat situations where the `field` query returns multiple results (e.g. the field contains a list), available options are:

- `any` (the default) require any to match,
- `all` require all to match,
- `first` requrie the first to match.

All other fields are passed to the condition plugin, e.g. numeric comparison operations require a `value` field to compare against.

Scope

By default, introspection rules are applied to all nodes being inspected. In order for the rule to be applied only to specific nodes, a matching scope variable must be set to both the rule and the node. To set the scope for a rule include field `"scope"` in JSON file before importing. For example:

```
cat <json file>
  {
    "description": "...",
    "actions": [...],
    "conditions": [...],
    "scope": "SCOPE"
  }
```

Set the property `inspection_scope` on the node you want the rule to be applied to:

```
openstack baremetal node set --property inspection_scope="SCOPE" <node>
```

Now, when inspecting, the rule will be applied only to nodes with matching scope value. It will also ignore nodes that do not have `inspection_scope` property set. Note that if a rule has no scope set, it will be applied to all nodes, regardless if they have `inspection_scope` set or not.

Actions

An action is represented by an object with fields:

`action` type of action. Possible values are defined by plugins.

All other fields are passed to the action plugin.

Default available actions include:

- `fail` fail introspection. Requires a `message` parameter for the failure message.
- `set-attribute` sets an attribute on an Ironic node. Requires a `path` field, which is the path to the attribute as used by ironic (e.g. `/properties/something`), and a `value` to set.
- `set-capability` sets a capability on an Ironic node. Requires `name` and `value` fields, which are the name and the value for a new capability accordingly. Existing value for this same capability is replaced.
- `extend-attribute` the same as `set-attribute`, but treats existing value as a list and appends value to it. If optional `unique` parameter is set to `True`, nothing will be added if given value is already in a list.
- `add-trait` adds a trait to an Ironic node. Requires a `name` field with the name of the trait to add.
- `remove-trait` removes a trait from an Ironic node. Requires a `name` field with the name of the trait to remove.

Starting from Mitaka release, `value` field in actions supports fetching data from introspection, using python string formatting notation:

```
{"action": "set-attribute", "path": "/driver_info/ipmi_address",
 "value": "{data[inventory][bmc_address]}"}
```

Note that any value referenced in this way will be converted to a string.

If `value` is a dict or list, strings nested at any level within the structure will be formatted as well:

```
{"action": "set-attribute", "path": "/properties/root_device",
 "value": {"serial": "{data[root_device][serial]}"}}
```

Plugins

ironic-inspector heavily relies on plugins for data processing. Even the standard functionality is largely based on plugins. Set `processing_hooks` option in the configuration file to change the set of plugins to be run on introspection data. Note that order does matter in this option, especially for hooks that have dependencies on other hooks.

These are plugins that are enabled by default and should not be disabled, unless you understand what youre doing:

scheduler validates and updates basic hardware scheduling properties: CPU number and architecture, memory and disk size.

> **Note:** Diskless nodes have the disk size property `local_gb == 0`. Always use node driver `root_device` hints to prevent unexpected HW failures passing silently.

validate_interfaces validates network interfaces information. Creates new ports, optionally deletes ports that were not present in the introspection data. Also sets the `pxe_enabled` flag for the PXE-booting port and unsets it for all the other ports to avoid **nova** picking a random port to boot the node.

The following plugins are enabled by default, but can be disabled if not needed:

ramdisk_error reports error, if `error` field is set by the ramdisk, also optionally stores logs from `logs` field, see the API reference for details.

capabilities detect node capabilities: CPU, boot mode, etc. See *Capabilities Detection* for more details.

pci_devices gathers the list of all PCI devices returned by the ramdisk and compares to those defined in `alias` field(s) from `pci_devices` section of configuration file. The recognized PCI devices and their count are then stored in node properties. This information can be later used in nova flavors for node scheduling.

Here are some plugins that can be additionally enabled:

example example plugin logging its input and output.

raid_device gathers block devices from ramdisk and exposes root device in multiple runs.

extra_hardware stores the value of the data key returned by the ramdisk as a JSON encoded string in a Swift object. The plugin will also attempt to convert the data into a format usable by introspection rules. If this is successful then the new format will be stored in the extra key. The data key is then deleted from the introspection data, as unless converted its assumed unusable by introspection rules.

local_link_connection Processes LLDP data returned from inspection specifically looking for the port ID and chassis ID, if found it configures the local link connection information on the nodes Ironic ports with that data. To enable LLDP in the inventory from IPA `ipa-collect-lldp=1` should be passed as a kernel parameter to the IPA ramdisk. In order to avoid processing the raw LLDP data twice, the `lldp_basic` plugin should also be installed and run prior to this plugin.

lldp_basic Processes LLDP data returned from inspection and parses TLVs from the Basic Management (802.1AB), 802.1Q, and 802.3 sets and stores the processed data back to the Ironic inspector data in Swift.

physnet_cidr_map Configures the `physical_network` property of the nodes Ironic port when the IP address is in a configured CIDR mapping. CIDR to physical network mappings is set in configuration using the `[port_physnet]/cidr_map` option, for example:

```
[port_physnet]
cidr_map = 10.10.10.0/24:physnet_a, 2001:db8::/64:physnet_b
```

accelerators Processes PCI data returned from inspection and compares with the accelerator inventory, it will update accelerator device information to the properties field of the ironic node if any accelerator device is found, for example:

```
{'local_gb': '1115', 'cpus': '40', 'cpu_arch': 'x86_64', 'memory_mb':
↪'32768',
 'capabilities': 'boot_mode:bios,cpu_vt:true,cpu_aes:true,cpu_
↪hugepages:true,cpu_hugepages_1g:true,cpu_txt:true',
 'accel': [{'vendor_id': '10de', 'device_id': '1eb8', 'type': 'GPU',
            'pci_address': '0000:82:00.0',
            'device_info': 'NVIDIA Corporation Tesla T4'}]
}
```

Refer to *How To Contribute* for information on how to write your own plugin.

Discovery

Starting from Mitaka, **ironic-inspector** is able to register new nodes in Ironic.

The existing `node-not-found-hook` handles what happens if **ironic-inspector** receives inspection data from a node it can not identify. This can happen if a node is manually booted without registering it with Ironic first.

For discovery, the configuration file option `node_not_found_hook` should be set to load the hook called `enroll`. This hook will enroll the unidentified node into Ironic using the `fake-hardware` hardware type. This is a configurable option: set `enroll_node_driver` in the **ironic-inspector** configuration file to the hardware type you want. You can also configure arbitrary fields to set on discovery, for example:

```
[discovery]
enroll_node_driver = ipmi
enroll_node_fields = management_interface:noop,resource_class:baremetal
```

The `enroll` hook will also set the `ipmi_address` property on the new node, if its available in the introspection data we received, see ramdisk callback.

Once the `enroll` hook is finished, **ironic-inspector** will process the introspection data in the same way it would for an identified node. It runs the processing *plugins*, and after that it runs introspection rules, which would allow for more customisable node configuration, see *rules*.

A rule to set a nodes Ironic driver to `ipmi` and populate the required `driver_info` for that driver would look like:

```
[{
    "description": "Set IPMI driver_info if no credentials",
    "actions": [
        {"action": "set-attribute", "path": "driver", "value": "ipmi"},
        {"action": "set-attribute", "path": "driver_info/ipmi_username",
         "value": "username"},
        {"action": "set-attribute", "path": "driver_info/ipmi_password",
         "value": "password"}
    ],
    "conditions": [
        {"op": "is-empty", "field": "node://driver_info.ipmi_password"},
        {"op": "is-empty", "field": "node://driver_info.ipmi_username"}
    ]
},{
    "description": "Set deploy info if not already set on node",
    "actions": [
```

(continues on next page)

(continued from previous page)

```
        {"action": "set-attribute", "path": "driver_info/deploy_kernel",
         "value": "<glance uuid>"},
        {"action": "set-attribute", "path": "driver_info/deploy_ramdisk",
         "value": "<glance uuid>"}
    ],
    "conditions": [
        {"op": "is-empty", "field": "node://driver_info.deploy_ramdisk"},
        {"op": "is-empty", "field": "node://driver_info.deploy_kernel"}
    ]
}]
```

All nodes discovered and enrolled via the `enroll` hook, will contain an `auto_discovered` flag in the introspection data, this flag makes it possible to distinguish between manually enrolled nodes and auto-discovered nodes in the introspection rules using the rule condition `eq`:

```
{
    "description": "Enroll auto-discovered nodes with ipmi hardware type",
    "actions": [
        {"action": "set-attribute", "path": "driver", "value": "ipmi"}
    ],
    "conditions": [
        {"op": "eq", "field": "data://auto_discovered", "value": true}
    ]
}
```

Reapplying introspection on stored data

To allow correcting mistakes in introspection rules the API provides an entry point that triggers the introspection over stored data. The data to use for processing is kept in Swift separately from the data already processed. Reapplying introspection overwrites processed data in the store. Updating the introspection data through the endpoint isnt supported yet. Following preconditions are checked before reapplying introspection:

- no data is being sent along with the request
- Swift store is configured and enabled
- introspection data is stored in Swift for the node UUID
- node record is kept in database for the UUID
- introspection is not ongoing for the node UUID

Should the preconditions fail an immediate response is given to the user:

- `400` if the request contained data or in case Swift store is not enabled in configuration
- `404` in case Ironic doesnt keep track of the node UUID
- `409` if an introspection is already ongoing for the node

If the preconditions are met a background task is executed to carry out the processing and a `202 Accepted` response is returned to the endpoint user. As requested, these steps are performed in the background task:

- preprocessing hooks

- post processing hooks, storing result in Swift
- introspection rules

These steps are avoided, based on the feature requirements:

- `node_not_found_hook` is skipped
- power operations
- roll-back actions done by hooks

Limitations:

- theres no way to update the unprocessed data atm.
- the unprocessed data is never cleaned from the store
- check for stored data presence is performed in background; missing data situation still results in a `202` response

Capabilities Detection

Starting with the Newton release, **Ironic Inspector** can optionally discover several node capabilities. A recent (Newton or newer) IPA image is required for it to work.

Boot mode

The current boot mode (BIOS or UEFI) can be detected and recorded as `boot_mode` capability in Ironic. It will make some drivers to change their behaviour to account for this capability. Set the `[capabilities]boot_mode` configuration option to `True` to enable.

CPU capabilities

Several CPU flags are detected by default and recorded as following capabilities:

- `cpu_aes` AES instructions.
- `cpu_vt` virtualization support.
- `cpu_txt` TXT support.
- `cpu_hugepages` huge pages (2 MiB) support.
- `cpu_hugepages_1g` huge pages (1 GiB) support.

It is possible to define your own rules for detecting CPU capabilities. Set the `[capabilities]cpu_flags` configuration option to a mapping between a CPU flag and a capability, for example:

```
cpu_flags = aes:cpu_aes,svm:cpu_vt,vmx:cpu_vt
```

See the default value of this option for a more detail example.

InfiniBand support

Starting with the Ocata release, **Ironic Inspector** supports detection of InfiniBand network interfaces. A recent (Ocata or newer) IPA image is required for that to work. When an InfiniBand network interface is discovered, the **Ironic Inspector** adds a `client-id` attribute to the `extra` attribute in the ironic port. The **Ironic Inspector** should be configured with `iptables.ethoib_interfaces` to indicate the Ethernet Over InfiniBand (EoIB) which are used for physical access to the DHCP network. For example if **Ironic Inspector** DHCP server is using `br-inspector` and the `br-inspector` has EoIB port e.g. `eth0`, the `iptables.ethoib_interfaces` should be set to `eth0`. The `iptables.ethoib_interfaces` allows to map the baremetal GUID to its EoIB MAC based on the neighs files. This is needed for blocking DHCP traffic of the nodes (MACs) which are not part of the introspection.

The format of the `/sys/class/net/<ethoib>/eth/neighs` file:

```
# EMAC=<ethernet mac of the ethoib> IMAC=<qp number:lid:GUID>
# For example:
IMAC=97:fe:80:00:00:00:00:00:00:7c:fe:90:03:00:29:26:52
qp number=97:fe
lid=80:00:00:00:00:00:00
GUID=7c:fe:90:03:00:29:26:52
```

Example of content:

```
EMAC=02:00:02:97:00:01␣
↪IMAC=97:fe:80:00:00:00:00:00:00:7c:fe:90:03:00:29:26:52
EMAC=02:00:00:61:00:02␣
↪IMAC=61:fe:80:00:00:00:00:00:00:7c:fe:90:03:00:29:24:4f
```

3.3.3 HTTP API Reference

- Bare Metal Introspection API Reference.

3.3.4 Troubleshooting

Troubleshooting

Errors when starting introspection

- *Invalid provision state available*

  In Kilo release with *python-ironicclient* 0.5.0 or newer Ironic defaults to reporting provision state `AVAILABLE` for newly enrolled nodes. **ironic-inspector** will refuse to conduct introspection in this state, as such nodes are supposed to be used by Nova for scheduling. See *node states* for instructions on how to put nodes into the correct state.

Introspection times out

There may be 3 reasons why introspection can time out after some time (defaulting to 60 minutes, altered by `timeout` configuration option):

1. Fatal failure in processing chain before node was found in the local cache. See *Troubleshooting data processing* for the hints.
2. Failure to load the ramdisk on the target node. See *Troubleshooting PXE boot* for the hints.
3. Failure during ramdisk run. See *Troubleshooting ramdisk run* for the hints.

Troubleshooting data processing

In this case **ironic-inspector** logs should give a good idea what went wrong. E.g. for RDO or Fedora the following command will output the full log:

```
sudo journalctl -u openstack-ironic-inspector
```

(use `openstack-ironic-discoverd` for version < 2.0.0).

Note: Service name and specific command might be different for other Linux distributions (and for old version of **ironic-inspector**).

If `ramdisk_error` plugin is enabled and `ramdisk_logs_dir` configuration option is set, **ironic-inspector** will store logs received from the ramdisk to the `ramdisk_logs_dir` directory. This depends, however, on the ramdisk implementation.

A local cache miss during data processing would leave a message like:

```
ERROR ironic_python_agent.inspector [-] inspectorerror 400: {"error":{
↪"message":"The following failures happened during running pre-processing␣
↪hooks:\nLook up error: Could not find a node for attributes {'bmc_address
↪': u'10.x.y.z', 'mac': [u'00:aa:bb:cc:dd:ee', u'00:aa:bb:cc:dd:ef']}"}}
```

One potential explanation for such an error is a misconfiguration in the BMC where a channel with the wrong IP address is active (and hence detected and reported back by the Ironic Python Agent), but can then not be matched to the IP address Ironic has in its cache for this node.

Troubleshooting PXE boot

PXE booting most often becomes a problem for bare metal environments with several physical networks. If the hardware vendor provides a remote console (e.g. iDRAC for DELL), use it to connect to the machine and see what is going on. You may need to restart introspection.

Another source of information is DHCP and TFTP server logs. Their location depends on how the servers were installed and run. For RDO or Fedora use:

```
$ sudo journalctl -u openstack-ironic-inspector-dnsmasq
```

(use `openstack-ironic-discoverd-dnsmasq` for version < 2.0.0).

The last resort is `tcpdump` utility. Use something like

```
$ sudo tcpdump -i any port 67 or port 68 or port 69
```

to watch both DHCP and TFTP traffic going through your machine. Replace `any` with a specific network interface to check that DHCP and TFTP requests really reach it.

If you see node not attempting PXE boot or attempting PXE boot on the wrong network, reboot the machine into BIOS settings and make sure that only one relevant NIC is allowed to PXE boot.

If you see node attempting PXE boot using the correct NIC but failing, make sure that:

1. network switches configuration does not prevent PXE boot requests from propagating,
2. there is no additional firewall rules preventing access to port 67 on the machine where *ironic-inspector* and its DHCP server are installed.

If you see node receiving DHCP address and then failing to get kernel and/or ramdisk or to boot them, make sure that:

1. TFTP server is running and accessible (use `tftp` utility to verify),
2. no firewall rules prevent access to TFTP port,
3. SELinux is configured properly to allow external TFTP access,

   If SELinux is neither permissive nor disabled, you should config `tftp_home_dir` in SELinux by executing the command

   ```
   $ sudo setsebool -P tftp_home_dir 1
   ```

   See the man page for more details.
4. DHCP server is correctly set to point to the TFTP server,
5. `pxelinux.cfg/default` within TFTP root contains correct reference to the kernel and ramdisk.

Note: If using iPXE instead of PXE, check the HTTP server logs and the iPXE configuration instead.

Troubleshooting ramdisk run

First, check if the ramdisk logs were stored locally as described in the *Troubleshooting data processing* section. If not, ensure that the ramdisk actually booted as described in the *Troubleshooting PXE boot* section.

Finally, you can try connecting to the IPA ramdisk. If you have any remote console access to the machine, you can check the logs as they appear on the screen. Otherwise, you can rebuild the IPA image with your SSH key to be able to log into it. Use the dynamic-login or devuser element for a DIB-based build or put an authorized_keys file in `/usr/share/oem/` for a CoreOS-based one.

Troubleshooting DNS issues on Ubuntu

Ubuntu uses local DNS caching, so tries localhost for DNS results first before calling out to an external DNS server. When DNSmasq is installed and configured for use with ironic-inspector, it can cause problems by interfering with the local DNS cache. To fix this issue ensure that `/etc/resolve.conf` points to your external DNS servers and not to `127.0.0.1`.

On Ubuntu 14.04 this can be done by editing your `/etc/resolvconf/resolv.conf.d/head` and adding your nameservers there. This will ensure they will come up first when `/etc/resolv.conf` is regenerated.

Running Inspector in a VirtualBox environment

By default VirtualBox does not expose a DMI table to the guest. This prevents ironic-inspector from being able to discover the properties of the a node. In order to run ironic-inspector on a VirtualBox guest the host must be configured to expose DMI data inside the guest. To do this run the following command on the VirtualBox host:

```
VBoxManage setextradata {NodeName} "VBoxInternal/Devices/pcbios/0/Config/
↪DmiExposeMemoryTable" 1
```

Note: Replace *{NodeName}* with the name of the guest you wish to expose the DMI table on. This command will need to be run once per host to enable this functionality.

HTTP API

See https://docs.openstack.org/api-ref/baremetal-introspection/

3.4 Administrator Guide

3.4.1 How to upgrade Ironic Inspector

Upgrade Guide

The release notes should always be read carefully when upgrading the ironic-inspector service. Starting with the Mitaka series, specific upgrade steps and considerations are well-documented in the release notes.

Upgrades are only supported one series at a time, or within a series. Only offline (with downtime) upgrades are currently supported.

When upgrading ironic-inspector, the following steps should always be taken:

- Update ironic-inspector code, without restarting the service yet.
- Stop the ironic-inspector service.
- Run database migrations:

```
ironic-inspector-dbsync --config-file <PATH-TO-INSPECTOR.CONF> upgrade
```

- Start the ironic-inspector service.
- Upgrade the ironic-python-agent image used for introspection.

Note: There is no implicit upgrade order between ironic and ironic-inspector, unless the release notes say otherwise.

Migrating introspection data

Starting with Stein release, ironic-inspector supports two introspection data storage backends: `swift` and `database`. If you decide to change the backend, you can use the provided command to migrate the data:

```
ironic-inspector-migrate-data --from swift --to database --config-file /
↪etc/ironic-inspector/inspector.conf
```

Note: Configuration for **both** backends is expected to be present in the configuration file for this command to succeed.

3.4.2 Dnsmasq PXE filter driver

dnsmasq PXE filter

An inspection PXE DHCP stack is often implemented by the **dnsmasq** service. The **dnsmasq** PXE filter implementation relies on directly configuring the **dnsmasq** DHCP service to provide a caching PXE traffic filter of node MAC addresses.

How it works

The filter works by populating the **dnsmasq** DHCP hosts directory with a configuration file per MAC address. Each file is either enabling or disabling, thru the `ignore` directive, the DHCP service for a particular MAC address:

```
$ cat /etc/dnsmasq.d/de-ad-be-ef-de-ad
de:ad:be:ef:de:ad,ignore
$
```

The filename is used to keep track of all MAC addresses in the cache, avoiding file parsing. The content of the file determines the MAC address access policy.

Thanks to the `inotify` facility, **dnsmasq** is notified once a new file is *created* or an existing file is *modified* in the DHCP hosts directory. Thus, to allow a MAC address, the filter removes the `ignore` directive:

```
$ cat /etc/dnsmasq.d/de-ad-be-ef-de-ad
de:ad:be:ef:de:ad
$
```

The hosts directory content establishes a *cached* MAC addresses filter that is kept synchronized with the **ironic** port list.

Note: The **dnsmasq** inotify facility implementation doesnt react to a file being removed or truncated.

Configuration

The `inotify` facility was introduced to **dnsmasq** in the version *2.73*. This filter driver has been checked by **ironic-inspector** CI with **dnsmasq** versions *>=2.76*.

To enable the **dnsmasq** PXE filter, update the PXE filter driver name in the **ironic-inspector** configuration file:

```
[pxe_filter]
driver = dnsmasq
```

The DHCP hosts directory can be specified to override the default `/var/lib/ironic-inspector/dhcp-hostsdir`:

```
[dnsmasq_pxe_filter]
dhcp_hostsdir = /etc/ironic-inspector/dhcp-hostsdir
```

The filter design relies on the hosts directory being in exclusive **ironic-inspector** control. The hosts directory should be considered a *private cache* directory of **ionic-inspector** that **dnsmasq** polls configuration updates from, through the `inotify` facility. The directory has to be writable by **ironic-inspector** and readable by **dnsmasq**.

It is also possible to override the default (empty) **dnsmasq** start and stop commands to, for instance, directly control the **dnsmasq** service:

```
[dnsmasq_pxe_filter]
dnsmasq_start_command = dnsmasq --conf-file /etc/ironic-inspector/dnsmasq.
↪conf
dnsmasq_stop_command = kill $(cat /var/run/dnsmasq.pid)
```

Note: The commands support shell expansion. The default empty start command means the **dnsmasq** service wont be started upon the filter initialization. Conversely, the default empty stop command means the service wont be stopped upon an (error) exit.

Note: These commands are executed through the rootwrap facility, so overriding may require a filter file to be created in the `rootwrap.d` directory. A sample configuration to use with the **systemctl** facility might be:

```
sudo cat > /etc/ironic-inspector/rootwrap.d/ironic-inspector-dnsmasq-
↪systemctl.filters <<EOF
[Filters]
# ironic_inspector/pxe_filter/dnsmasq.py
systemctl: CommandFilter, systemctl, root, restart, dnsmasq
systemctl: CommandFilter, systemctl, root, stop, dnsmasq
EOF
```

Caveats

The initial synchronization will put some load on the **dnsmasq** service starting based on the amount of ports **ironic** keeps. The start-up can take up to a minute of full CPU load for huge amounts of MACs (tens of thousands). Subsequent filter synchronizations will only cause the **dnsmasq** to parse the modified files. Typically those are the bare metal nodes being added or phased out from the compute service, meaning dozens of file updates per sync call.

The **ironic-inspector** takes over the control of the DHCP hosts directory to implement its filter cache. Files are generated dynamically so should not be edited by hand. To minimize the interference between the deployment and introspection, **ironic-inspector** has to start the **dnsmasq** service only after the initial synchronization. Conversely, the **dnsmasq** service is stopped upon (unexpected) **ironic-inspector** exit.

To avoid accumulating stale DHCP host files over time, the driver cleans up the DHCP hosts directory before the initial synchronization during the start-up.

Although the filter driver tries its best to always stop the **dnsmasq** service, it is recommended that the operator configures the **dnsmasq** service in such a way that it terminates upon **ironic-inspector** (unexpected) exit to prevent a stale deny list from being used by the **dnsmasq** service.

CHAPTER

FOUR

CONTRIBUTOR DOCS

4.1 How To Contribute

4.1.1 Basics

- Our source code is hosted on OpenStack GitHub, but please do not send pull requests there.
- Please follow usual OpenStack Gerrit Workflow to submit a patch.
- Update change log in README.rst on any significant change.
- It goes without saying that any code change should by accompanied by unit tests.
- Note the branch youre proposing changes to. `master` is the current focus of development, use `stable/VERSION` for proposing an urgent fix, where `VERSION` is the current stable series. E.g. at the moment of writing the stable branch is `stable/1.0`.
- Please file an RFE in StoryBoard for any significant code change and a regular story for any significant bug fix.

4.1.2 Development Environment

First of all, install *tox* utility. Its likely to be in your distribution repositories under name of `python-tox`. Alternatively, you can install it from PyPI.

Next checkout and create environments:

```
git clone https://github.com/openstack/ironic-inspector.git
cd ironic-inspector
tox
```

Repeat *tox* command each time you need to run tests. If you dont have Python interpreter of one of supported versions (currently 3.6 and 3.7), use `-e` flag to select only some environments, e.g.

```
tox -e py36
```

Note: This command also runs tests for database migrations. By default the sqlite backend is used. For testing with mysql or postgresql, you need to set up a db named openstack_citest with user openstack_citest and password openstack_citest on localhost. Use the script `tools/test_setup.sh` to set the database up the same way as done in the OpenStack CI environment.

Note: Users of Fedora <= 23 will need to run sudo dnf releasever=24 update python-virtualenv to run unit tests

To run the functional tests, use:

```
tox -e func
```

Once you have added new state or transition into inspection state machine, you should regenerate *State machine diagram* with:

```
tox -e genstates
```

Run the service with:

```
.tox/py36/bin/ironic-inspector --config-file example.conf
```

Of course you may have to modify `example.conf` to match your OpenStack environment. See the install guide for information on generating or downloading an example configuration file.

You can develop and test **ironic-inspector** using DevStack - see *Deploying Ironic Inspector with DevStack* for the current status.

4.1.3 Deploying Ironic Inspector with DevStack

DevStack provides a way to quickly build a full OpenStack development environment with requested components. There is a plugin for installing **ironic-inspector** in DevStack. Installing **ironic-inspector** requires a machine running Ubuntu 14.04 (or later) or Fedora 23 (or later). Make sure this machine is fully up to date and has the latest packages installed before beginning this process.

Download DevStack:

```
git clone https://git.openstack.org/openstack-dev/devstack.git
cd devstack
```

Create `local.conf` file with minimal settings required to enable both the **ironic** and the **ironic-inspector**. You can start with the *Example local.conf* and extend it as needed.

Example local.conf

```
[[local|localrc]]
# 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://opendev.org/openstack/ironic
```

(continues on next page)

(continued from previous page)

```
enable_plugin ironic-inspector https://opendev.org/openstack/ironic-
↪inspector

# 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

# Disable Horizon
disable_service horizon

# Disable Cinder
disable_service cinder c-sch c-api c-vol

# 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

# Enable inspection via ironic-inspector
IRONIC_ENABLED_INSPECT_INTERFACES=inspector,no-inspect
# Make it the default for all hardware types:
IRONIC_DEFAULT_INSPECT_INTERFACE=inspector

# 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=2048
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
IRONIC_INSPECTOR_BUILD_RAMDISK=False
```

(continues on next page)

(continued from previous page)

```
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=/opt/stack/devstack.log
LOGDIR=/opt/stack/logs
IRONIC_VM_LOG_DIR=/opt/stack/ironic-bm-logs
```

Notes

- Set IRONIC_INSPECTOR_BUILD_RAMDISK to True if you want to build ramdisk. Default value is False and ramdisk will be downloaded instead of building.
- 1024 MiB of RAM is a minimum required for the default build of IPA based on CoreOS. If you plan to use another operating system and build IPA with diskimage-builder 2048 MiB is recommended.
- Network configuration is pretty sensitive, better not to touch it without deep understanding.
- This configuration disables **horizon**, **heat**, **cinder** and **tempest**, adjust it if you need these services.

Start the install:

```
./stack.sh
```

Usage

After installation is complete, you can source `openrc` in your shell, and then use the OpenStack CLI to manage your DevStack:

```
source openrc admin demo
```

Show DevStack screens:

```
screen -x stack
```

To exit screen, hit `CTRL-a d`.

List baremetal nodes:

```
openstack baremetal node list
```

Bring the node to manageable state:

```
openstack baremetal node manage <NodeID>
```

Inspect the node:

```
openstack baremetal node inspect <NodeID>
```

Note: The deploy driver used must support the inspect interface. See also the Ironic Python Agent.

A node can also be inspected using the following command. However, this will not affect the provision state of the node:

```
openstack baremetal introspection start <NodeID>
```

Check inspection status:

```
openstack baremetal introspection status <NodeID>
```

Optionally, get the inspection data:

```
openstack baremetal introspection data save <NodeID>
```

4.1.4 Writing a Plugin

- **ironic-inspector** allows you to hook code into the data processing chain after introspection. Inherit `ProcessingHook` class defined in ironic_inspector.plugins.base module and overwrite any or both of the following methods:

  `before_processing(introspection_data,**)` called before any data processing, providing the raw data. Each plugin in the chain can modify the data, so order in which plugins are loaded matters here. Returns nothing.

  `before_update(introspection_data,node_info,**)` called after node is found and ports are created, but before data is updated on a node. Please refer to the docstring for details and examples.

  You can optionally define the following attribute:

  `dependencies` a list of entry point names of the hooks this hook depends on. These hooks are expected to be enabled before the current hook.

  Make your plugin a setuptools entry point under `ironic_inspector.hooks.processing` namespace and enable it in the configuration file (`processing.processing_hooks` option).

- **ironic-inspector** allows plugins to override the action when node is not found in node cache. Write a callable with the following signature:

  `(introspection_data,**)` called when node is not found in cache, providing the processed data. Should return a `NodeInfo` class instance.

  Make your plugin a setuptools entry point under `ironic_inspector.hooks.node_not_found` namespace and enable it in the configuration file (`processing.node_not_found_hook` option).

- **ironic-inspector** allows more condition types to be added for Introspection Rules. Inherit `RuleConditionPlugin` class defined in ironic_inspector.plugins.base module and overwrite at least the following method:

check(node_info,field,params,)** called to check that condition holds for a given field. Field value is provided as `field` argument, `params` is a dictionary defined at the time of condition creation. Returns boolean value.

The following methods and attributes may also be overridden:

validate(params,)** called to validate parameters provided during condition creating. Default implementation requires keys listed in `REQUIRED_PARAMS` (and only them).

REQUIRED_PARAMS contains set of required parameters used in the default implementation of `validate` method, defaults to `value` parameter.

ALLOW_NONE if its set to `True`, missing fields will be passed as `None` values instead of failing the condition. Defaults to `False`.

Make your plugin a setuptools entry point under `ironic_inspector.rules.conditions` namespace.

- **ironic-inspector** allows more action types to be added for Introspection Rules. Inherit `RuleActionPlugin` class defined in ironic_inspector.plugins.base module and overwrite at least the following method:

  apply(node_info,params,)** called to apply the action.

  The following methods and attributes may also be overridden:

  validate(params,)** called to validate parameters provided during actions creating. Default implementation requires keys listed in `REQUIRED_PARAMS` (and only them).

  REQUIRED_PARAMS contains set of required parameters used in the default implementation of `validate` method, defaults to no parameters.

  Make your plugin a setuptools entry point under `ironic_inspector.rules.conditions` namespace.

Note: `**` argument is needed so that we can add optional arguments without breaking out-of-tree plugins. Please make sure to include and ignore it.

4.1.5 Making changes to the database

In order to make a change to the ironic-inspector database you must update the database models found in ironic_inspector.db and then create a migration to reflect that change.

There are two ways to create a migration which are described below, both of these generate a new migration file. In this file there is only one function:

- **`upgrade` - The function to run when** `ironic-inspector-dbsync upgrade` is run, and should be populated with code to bring the database up to its new state from the state it was in after the last migration.

For further information on creating a migration, refer to Create a Migration Script from the alembic documentation.

Autogenerate

This is the simplest way to create a migration. Alembic will compare the models to an up to date database, and then attempt to write a migration based on the differences. This should generate correct migrations in most cases however there are some cases when it can not detect some changes and may require manual modification, see What does Autogenerate Detect (and what does it not detect?) from the alembic documentation.

```
ironic-inspector-dbsync upgrade
ironic-inspector-dbsync revision -m "A short description" --autogenerate
```

Manual

This will generate an empty migration file, with the correct revision information already included. However the upgrade function is left empty and must be manually populated in order to perform the correct actions on the database:

```
ironic-inspector-dbsync revision -m "A short description"
```

4.1.6 Implementing PXE Filter Drivers

Background

inspector in-band introspection PXE-boots the Ironic Python Agent live image, to inspect the baremetal server. **ironic** also PXE-boots IPA to perform tasks on a node, such as deploying an image. **ironic** uses **neutron** to provide DHCP, however **neutron** does not provide DHCP for unknown MAC addresses so **inspector** has to use its own DHCP/TFTP stack for discovery and inspection.

When **ironic** and **inspector** are operating in the same L2 network, there is a potential for the two DHCPs to race, which could result in a node being deployed by **ironic** being PXE booted by **inspector**.

To prevent DHCP races between the **inspector** DHCP and **ironic** DHCP, **inspector** has to be able to filter which nodes can get a DHCP lease from the **inspector** DHCP server. These filters can then be used to prevent nodes enrolled in **ironic** inventory from being PXE-booted unless they are explicitly moved into the `inspected` state.

Filter Interface

The contract between **inspector** and a PXE filter driver is described in the *`FilterDriver`* interface. The methods a driver has to implement are:

- *`init_filter()`* called on the service start to initialize internal driver state
- *`sync()`* called both periodically and when a node starts or finishes introspection to allow or deny its ports MAC addresses in the driver
- *`tear_down_filter()`* called on service exit to reset the internal driver state

The driver-specific configuration is suggested to be parsed during instantiation. Theres also a convenience generic interface implementation *`BaseFilter`* that provides base locking and initialization implementation. If required, a driver can opt-out from the periodic synchronization by overriding the *`get_periodic_sync_task()`*.

4.1.7 Python API

ironic_inspector

ironic_inspector package

Subpackages

ironic_inspector.cmd package

Submodules

ironic_inspector.cmd.all module

The Ironic Inspector service.

`ironic_inspector.cmd.all.`**`main`**(*args=['-b', 'latex', 'doc/source', 'doc/build/pdf']*)

ironic_inspector.cmd.conductor module

The Ironic Inspector Conductor service.

`ironic_inspector.cmd.conductor.`**`main`**(*args=['-b', 'latex', 'doc/source', 'doc/build/pdf']*)

ironic_inspector.cmd.dbsync module

`ironic_inspector.cmd.dbsync.`**`add_alembic_command`**(*subparsers*, *name*)

`ironic_inspector.cmd.dbsync.`**`add_command_parsers`**(*subparsers*)

`ironic_inspector.cmd.dbsync.`**`do_alembic_command`**(*config*, *cmd*, **args*, ***kwargs*)

`ironic_inspector.cmd.dbsync.`**`do_revision`**(*config*, *cmd*, **args*, ***kwargs*)

`ironic_inspector.cmd.dbsync.`**`main`**(*args=['-b', 'latex', 'doc/source', 'doc/build/pdf']*)

`ironic_inspector.cmd.dbsync.`**`with_revision`**(*config*, *cmd*, **args*, ***kwargs*)

ironic_inspector.cmd.migration module

Migrate introspected data between Swift and database.

`class` `ironic_inspector.cmd.migration.`**`MigrationTool`**

Bases: `object`

`main`()

`ironic_inspector.cmd.migration.`**`main`**()

ironic_inspector.cmd.wsgi module

WSGI script for Ironic Inspector API, installed by pbr.

`ironic_inspector.cmd.wsgi.`**`initialize_wsgi_app`**`()`

Module contents

ironic_inspector.common package

Submodules

ironic_inspector.common.context module

`class` `ironic_inspector.common.context.`**`RequestContext`**`(`*is_public_api=False*, ***kwargs*`)`

Bases: `oslo_context.context.RequestContext`

Extends security contexts from the oslo.context library.

`classmethod from_dict``(`*values*, ***kwargs*`)`

Construct a context object from a provided dictionary.

`classmethod from_environ``(`*environ*, ***kwargs*`)`

Load a context object from a request environment.

If keyword arguments are provided then they override the values in the request environment.

Parameters **`environ`** (*`dict`*) The environment dictionary associated with a request.

`to_policy_values``()`

A dictionary of context attributes to enforce policy with.

oslo.policy enforcement requires a dictionary of attributes representing the current logged in user on which it applies policy enforcement. This dictionary defines a standard list of attributes that should be available for enforcement across services.

It is expected that services will often have to override this method with either deprecated values or additional attributes used by that service specific policy.

ironic_inspector.common.coordination module

`class` `ironic_inspector.common.coordination.`**`Coordinator`**`(`*prefix=None*`)`

Bases: `object`

Tooz coordination wrapper.

`get_lock``(`*uuid*`)`

Get lock for node uuid.

`get_members``()`

Get members in the service group.

`group_name = b'ironic_inspector.service_group'`

join_group()
Join service group.

leave_group()
Leave service group

lock_prefix = 'ironic_inspector.'

run_elect_coordinator()
Trigger a new leader election.

start(*heartbeat=True*)
Start coordinator.

Parameters **heartbeat** Whether spawns a new thread to keep heartbeating with the tooz backend. Unless there is periodic task to do heartbeat manually, it should be always set to True.

stop()
Disconnect from coordination backend and stop heartbeat.

ironic_inspector.common.coordination.**get_coordinator**(*prefix=None*)

ironic_inspector.common.ironic module

exception ironic_inspector.common.ironic.**NotFound**(*node_ident*, *code=404*, **args*, ***kwargs*)
Bases: *ironic_inspector.utils.Error*

Node not found in Ironic.

ironic_inspector.common.ironic.**call_with_retries**(*func*, **args*, ***kwargs*)
Call an ironic client function retrying all errors.

If an ironic client exception is raised, try calling the func again, at most 5 times, waiting 1 sec between each call. If on the 5th attempt the func raises again, the exception is propagated to the caller.

ironic_inspector.common.ironic.**capabilities_to_dict**(*caps*)
Convert the Nodes capabilities into a dictionary.

ironic_inspector.common.ironic.**check_provision_state**(*node*)
Sanity checks the provision state of the node.

Parameters **node** An API client returned node object describing the baremetal node according to ironics node data model.

Returns None if no action is to be taken, True if the power node state should not be modified.

Raises Error on an invalid state being detected.

ironic_inspector.common.ironic.**dict_to_capabilities**(*caps_dict*)
Convert a dictionary into a string with the capabilities syntax.

ironic_inspector.common.ironic.**get_client**(*token=None*)
Get an ironic client connection.

`ironic_inspector.common.ironic.`**`get_ipmi_address`**(*node*)

Get the BMC address defined in node.driver_info dictionary

Possible names of BMC address value examined in order of list [ipmi_address] + CONF.ipmi_address_fields. The value could be an IP address or a hostname. DNS lookup performed for the first non empty value.

The first valid BMC address value returned along with its v4 and v6 IP addresses.

Parameters **`node`** Node object with defined driver_info dictionary

Returns tuple (ipmi_address, ipv4_address, ipv6_address)

`ironic_inspector.common.ironic.`**`get_node`**(*node_id*, *ironic=None*, ***kwargs*)

Get a node from Ironic.

Parameters

- **`node_id`** node UUID or name.
- **`ironic`** ironic client instance.
- **`kwargs`** arguments to pass to Ironic client.

Raises Error on failure

`ironic_inspector.common.ironic.`**`lookup_node`**(*macs=None*, *bmc_addresses=None*, *introspection_data=None*, *ironic=None*)

Lookup a node in the ironic database.

`ironic_inspector.common.ironic.`**`lookup_node_by_bmc_addresses`**(*addresses*, *introspection_data=None*, *ironic=None*, *fail=False*)

Find a node by its BMC address.

`ironic_inspector.common.ironic.`**`lookup_node_by_macs`**(*macs*, *introspection_data=None*, *ironic=None*, *fail=False*)

Find a node by its MACs.

`ironic_inspector.common.ironic.`**`reset_ironic_session`**()

Reset the global session variable.

Mostly useful for unit tests.

ironic_inspector.common.keystone module

`ironic_inspector.common.keystone.`**`add_auth_options`**(*options*, *service_type*)

`ironic_inspector.common.keystone.`**`get_adapter`**(*group*, ***adapter_kwargs*)

`ironic_inspector.common.keystone.`**`get_endpoint`**(*group*, ***kwargs*)

`ironic_inspector.common.keystone.`**`get_session`**(*group*)

`ironic_inspector.common.keystone.`**`register_auth_opts`**(*group*, *service_type*)

ironic_inspector.common.lldp_parsers module

Names and mapping functions used to map LLDP TLVs to name/value pairs

class `ironic_inspector.common.lldp_parsers.`**`LLDPBasicMgmtParser`**(*nv=None*)

Bases: `ironic_inspector.common.lldp_parsers.LLDPParser`

Class to handle parsing of 802.1AB Basic Management set

This class will also handle 802.1Q and 802.3 OUI TLVs.

`add_capabilities`(*struct*, *name*, *data*)

Handle LLDP_TLV_SYS_CAPABILITIES

`add_mgmt_address`(*struct*, *name*, *data*)

Handle LLDP_TLV_MGMT_ADDRESS

There can be multiple Mgmt Address TLVs, store in list.

`handle_org_specific_tlv`(*struct*, *name*, *data*)

Handle Organizationally Unique ID TLVs

This class supports 802.1Q and 802.3 OUI TLVs.

See http://www.ieee802.org/1/pages/802.1Q-2014.html, Annex D and http://standards.ieee.org/about/get/802/802.3.html

class `ironic_inspector.common.lldp_parsers.`**`LLDPParser`**(*node_info*, *nv=None*)

Bases: `object`

Base class to handle parsing of LLDP TLVs

Each class that inherits from this base class must provide a parser map. Parser maps are used to associate a LLDP TLV with a function handler and arguments necessary to parse the TLV and generate one or more name/value pairs. Each LLDP TLV maps to a tuple with the following fields:

function - handler function to generate name/value pairs

construct - name of construct definition for TLV

name - user-friendly name of TLV. For TLVs that generate only one name/value pair this is the name used

len_check - boolean indicating if length check should be done on construct

Its valid to have a function handler of None, this is for TLVs that are not mapped to a name/value pair(e.g.LLDP_TLV_TTL).

add_dot1_link_aggregation(*struct*, *name*, *data*)
Add name/value pairs for TLV Dot1_LinkAggregationId

This is in base class since it can be used by both dot1 and dot3.

add_nested_value(*struct*, *name*, *data*)
Add a single nested name/value pair to the dict

add_single_value(*struct*, *name*, *data*)
Add a single name/value pair to the nv dict

append_value(*name*, *value*)
Add value to a list mapped to name

parse_tlv(*tlv_type*, *data*)
Parse TLVs from mapping table

This functions takes the TLV type and the raw data for this TLV and gets a tuple from the parser_map. The construct field in the tuple contains the construct lib definition of the TLV which can be parsed to access individual fields. Once the TLV is parsed, the handler function for each TLV will store the individual fields as name/value pairs in nv_dict.

If the handler function does not exist, then no name/value pairs will be added to nv_dict, but since the TLV was handled, True will be returned.

Param tlv_type - type identifier for TLV

Param data - raw TLV value

Returns True if TLV in parser_map and data is valid, otherwise False.

set_value(*name*, *value*)
Set name value pair in dictionary

The value for a name should not be changed if it exists.

class ironic_inspector.common.lldp_parsers.**LLDPdot1Parser**(*node_info*, *nv=None*)
Bases: *ironic_inspector.common.lldp_parsers.LLDPParser*

Class to handle parsing of 802.1Q TLVs

add_dot1_port_protocol_vlan(*struct*, *name*, *data*)
Handle dot1_PORT_PROTOCOL_VLANID

add_dot1_protocol_identities(*struct*, *name*, *data*)
Handle dot1_PROTOCOL_IDENTITY

There can be multiple protocol ids TLVs, store in list

add_dot1_vlans(*struct*, *name*, *data*)
Handle dot1_VLAN_NAME

There can be multiple vlan TLVs, add dictionary entry with id/vlan to list.

class ironic_inspector.common.lldp_parsers.**LLDPdot3Parser**(*node_info*, *nv=None*)
Bases: *ironic_inspector.common.lldp_parsers.LLDPParser*

Class to handle parsing of 802.3 TLVs

add_dot3_macphy_config(*struct*, *name*, *data*)
Handle dot3_MACPHY_CONFIG_STATUS

ironic_inspector.common.lldp_tlvs module

Link Layer Discovery Protocol TLVs

`ironic_inspector.common.lldp_tlvs.`**bytes_to_int**(*obj*)
Convert bytes to an integer

Param obj - array of bytes

`ironic_inspector.common.lldp_tlvs.`**get_autoneg_cap**(*pmd*)
Get autonegotiated capability strings

This returns a list of capability strings from the Physical Media Dependent (PMD) capability bits.

Parameters **pmd** PMD bits

Returns Sorted ist containing capability strings

`ironic_inspector.common.lldp_tlvs.`**mapping_for_enum**(*mapping*)
Return tuple used for keys as a dict

Param mapping - dict with tuple as keys

`ironic_inspector.common.lldp_tlvs.`**mapping_for_switch**(*mapping*)
Return dict from values

Param mapping - dict with tuple as keys

ironic_inspector.common.locking module

class `ironic_inspector.common.locking.`**BaseLock**
Bases: `object`

abstract acquire(*blocking=True*)
Acquire lock.

abstract is_locked()
Return lock status

abstract release()
Release lock.

class `ironic_inspector.common.locking.`**InternalLock**(*uuid*)
Bases: *ironic_inspector.common.locking.BaseLock*

Locking mechanism based on threading.Semaphore.

acquire(*blocking=True*)
Acquire lock.

is_locked()
Return lock status

release()
Release lock.

class `ironic_inspector.common.locking.`**`ToozLock`**(*lock*)

Bases: *`ironic_inspector.common.locking.BaseLock`*

Wrapper on tooz locks.

`acquire`(*blocking=True*)

Acquire lock.

`is_locked`()

Return lock status

`release`()

Release lock.

`ironic_inspector.common.locking.`**`get_lock`**(*uuid*)

ironic_inspector.common.rpc module

`ironic_inspector.common.rpc.`**`get_client`**(*topic=None*)

Get a RPC client instance.

Parameters **`topic`** The topic of the message will be delivered to. This argument is ignored if CONF.standalone is True.

`ironic_inspector.common.rpc.`**`get_server`**(*endpoints*)

Get a RPC server instance.

`ironic_inspector.common.rpc.`**`init`**()

ironic_inspector.common.rpc_service module

class `ironic_inspector.common.rpc_service.`**`RPCService`**(*host*)

Bases: `oslo_service.service.Service`

`start`()

Start a service.

`stop`()

Stop a service.

Parameters **`graceful`** indicates whether to wait for all threads to finish or terminate them instantly

ironic_inspector.common.service_utils module

`ironic_inspector.common.service_utils.`**`prepare_service`**(*args=None*)

ironic_inspector.common.swift module

class ironic_inspector.common.swift.**SwiftAPI**

Bases: object

API for communicating with Swift.

create_object(*object*, *data*, *container=None*, *headers=None*)

Uploads a given string to Swift.

Parameters

- **object** The name of the object in Swift
- **data** string data to put in the object
- **container** The name of the container for the object. Defaults to the value set in the configuration options.
- **headers** the headers for the object to pass to Swift

Returns The Swift UUID of the object

Raises utils.Error, if any operation with Swift fails.

get_object(*object*, *container=None*)

Downloads a given object from Swift.

Parameters

- **object** The name of the object in Swift
- **container** The name of the container for the object. Defaults to the value set in the configuration options.

Returns Swift object

Raises utils.Error, if the Swift operation fails.

ironic_inspector.common.swift.**get_introspection_data**(*uuid*, *suffix=None*)

Downloads introspection data from Swift.

Parameters

- **uuid** UUID of the Ironic node that the data came from
- **suffix** optional suffix to add to the underlying swift object name

Returns Swift object with the introspection data

ironic_inspector.common.swift.**reset_swift_session**()

Reset the global session variable.

Mostly useful for unit tests.

ironic_inspector.common.swift.**store_introspection_data**(*data*, *uuid*, *suffix=None*)

Uploads introspection data to Swift.

Parameters

- **data** data to store in Swift

- **uuid** UUID of the Ironic node that the data came from
- **suffix** optional suffix to add to the underlying swift object name

Returns name of the Swift object that the data is stored in

Module contents

ironic_inspector.conductor package

Submodules

ironic_inspector.conductor.manager module

class `ironic_inspector.conductor.manager.`**`ConductorManager`**

Bases: `object`

ironic inspector conductor manager

`RPC_API_VERSION = '1.3'`

`del_host``()`

Shutdown the ironic inspector conductor service.

`do_abort``(`***kwargs*`)`

`do_continue``(`***kwargs*`)`

`do_introspection``(`***kwargs*`)`

`do_reapply``(`***kwargs*`)`

`init_host``()`

Initialize Worker host

Init db connection, load and validate processing hooks, runs periodic tasks.

:returns None

`target = <Target version=1.3>`

`ironic_inspector.conductor.manager.`**`periodic_clean_up`**`()`

`ironic_inspector.conductor.manager.`**`periodic_leader_election`**`(`*conductor*`)`

`ironic_inspector.conductor.manager.`**`sync_with_ironic`**`(`*conductor*`)`

Module contents

ironic_inspector.conf package

Submodules

ironic_inspector.conf.accelerators module

`ironic_inspector.conf.accelerators.`**`list_opts`**`()`

`ironic_inspector.conf.accelerators.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.capabilities module

`ironic_inspector.conf.capabilities.`**`list_opts`**`()`

`ironic_inspector.conf.capabilities.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.coordination module

`ironic_inspector.conf.coordination.`**`list_opts`**`()`

`ironic_inspector.conf.coordination.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.default module

`ironic_inspector.conf.default.`**`list_opts`**`()`

`ironic_inspector.conf.default.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.discovery module

`ironic_inspector.conf.discovery.`**`list_opts`**`()`

`ironic_inspector.conf.discovery.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.dnsmasq_pxe_filter module

`ironic_inspector.conf.dnsmasq_pxe_filter.`**`list_opts`**`()`

`ironic_inspector.conf.dnsmasq_pxe_filter.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.extra_hardware module

`ironic_inspector.conf.extra_hardware.`**`list_opts`**`()`

`ironic_inspector.conf.extra_hardware.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.iptables module

`ironic_inspector.conf.iptables.`**`list_opts`**`()`

`ironic_inspector.conf.iptables.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.ironic module

`ironic_inspector.conf.ironic.`**`list_opts`**`()`

`ironic_inspector.conf.ironic.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.opts module

`ironic_inspector.conf.opts.`**`list_opts`**`()`

`ironic_inspector.conf.opts.`**`parse_args`**`(`*args*, *default_config_files=None*`)`

`ironic_inspector.conf.opts.`**`set_config_defaults`**`()`
: Return a list of oslo.config options available in Inspector code.

`ironic_inspector.conf.opts.`**`set_cors_middleware_defaults`**`()`
: Update default configuration options for oslo.middleware.

ironic_inspector.conf.pci_devices module

`ironic_inspector.conf.pci_devices.`**`list_opts`**`()`

`ironic_inspector.conf.pci_devices.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.port_physnet module

`ironic_inspector.conf.port_physnet.`**`list_opts`**`()`

`ironic_inspector.conf.port_physnet.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.processing module

`ironic_inspector.conf.processing.`**`list_opts`**`()`

`ironic_inspector.conf.processing.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.pxe_filter module

`ironic_inspector.conf.pxe_filter.`**`list_opts`**`()`

`ironic_inspector.conf.pxe_filter.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.service_catalog module

`ironic_inspector.conf.service_catalog.`**`list_opts`**`()`

`ironic_inspector.conf.service_catalog.`**`register_opts`**`(`*conf*`)`

ironic_inspector.conf.swift module

`ironic_inspector.conf.swift.`**`list_opts`**`()`

`ironic_inspector.conf.swift.`**`register_opts`**`(`*conf*`)`

Module contents

ironic_inspector.plugins package

Submodules

ironic_inspector.plugins.accel_device module

Gather and distinguish Accelerator PCI devices from inventory.

`class` `ironic_inspector.plugins.accel_device.`**`AccelDevicesHook`**

Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Processing hook for distinguishing accelerator devices.

`before_update``(`*introspection_data*, *node_info*, ***kwargs*`)`

Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **`introspection_data`** processed data from the ramdisk.
- **`node_info`** NodeInfo instance.
- **`kwargs`** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

ironic_inspector.plugins.base module

Base code for plugins support.

class ironic_inspector.plugins.base.**ProcessingHook**

Bases: object

Abstract base class for introspection data processing hooks.

before_processing(*introspection_data*, ***kwargs*)

Hook to run before any other data processing.

This hook is run even before sanity checks.

Parameters

- **introspection_data** raw information sent by the ramdisk, may be modified by the hook.
- **kwargs** used for extensibility without breaking existing hooks

Returns nothing.

before_update(*introspection_data*, *node_info*, ***kwargs*)

Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **introspection_data** processed data from the ramdisk.
- **node_info** NodeInfo instance.
- **kwargs** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

dependencies = []

An ordered list of hooks that must be enabled before this one.

The items here should be entry point names, not classes.

class ironic_inspector.plugins.base.**RuleActionPlugin**

Bases: *ironic_inspector.plugins.base.WithValidation*

Abstract base class for rule action plugins.

FORMATTED_PARAMS = []

List of params will be formatted with python format.

abstract apply(*node_info*, *params*, ***kwargs*)

Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class `ironic_inspector.plugins.base.`**`RuleConditionPlugin`**

Bases: *`ironic_inspector.plugins.base.WithValidation`*

Abstract base class for rule condition plugins.

`ALLOW_NONE = False`

Whether this condition accepts None when field is not found.

`REQUIRED_PARAMS = {'value'}`

`abstract check`(*node_info*, *field*, *params*, ***kwargs*)

Check if condition holds for a given field.

Parameters

- **`node_info`** NodeInfo object
- **`field`** field value
- **`params`** parameters as a dictionary, changing it here will change what will be stored in database
- **`kwargs`** used for extensibility without breaking existing plugins

Raises **`ValueError`** on unacceptable field value

Returns True if check succeeded, otherwise False

class `ironic_inspector.plugins.base.`**`WithValidation`**

Bases: `object`

`OPTIONAL_PARAMS = {}`

Set with names of optional parameters.

`REQUIRED_PARAMS = {}`

Set with names of required parameters.

`validate`(*params*, ***kwargs*)

Validate params passed during creation.

Default implementation checks for presence of fields from REQUIRED_PARAMS and fails for unexpected fields (not from REQUIRED_PARAMS + OPTIONAL_PARAMS).

Parameters

- **`params`** params as a dictionary
- **`kwargs`** used for extensibility without breaking existing plugins

Raises ValueError on validation failure

`ironic_inspector.plugins.base.`**`introspection_data_manager`**()

`ironic_inspector.plugins.base.`**`missing_entrypoints_callback`**(*names*)

Raise MissingHookError with comma-separated list of missing hooks

`ironic_inspector.plugins.base.`**`node_not_found_hook_manager`**(**args*)

`ironic_inspector.plugins.base.`**`processing_hooks_manager`**(**args*)

Create a Stevedore extension manager for processing hooks.

Parameters **`args`** arguments to pass to the hooks constructor.

`ironic_inspector.plugins.base.`**`reset`**`()`
Reset cached managers.

`ironic_inspector.plugins.base.`**`rule_actions_manager`**`()`
Create a Stevedore extension manager for actions in rules.

`ironic_inspector.plugins.base.`**`rule_conditions_manager`**`()`
Create a Stevedore extension manager for conditions in rules.

`ironic_inspector.plugins.base.`**`validate_processing_hooks`**`()`
Validate the enabled processing hooks.

Raises MissingHookError on missing or failed to load hooks

Raises RuntimeError on validation failure

Returns the list of hooks passed validation

ironic_inspector.plugins.base_physnet module

`class` `ironic_inspector.plugins.base_physnet.`**`BasePhysnetHook`**
Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Base class for plugins that assign a physical network to ports.

The mechanism for mapping a port to a physical network should be provided by a subclass via the get_physnet() method.

`before_update``(`*introspection_data*, *node_info*, ***kwargs*`)`
Process introspection data and patch port physical network.

`abstract get_physnet``(`*port*, *iface_name*, *introspection_data*`)`
Return a physical network to apply to a port.

Subclasses should implement this method to determine how to map a port to a physical network.

Parameters

- **`port`** The ironic port to patch.
- **`iface_name`** Name of the interface.
- **`introspection_data`** Introspection data.

Returns The physical network to set, or None.

ironic_inspector.plugins.capabilities module

Gather capabilities from inventory.

`class` `ironic_inspector.plugins.capabilities.`**`CapabilitiesHook`**
Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Processing hook for detecting capabilities.

`before_update``(`*introspection_data*, *node_info*, ***kwargs*`)`
Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **`introspection_data`** processed data from the ramdisk.
- **`node_info`** NodeInfo instance.
- **`kwargs`** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

ironic_inspector.plugins.discovery module

Enroll node not found hook hook.

`ironic_inspector.plugins.discovery.`**`enroll_node_not_found_hook`**(*introspection_data*, ***kwargs*)

ironic_inspector.plugins.example module

Example plugin.

`class` `ironic_inspector.plugins.example.`**`ExampleProcessingHook`**

Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

`before_processing`(*introspection_data*, ***kwargs*)

Hook to run before any other data processing.

This hook is run even before sanity checks.

Parameters

- **`introspection_data`** raw information sent by the ramdisk, may be modified by the hook.
- **`kwargs`** used for extensibility without breaking existing hooks

Returns nothing.

`before_update`(*introspection_data*, *node_info*, ***kwargs*)

Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **`introspection_data`** processed data from the ramdisk.
- **`node_info`** NodeInfo instance.
- **`kwargs`** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

class ironic_inspector.plugins.example.**ExampleRuleAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

apply(*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

ironic_inspector.plugins.example.**example_not_found_hook**(*introspection_data*, ***kwargs*)

ironic_inspector.plugins.extra_hardware module

Plugin to store extra hardware information in Swift.

Stores the value of the data key returned by the ramdisk as a JSON encoded string in a Swift object. The object is named extra_hardware-<node uuid> and is stored in the inspector container.

class ironic_inspector.plugins.extra_hardware.**ExtraHardwareHook**
Bases: *ironic_inspector.plugins.base.ProcessingHook*

Processing hook for saving extra hardware information in Swift.

before_update(*introspection_data*, *node_info*, ***kwargs*)
Stores the data key from introspection_data in Swift.

If the data key exists, updates Ironic extra column hardware_swift_object key to the name of the Swift object, and stores the data in the inspector container in Swift.

Otherwise, it does nothing.

ironic_inspector.plugins.introspection_data module

Backends for storing introspection data.

class ironic_inspector.plugins.introspection_data.**BaseStorageBackend**
Bases: object

abstract get(*node_uuid*, *processed=True*, *get_json=False*)
Get introspected data from storage backend.

Parameters

- **node_uuid** node UUID.
- **processed** Specify whether the data to be retrieved is processed or not.
- **get_json** Specify whether return the introspection data in json format, string value is returned if False.

Returns the introspection data.

Raises IntrospectionDataStoreDisabled if storage backend is disabled.

abstract save(*node_uuid*, *data*, *processed=True*)
Save introspected data to storage backend.

Parameters

- **node_uuid** node UUID.
- **data** the introspected data to be saved, in dict format.
- **processed** Specify whether the data to be saved is processed or not.

Raises IntrospectionDataStoreDisabled if storage backend is disabled.

class `ironic_inspector.plugins.introspection_data.`**DatabaseStore**
Bases: `object`

get(*node_uuid*, *processed=True*, *get_json=False*)

save(*node_uuid*, *data*, *processed=True*)

class `ironic_inspector.plugins.introspection_data.`**NoStore**
Bases: *ironic_inspector.plugins.introspection_data.BaseStorageBackend*

get(*node_uuid*, *processed=True*, *get_json=False*)
Get introspected data from storage backend.

Parameters

- **node_uuid** node UUID.
- **processed** Specify whether the data to be retrieved is processed or not.
- **get_json** Specify whether return the introspection data in json format, string value is returned if False.

Returns the introspection data.

Raises IntrospectionDataStoreDisabled if storage backend is disabled.

save(*node_uuid*, *data*, *processed=True*)
Save introspected data to storage backend.

Parameters

- **node_uuid** node UUID.
- **data** the introspected data to be saved, in dict format.
- **processed** Specify whether the data to be saved is processed or not.

Raises IntrospectionDataStoreDisabled if storage backend is disabled.

class `ironic_inspector.plugins.introspection_data.`**SwiftStore**
Bases: `object`

get(*node_uuid*, *processed=True*, *get_json=False*)

save(*node_uuid*, *data*, *processed=True*)

ironic_inspector.plugins.lldp_basic module

LLDP Processing Hook for basic TLVs

class `ironic_inspector.plugins.lldp_basic.`**`LLDPBasicProcessingHook`**

Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Process mandatory and optional LLDP packet fields

Loop through raw LLDP TLVs and parse those from the basic management, 802.1, and 802.3 TLV sets. Store parsed data back to the ironic-inspector database.

`before_update`(*introspection_data*, *node_info*, ***kwargs*)

Process LLDP data and update all_interfaces with processed data

ironic_inspector.plugins.local_link_connection module

Generic LLDP Processing Hook

class `ironic_inspector.plugins.local_link_connection.`**`GenericLocalLinkConnectionH`**

Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Process mandatory LLDP packet fields

Non-vendor specific LLDP packet fields processed for each NIC found for a baremetal node, port ID and chassis ID. These fields if found and if valid will be saved into the local link connection info port id and switch id fields on the Ironic port that represents that NIC.

`before_update`(*introspection_data*, *node_info*, ***kwargs*)

Process LLDP data and patch Ironic port local link connection

ironic_inspector.plugins.pci_devices module

Gather and distinguish PCI devices from inventory.

class `ironic_inspector.plugins.pci_devices.`**`PciDevicesHook`**

Bases: *`ironic_inspector.plugins.base.ProcessingHook`*

Processing hook for counting and distinguishing various PCI devices.

That information can be later used by nova for node scheduling.

`aliases = {}`

`before_update`(*introspection_data*, *node_info*, ***kwargs*)

Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **`introspection_data`** processed data from the ramdisk.
- **`node_info`** NodeInfo instance.
- **`kwargs`** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

ironic_inspector.plugins.physnet_cidr_map module

Port Physical Network Hook

class ironic_inspector.plugins.physnet_cidr_map.**PhysnetCidrMapHook**

Bases: *ironic_inspector.plugins.base_physnet.BasePhysnetHook*

Process port physical network

Set the physical_network field of baremetal ports based on a cidr to physical network mapping in the configuration.

get_physnet (*port*, *iface_name*, *introspection_data*)

Return a physical network to apply to a port.

Parameters

- **port** The ironic port to patch.
- **iface_name** Name of the interface.
- **introspection_data** Introspection data.

Returns The physical network to set, or None.

ironic_inspector.plugins.raid_device module

Gather root device hint from recognized block devices.

class ironic_inspector.plugins.raid_device.**RaidDeviceDetection**

Bases: *ironic_inspector.plugins.base.ProcessingHook*

Processing hook for learning the root device after RAID creation.

The plugin can figure out the root device in 2 runs. First, it saves the discovered block device serials in node.extra. The second run will check the difference between the recently discovered block devices and the previously saved ones. After saving the root device in node.properties, it will delete the temporarily saved block device serials in node.extra.

This way, it helps to figure out the root device hint in cases when otherwise Ironic doesnt have enough information to do so. Such a usecase is DRAC RAID configuration where the BMC doesnt provide any useful information about the created RAID disks. Using this plugin immediately before and after creating the root RAID device will solve the issue of root device hints.

In cases where theres no RAID volume on the node, the standard plugin will fail due to the missing local_gb value. This plugin fakes the missing value, until its corrected during later runs. Note, that for this to work the plugin needs to take precedence over the standard plugin.

before_processing (*introspection_data*, ***kwargs*)

Adds fake local_gb value if its missing from introspection_data.

before_update (*introspection_data*, *node_info*, ***kwargs*)

Hook to run before Ironic node update.

This hook is run after node is found and ports are created, just before the node is updated with the data.

Parameters

- **introspection_data** processed data from the ramdisk.
- **node_info** NodeInfo instance.
- **kwargs** used for extensibility without breaking existing hooks.

Returns nothing.

[RFC 6902] - http://tools.ietf.org/html/rfc6902

ironic_inspector.plugins.rules module

Standard plugins for rules API.

class ironic_inspector.plugins.rules.**AddTraitAction**

Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

REQUIRED_PARAMS = {'name'}

apply(*node_info*, *params*, ***kwargs*)

Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class ironic_inspector.plugins.rules.**ContainsCondition**

Bases: *ironic_inspector.plugins.rules.ReCondition*

check(*node_info*, *field*, *params*, ***kwargs*)

Check if condition holds for a given field.

Parameters

- **node_info** NodeInfo object
- **field** field value
- **params** parameters as a dictionary, changing it here will change what will be stored in database
- **kwargs** used for extensibility without breaking existing plugins

Raises **ValueError** on unacceptable field value

Returns True if check succeeded, otherwise False

class ironic_inspector.plugins.rules.**EmptyCondition**

Bases: *ironic_inspector.plugins.base.RuleConditionPlugin*

ALLOW_NONE = True

REQUIRED_PARAMS = {}

check (*node_info*, *field*, *params*, ***kwargs*)
Check if condition holds for a given field.

Parameters

- **node_info** NodeInfo object
- **field** field value
- **params** parameters as a dictionary, changing it here will change what will be stored in database
- **kwargs** used for extensibility without breaking existing plugins

Raises **ValueError** on unacceptable field value

Returns True if check succeeded, otherwise False

class ironic_inspector.plugins.rules.**EqCondition**
Bases: *ironic_inspector.plugins.rules.SimpleCondition*

op (*b*, */*)
Same as a == b.

class ironic_inspector.plugins.rules.**ExtendAttributeAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

FORMATTED_PARAMS = ['value']

OPTIONAL_PARAMS = {'unique'}

REQUIRED_PARAMS = {'path', 'value'}

apply (*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class ironic_inspector.plugins.rules.**FailAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

REQUIRED_PARAMS = {'message'}

apply (*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class `ironic_inspector.plugins.rules.`**`GeCondition`**
Bases: *`ironic_inspector.plugins.rules.SimpleCondition`*

`op`(*b*, */*)
Same as a >= b.

class `ironic_inspector.plugins.rules.`**`GtCondition`**
Bases: *`ironic_inspector.plugins.rules.SimpleCondition`*

`op`(*b*, */*)
Same as a > b.

class `ironic_inspector.plugins.rules.`**`LeCondition`**
Bases: *`ironic_inspector.plugins.rules.SimpleCondition`*

`op`(*b*, */*)
Same as a <= b.

class `ironic_inspector.plugins.rules.`**`LtCondition`**
Bases: *`ironic_inspector.plugins.rules.SimpleCondition`*

`op`(*b*, */*)
Same as a < b.

class `ironic_inspector.plugins.rules.`**`MatchesCondition`**
Bases: *`ironic_inspector.plugins.rules.ReCondition`*

`check`(*node_info*, *field*, *params*, ***kwargs*)
Check if condition holds for a given field.

Parameters

- **`node_info`** NodeInfo object
- **`field`** field value
- **`params`** parameters as a dictionary, changing it here will change what will be stored in database
- **`kwargs`** used for extensibility without breaking existing plugins

Raises `ValueError` on unacceptable field value

Returns True if check succeeded, otherwise False

class `ironic_inspector.plugins.rules.`**`NeCondition`**
Bases: *`ironic_inspector.plugins.rules.SimpleCondition`*

`op`(*b*, */*)
Same as a != b.

class `ironic_inspector.plugins.rules.`**`NetCondition`**
Bases: *`ironic_inspector.plugins.base.RuleConditionPlugin`*

`check`(*node_info*, *field*, *params*, ***kwargs*)
Check if condition holds for a given field.

Parameters

- **`node_info`** NodeInfo object
- **`field`** field value

- **params** parameters as a dictionary, changing it here will change what will be stored in database
- **kwargs** used for extensibility without breaking existing plugins

Raises ValueError on unacceptable field value

Returns True if check succeeded, otherwise False

validate(*params*, ***kwargs*)
Validate params passed during creation.

Default implementation checks for presence of fields from REQUIRED_PARAMS and fails for unexpected fields (not from REQUIRED_PARAMS + OPTIONAL_PARAMS).

Parameters

- **params** params as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises ValueError on validation failure

class ironic_inspector.plugins.rules.**ReCondition**
Bases: *ironic_inspector.plugins.base.RuleConditionPlugin*

validate(*params*, ***kwargs*)
Validate params passed during creation.

Default implementation checks for presence of fields from REQUIRED_PARAMS and fails for unexpected fields (not from REQUIRED_PARAMS + OPTIONAL_PARAMS).

Parameters

- **params** params as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises ValueError on validation failure

class ironic_inspector.plugins.rules.**RemoveTraitAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

REQUIRED_PARAMS = {'name'}

apply(*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class ironic_inspector.plugins.rules.**SetAttributeAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

FORMATTED_PARAMS = ['value']

OPTIONAL_PARAMS = {'reset_interfaces', 'value'}

REQUIRED_PARAMS = {'path'}

apply (*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

validate (*params*, ***kwargs*)
Validate params passed during creation.

Default implementation checks for presence of fields from REQUIRED_PARAMS and fails for unexpected fields (not from REQUIRED_PARAMS + OPTIONAL_PARAMS).

Parameters

- **params** params as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises ValueError on validation failure

class ironic_inspector.plugins.rules.**SetCapabilityAction**
Bases: *ironic_inspector.plugins.base.RuleActionPlugin*

FORMATTED_PARAMS = ['value']

OPTIONAL_PARAMS = {'value'}

REQUIRED_PARAMS = {'name'}

apply (*node_info*, *params*, ***kwargs*)
Run action on successful rule match.

Parameters

- **node_info** NodeInfo object
- **params** parameters as a dictionary
- **kwargs** used for extensibility without breaking existing plugins

Raises utils.Error on failure

class ironic_inspector.plugins.rules.**SimpleCondition**
Bases: *ironic_inspector.plugins.base.RuleConditionPlugin*

check (*node_info*, *field*, *params*, ***kwargs*)
Check if condition holds for a given field.

Parameters

- **node_info** NodeInfo object
- **field** field value
- **params** parameters as a dictionary, changing it here will change what will be stored in database

- **kwargs** used for extensibility without breaking existing plugins

Raises ValueError on unacceptable field value

Returns True if check succeeded, otherwise False

op = None

ironic_inspector.plugins.rules.**coerce**(*value*, *expected*)

ironic_inspector.plugins.standard module

Standard set of plugins.

class ironic_inspector.plugins.standard.**RamdiskErrorHook**

Bases: *ironic_inspector.plugins.base.ProcessingHook*

Hook to process error send from the ramdisk.

before_processing(*introspection_data*, ***kwargs*)

Hook to run before any other data processing.

This hook is run even before sanity checks.

Parameters

- **introspection_data** raw information sent by the ramdisk, may be modified by the hook.
- **kwargs** used for extensibility without breaking existing hooks

Returns nothing.

class ironic_inspector.plugins.standard.**RootDiskSelectionHook**

Bases: *ironic_inspector.plugins.base.ProcessingHook*

Smarter root disk selection using Ironic root device hints.

This hook must always go before SchedulerHook, otherwise root_disk field might not be updated.

before_update(*introspection_data*, *node_info*, ***kwargs*)

Process root disk information.

class ironic_inspector.plugins.standard.**SchedulerHook**

Bases: *ironic_inspector.plugins.base.ProcessingHook*

Nova scheduler required properties.

KEYS = ('cpus', 'cpu_arch', 'memory_mb')

before_update(*introspection_data*, *node_info*, ***kwargs*)

Update node with scheduler properties.

class ironic_inspector.plugins.standard.**ValidateInterfacesHook**

Bases: *ironic_inspector.plugins.base.ProcessingHook*

Hook to validate network interfaces.

before_processing(*introspection_data*, ***kwargs*)

Validate information about network interfaces.

before_update(*introspection_data*, *node_info*, ***kwargs*)
Create new ports and drop ports that are not present in the data.

Module contents

ironic_inspector.pxe_filter package

Submodules

ironic_inspector.pxe_filter.base module

Base code for PXE boot filtering.

class ironic_inspector.pxe_filter.base.**BaseFilter**
Bases: *ironic_inspector.pxe_filter.interface.FilterDriver*

The generic PXE boot filtering interface implementation.

This driver doesnt do anything but provides a basic synchronization and initialization logic for some drivers to reuse. Subclasses have to provide a custom sync() method.

fsm = <automaton.machines.FiniteMachine object>

fsm_reset_on_error()
Reset the filter driver upon generic exception.

The context is self.fsm. The automaton.exceptions.NotFound error is cast to the InvalidFilterDriverState error. Other exceptions trigger self.reset()

Raises InvalidFilterDriverState

Returns nothing.

get_periodic_sync_task()
Get periodic sync task for the filter.

The periodic task returned is casting the InvalidFilterDriverState to the periodics.NeverAgain exception to quit looping.

Raises periodics.NeverAgain

Returns a periodic task to be run in the background.

init_filter()
Base driver initialization logic. Locked.

Raises InvalidFilterDriverState

Returns nothing.

reset()
Reset internal driver state.

This method is called by the fsm_context manager upon exception as well as by the tear_down_filter method. A subclass might wish to override as necessary, though must not lock the driver. The overriding subclass should up-call.

Returns nothing.

property state
Current driver state.

sync(*ironic*)
Base driver sync logic. Locked.

Parameters **ironic** obligatory ironic client instance

Returns nothing.

tear_down_filter()
Base driver tear down logic. Locked.

Returns nothing.

class `ironic_inspector.pxe_filter.base.`**Events**
Bases: `object`

PXE filter driver transitions.

initialize = 'initialize'

reset = 'reset'

sync = 'sync'

exception `ironic_inspector.pxe_filter.base.`**InvalidFilterDriverState**
Bases: `RuntimeError`

The fsm of the filter driver raised an error.

class `ironic_inspector.pxe_filter.base.`**NoopFilter**
Bases: *`ironic_inspector.pxe_filter.base.BaseFilter`*

A trivial PXE boot filter.

get_periodic_sync_task()
Get periodic sync task for the filter.

The periodic task returned is casting the InvalidFilterDriverState to the periodics.NeverAgain exception to quit looping.

Raises periodics.NeverAgain

Returns a periodic task to be run in the background.

class `ironic_inspector.pxe_filter.base.`**States**
Bases: `object`

PXE filter driver states.

initialized = 'initialized'

uninitialized = 'uninitialized'

`ironic_inspector.pxe_filter.base.`**driver**()
Get the driver for the PXE filter.

Returns the singleton PXE filter driver object.

`ironic_inspector.pxe_filter.base.`**get_active_macs**(*ironic*)

`ironic_inspector.pxe_filter.base.`**get_inactive_macs**(*ironic*)

`ironic_inspector.pxe_filter.base.`**get_ironic_macs**(*ironic*)

`ironic_inspector.pxe_filter.base.`**`locked_driver_event`**`(`*event*`)`

Call driver method having processed the fsm event.

ironic_inspector.pxe_filter.dnsmasq module

`class` `ironic_inspector.pxe_filter.dnsmasq.`**`DnsmasqFilter`**

Bases: *`ironic_inspector.pxe_filter.base.BaseFilter`*

The dnsmasq PXE filter driver.

A pxe filter driver implementation that controls access to dnsmasq through amending its configuration.

`init_filter``()`

Performs an initial sync with ironic and starts dnsmasq.

The initial _sync() call reduces the chances dnsmasq might lose some inotify deny list events by prefetching the list before dnsmasq is started.

Raises OSError, IOError.

Returns None.

`reset``()`

Stop dnsmasq and upcall reset.

`sync``(`*ironic*`)`

Sync dnsmasq configuration with current Ironic&Inspector state.

Polls all ironic ports. Those being inspected, the active ones, are added to the allow list while the rest are added to the deny list in the dnsmasq configuration.

Parameters **`ironic`** an ironic client instance.

Raises OSError, IOError.

Returns None.

ironic_inspector.pxe_filter.interface module

The code of the PXE boot filtering interface.

`class` `ironic_inspector.pxe_filter.interface.`**`FilterDriver`**

Bases: `object`

The PXE boot filtering interface.

`abstract get_periodic_sync_task``()`

Get periodic sync task for the filter.

Returns a periodic task to be run in the background.

`abstract init_filter``()`

Initialize the internal driver state.

This method should be idempotent and may perform system-wide filter state changes. Can be synchronous.

Returns nothing.

abstract sync(*ironic*)

Synchronize the filter with ironic and inspector.

To be called both periodically and as needed by inspector. The filter should tear down its internal state if the sync method raises in order to propagate filtering exception between periodic and on-demand sync call. To this end, a driver should raise from the sync call if its internal state isnt properly initialized.

Parameters **ironic** an ironic client instance.

Returns nothing.

abstract tear_down_filter()

Reset the filter.

This method should be idempotent and may perform system-wide filter state changes. Can be synchronous.

Returns nothing.

ironic_inspector.pxe_filter.iptables module

class ironic_inspector.pxe_filter.iptables.**IptablesFilter**

Bases: *ironic_inspector.pxe_filter.base.BaseFilter*

A PXE boot filtering interface implementation.

init_filter()

Base driver initialization logic. Locked.

Raises InvalidFilterDriverState

Returns nothing.

reset()

Reset internal driver state.

This method is called by the fsm_context manager upon exception as well as by the tear_down_filter method. A subclass might wish to override as necessary, though must not lock the driver. The overriding subclass should up-call.

Returns nothing.

sync(*ironic*)

Sync firewall filter rules for introspection.

Gives access to PXE boot port for any machine, except for those, whose MAC is registered in Ironic and is not on introspection right now.

This function is called from both introspection initialization code and from periodic task. This function is supposed to be resistant to unexpected iptables state.

`init()` function must be called once before any call to this function. This function is using `eventlet` semaphore to serialize access from different green threads.

Parameters **ironic** an ironic client instance.

Returns nothing.

Module contents

Submodules

ironic_inspector.api_tools module

Generic Rest Api tools.

`ironic_inspector.api_tools.`**`limit_field`**`(`*value*`)`
Fetch the pagination limit field from flask.request.args.

Returns the limit

`ironic_inspector.api_tools.`**`marker_field`**`(`*value*`)`
Fetch the pagination marker field from flask.request.args.

Returns an uuid

`ironic_inspector.api_tools.`**`raises_coercion_exceptions`**`(`*fn*`)`
Convert coercion function exceptions to utils.Error.

Raises utils.Error when the coercion function raises an AssertionError or a ValueError

`ironic_inspector.api_tools.`**`request_field`**`(`*field_name*`)`
Decorate a function that coerces the specified field.

Parameters **`field_name`** name of the field to fetch

Returns a decorator

ironic_inspector.db module

SQLAlchemy models for inspection data and shared database code.

`class` `ironic_inspector.db.`**`Attribute`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

`name`

`node_uuid`

`uuid`

`value`

`class` `ironic_inspector.db.`**`IntrospectionData`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

`data`

`processed`

`uuid`

`class` `ironic_inspector.db.`**`ModelBase`**
Bases: `oslo_db.sqlalchemy.models.ModelBase`

`class` `ironic_inspector.db.`**`Node`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

error

finished_at

manage_boot

started_at

state

uuid

version_id

class ironic_inspector.db.**Option**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

name

uuid

value

class ironic_inspector.db.**Rule**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

actions

conditions

created_at

description

disabled

scope

uuid

class ironic_inspector.db.**RuleAction**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

action

as_dict()

id

params

rule

class ironic_inspector.db.**RuleCondition**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

as_dict()

field

id

invert

multiple

op

params

rule

ironic_inspector.db.**ensure_transaction**(*session=None*)

ironic_inspector.db.**get_context_manager**()
Create transaction context manager lazily.

Returns The transaction context manager.

ironic_inspector.db.**get_reader_session**()
Help method to get reader session.

Returns The reader session.

ironic_inspector.db.**get_writer_session**()
Help method to get writer session.

Returns The writer session.

ironic_inspector.db.**init**()
Initialize the database.

Method called on service start up, initialize transaction context manager and try to create db session.

ironic_inspector.db.**model_query**(*model*, **args*, ***kwargs*)
Query helper for simpler session usage.

Parameters **session** if present, the session to use

ironic_inspector.introspect module

Handling introspection request.

ironic_inspector.introspect.**abort**(*node_id*, *token=None*)
Abort running introspection.

Parameters

- **node_id** node UUID or name
- **token** authentication token

Raises Error

ironic_inspector.introspect.**introspect**(*node_id*, *manage_boot=True*, *token=None*)
Initiate hardware properties introspection for a given node.

Parameters

- **node_id** node UUID or name
- **manage_boot** whether to manage boot for this node
- **token** authentication token

Raises Error

ironic_inspector.introspection_state module

Introspection state.

class ironic_inspector.introspection_state.**Events**

Bases: object

Events that change introspection state.

abort = 'abort'

abort_end = 'abort_end'

classmethod all()

Return a list of all events.

error = 'error'

finish = 'finish'

process = 'process'

reapply = 'reapply'

start = 'start'

timeout = 'timeout'

wait = 'wait'

class ironic_inspector.introspection_state.**States**

Bases: object

States of an introspection.

aborting = 'aborting'

classmethod all()

Return a list of all states.

enrolling = 'enrolling'

error = 'error'

finished = 'finished'

processing = 'processing'

reapplying = 'reapplying'

starting = 'starting'

waiting = 'waiting'

ironic_inspector.main module

ironic_inspector.main.**add_version_headers**(*res*)

ironic_inspector.main.**api**(*path*, *is_public_api=False*, *rule=None*, *verb_to_rule_map=None*, ***flask_kwargs*)

Decorator to wrap api methods.

Performs flask routing, exception conversion, generation of oslo context for request and API access policy enforcement.

Parameters

- **path** flask app route path
- **is_public_api** whether this API path should be treated as public, with minimal access enforcement
- **rule** API access policy rule to enforce. If rule is None, the default policy rule will be enforced, which is deny all if not overridden in policy confif file.
- **verb_to_rule_map** if both rule and this are given, defines mapping between http verbs (uppercase) and strings to format the rule string with
- **kwargs** all the rest kwargs are passed to flask app.route

ironic_inspector.main.**api_continue**()

ironic_inspector.main.**api_introspection**(*node_id*)

ironic_inspector.main.**api_introspection_abort**(*node_id*)

ironic_inspector.main.**api_introspection_data**(*node_id*)

ironic_inspector.main.**api_introspection_reapply**(*node_id*)

ironic_inspector.main.**api_introspection_statuses**()

ironic_inspector.main.**api_introspection_unprocessed_data**(*node_id*)

ironic_inspector.main.**api_root**()

ironic_inspector.main.**api_rule**(*uuid*)

ironic_inspector.main.**api_rules**()

ironic_inspector.main.**check_api_version**()

ironic_inspector.main.**convert_exceptions**(*func*)

ironic_inspector.main.**create_link_object**(*urls*)

ironic_inspector.main.**error_response**(*exc*, *code=500*)

ironic_inspector.main.**generate_introspection_status**(*node*)

Return a dict representing current node status.

Parameters **node** a NodeInfo instance

Returns dictionary

ironic_inspector.main.**generate_resource_data**(*resources*)

ironic_inspector.main.**get_app**()

Get the flask instance.

`ironic_inspector.main.`**`get_client_compat`**`()`

`ironic_inspector.main.`**`get_random_topic`**`()`

`ironic_inspector.main.`**`handle_404`**`(`*error*`)`

`ironic_inspector.main.`**`rule_repr`**`(`*rule*, *short*`)`

`ironic_inspector.main.`**`start_coordinator`**`()`
Create a coordinator instance for non-standalone case.

`ironic_inspector.main.`**`version_root`**`(`*version*`)`

ironic_inspector.node_cache module

Cache for nodes currently under introspection.

`class` `ironic_inspector.node_cache.`**`NodeInfo`**`(`*uuid*, *version_id=None*, *state=None*, *started_at=None*, *finished_at=None*, *error=None*, *node=None*, *ports=None*, *ironic=None*, *manage_boot=True*`)`

Bases: `object`

Record about a node in the cache.

This class optionally allows to acquire a lock on a node. Note that the class instance itself is NOT thread-safe, you need to create a new instance for every thread.

`acquire_lock``(`*blocking=True*`)`
Acquire a lock on the associated node.

Exits with success if a lock is already acquired using this NodeInfo object.

Parameters **`blocking`** if True, wait for lock to be acquired, otherwise return immediately.

Returns boolean value, whether lock was acquired successfully

`add_attribute``(`*name*, *value*, *session=None*`)`
Store look up attribute for a node in the database.

Parameters

- **`name`** attribute name
- **`value`** attribute value or list of possible values
- **`session`** optional existing database session

`add_trait``(`*trait*, *ironic=None*`)`
Add a trait to the node.

Parameters

- **`trait`** trait to add
- **`ironic`** Ironic client to use instead of self.ironic

`property attributes`
Node look up attributes as a dict.

commit()
Commit current node status into the database.

create_ports(*ports*, *ironic=None*)
Create one or several ports for this node.

Parameters

- **ports** List of ports with all their attributes e.g [{mac: xx, ip: xx, client_id: None}, {mac: xx, ip: None, client_id: None}] It also support the old style of list of macs. A warning is issued if port already exists on a node.
- **ironic** Ironic client to use instead of self.ironic

delete_port(*port*, *ironic=None*)
Delete port.

Parameters

- **port** port object or its MAC
- **ironic** Ironic client to use instead of self.ironic

finished(*event*, *error=None*)
Record status for this node and process a terminal transition.

Also deletes look up attributes from the cache.

Parameters

- **event** the event to process
- **error** error message

classmethod from_row(*row*, *ironic=None*, *node=None*)
Construct NodeInfo from a database row.

fsm_event(*event*, *strict=False*)
Update node_info.state based on a fsm.process_event(event) call.

An AutomatonException triggers an error event. If strict, node_info.finished(istate.Events.error, error=str(exc)) is called with the AutomatonException instance and a EventError raised.

Parameters **event** an event to process by the fsm

Strict whether to fail the introspection upon an invalid event

Raises NodeStateInvalidEvent

get_by_path(*path*)
Get field value by ironic-style path (e.g. /extra/foo).

Parameters **path** path to a field

Returns field value

Raises KeyError if field was not found

invalidate_cache()
Clear all cached info, so that its reloaded next time.

property ironic
Ironic client instance.

property manage_boot
Whether to manage boot for this node.

node(*ironic=None*)
Get Ironic node object associated with the cached node record.

property options
Node introspection options as a dict.

patch(*patches*, *ironic=None*, ***kwargs*)
Apply JSON patches to a node.

Refreshes cached node instance.

Parameters

- **patches** JSON patches to apply
- **ironic** Ironic client to use instead of self.ironic
- **kwargs** Arguments to pass to ironicclient.

Raises openstacksdk exceptions

patch_port(*port*, *patches*, *ironic=None*)
Apply JSON patches to a port.

Parameters

- **port** port object or its MAC
- **patches** JSON patches to apply
- **ironic** Ironic client to use instead of self.ironic

ports(*ironic=None*)
Get Ironic port objects associated with the cached node record.

This value is cached as well, use invalidate_cache() to clean.

Returns dict MAC -> port object

release_lock()
Release a lock on a node.

Does nothing if lock was not acquired using this NodeInfo object.

remove_trait(*trait*, *ironic=None*)
Remove a trait from the node.

Parameters

- **trait** trait to add
- **ironic** Ironic client to use instead of self.ironic

replace_field(*path*, *func*, ***kwargs*)
Replace a field on ironic node.

Parameters

- **path** path to a field as used by the ironic client
- **func** function accepting an old value and returning a new one

- **kwargs** if default value is passed here, it will be used when no existing value is found.

Raises KeyError if value is not found and default is not set

Raises everything that patch() may raise

set_option(*name*, *value*)
Set an option for a node.

property state
State of the node_info object.

update_capabilities(*ironic=None*, ***caps*)
Update capabilities on a node.

Parameters

- **caps** capabilities to update
- **ironic** Ironic client to use instead of self.ironic

update_properties(*ironic=None*, ***props*)
Update properties on a node.

Parameters

- **props** properties to update
- **ironic** Ironic client to use instead of self.ironic

property version_id
Get the version id

ironic_inspector.node_cache.**active_macs**()
List all MACs that are on introspection right now.

ironic_inspector.node_cache.**add_node**(*uuid*, *state*, *manage_boot=True*, ***attributes*)
Store information about a node under introspection.

All existing information about this node is dropped. Empty values are skipped.

Parameters

- **uuid** Ironic node UUID
- **state** The initial state of the node
- **manage_boot** whether to manage boot for this node
- **attributes** attributes known about this node (like macs, BMC etc); also ironic client instance may be passed under ironic

Returns NodeInfo

ironic_inspector.node_cache.**clean_up**()
Clean up the cache.

Finish introspection for timed out nodes.

Returns list of timed out node UUIDs `ironic_inspector.node_cache.`**`create_node`**(*driver*, *ironic=None*, ***attributes*)

Create ironic node and cache it.

- Create new node in ironic.
- Cache it in inspector.
- Sets node_info state to enrolling.

Parameters

- **`driver`** driver for Ironic node.
- **`ironic`** ironic client instance.
- **`attributes`** dict, additional keyword arguments to pass to the ironic client on node creation.

Returns NodeInfo, or None in case error happened.

`ironic_inspector.node_cache.`**`delete_nodes_not_in_list`**(*uuids*)

Delete nodes which dont exist in Ironic node UUIDs.

Parameters **`uuids`** Ironic node UUIDs

`ironic_inspector.node_cache.`**`find_node`**(***attributes*)

Find node in cache.

Looks up a node based on attributes in a best-match fashion. This function acquires a lock on a node.

Parameters **`attributes`** attributes known about this node (like macs, BMC etc) also ironic client instance may be passed under ironic

Returns structure NodeInfo with attributes `uuid` and `created_at`

Raises Error if node is not found or multiple nodes match the attributes

`ironic_inspector.node_cache.`**`fsm_event_after`**(*event*, *strict=False*)

Trigger an fsm event after the function execution.

It is assumed the first function arg of the decorated function is always a NodeInfo instance.

Parameters

- **`event`** the event to process after the function call
- **`strict`** make an invalid fsm event trigger an error event

`ironic_inspector.node_cache.`**`fsm_event_before`**(*event*, *strict=False*)

Trigger an fsm event before the function execution.

It is assumed the first function arg of the decorated function is always a NodeInfo instance.

Parameters

- **`event`** the event to process before the function call
- **`strict`** make an invalid fsm event trigger an error event

`ironic_inspector.node_cache.`**`fsm_transition`**(*event*, *reentrant=True*, ***exc_kwargs*)

Decorate a function to perform a (non-)reentrant transition.

If True, reentrant transition will be performed at the end of a function call. If False, the transition will be performed before the function call. The function is decorated with the triggers_fsm_error_transition decorator as well.

Parameters

- **event** the event to bind the transition to.
- **reentrant** whether the transition is reentrant.
- **exc_kwargs** passed on to the triggers_fsm_error_transition decorator

`ironic_inspector.node_cache.`**`get_introspection_data`**(*node_id*, *processed=True*)

Get introspection data for this node.

Parameters

- **node_id** node UUID.
- **processed** Specify the type of introspected data, set to False indicates retrieving the unprocessed data.

Returns A dictionary representation of intropsected data

`ironic_inspector.node_cache.`**`get_node`**(*node_id*, *ironic=None*)

Get node from cache.

Parameters

- **node_id** node UUID or name.
- **ironic** optional ironic client instance

Returns structure NodeInfo.

`ironic_inspector.node_cache.`**`get_node_list`**(*ironic=None*, *marker=None*, *limit=None*)

Get node list from the cache.

The list of the nodes is ordered based on the (started_at, uuid) attribute pair, newer items first.

Parameters

- **ironic** optional ironic client instance
- **marker** pagination marker (an UUID or None)
- **limit** pagination limit; None for default CONF.api_max_limit

Returns a list of NodeInfo instances.

`ironic_inspector.node_cache.`**`introspection_active`**()

Check if introspection is active for at least one node.

`ironic_inspector.node_cache.`**`record_node`**(*ironic=None*, *bmc_addresses=None*, *macs=None*)

Create a cache record for a known active node.

Parameters

- **ironic** ironic client instance.
- **bmc_addresses** list of BMC addresses.
- **macs** list of MAC addresses.

Returns NodeInfo

`ironic_inspector.node_cache.`**`release_lock`**(*func*)

Decorate a node_info-function to release the node_info lock.

Assumes the first parameter of the function func is always a NodeInfo instance.

`ironic_inspector.node_cache.`**`start_introspection`**(*uuid*, ***kwargs*)

Start the introspection of a node.

If a node_info record exists in the DB, a start transition is used rather than dropping the record in order to check for the start transition validity in particular node state.

Parameters

- **`uuid`** Ironic node UUID
- **`kwargs`** passed on to add_node()

Raises NodeStateInvalidEvent in case the start transition is invalid in the current node state

Raises NodeStateRaceCondition if a mismatch was detected between the node_info cache and the DB

Returns NodeInfo

`ironic_inspector.node_cache.`**`store_introspection_data`**(*node_id*, *introspection_data*, *processed=True*)

Store introspection data for this node.

Parameters

- **`node_id`** node UUID.
- **`introspection_data`** A dictionary of introspection data
- **`processed`** Specify the type of introspected data, set to False indicates the data is unprocessed.

`ironic_inspector.node_cache.`**`triggers_fsm_error_transition`**(*errors=(<class 'Exception'>,), no_errors=(<class 'ironic_inspector.utils.NodeState <class 'ironic_inspector.utils.NodeState*

Trigger an fsm error transition upon certain errors.

It is assumed the first function arg of the decorated function is always a NodeInfo instance.

Parameters

- **`errors`** a tuple of exceptions upon which an error event is triggered. Re-raised.
- **`no_errors`** a tuple of exceptions that wont trigger the error event.

ironic_inspector.policy module

`ironic_inspector.policy.`**`authorize`**(*rule*, *target*, *creds*, **args*, ***kwargs*)

A shortcut for policy.Enforcer.authorize()

Checks authorization of a rule against the target and credentials, and raises an exception if the rule is not defined. args and kwargs are passed directly to oslo.policy Enforcer.authorize Always returns True if CONF.auth_strategy != keystone.

Parameters

- **`rule`** name of a registered oslo.policy rule
- **`target`** dict-like structure to check rule against
- **`creds`** dict of policy values from request

Returns True if request is authorized against given policy, False otherwise

Raises oslo_policy.policy.PolicyNotRegistered if supplied policy is not registered in oslo_policy

`ironic_inspector.policy.`**`get_enforcer`**()

Provides access to the single instance of Policy enforcer.

`ironic_inspector.policy.`**`get_oslo_policy_enforcer`**()

Get the enforcer instance to generate policy files.

This method is for use by oslopolicy CLI scripts. Those scripts need the output-file and namespace options, but having those in sys.argv means loading the inspector config options will fail as those are not expected to be present. So we pass in an arg list with those stripped out.

`ironic_inspector.policy.`**`init_enforcer`**(*policy_file=None*, *rules=None*, *default_rule=None*, *use_conf=True*)

Synchronously initializes the policy enforcer

Parameters

- **`policy_file`** Custom policy file to use, if none is specified, *CONF.oslo_policy.policy_file* will be used.
- **`rules`** Default dictionary / Rules to use. It will be considered just in the first instantiation.
- **`default_rule`** Default rule to use, CONF.oslo_policy.policy_default_rule will be used if none is specified.
- **`use_conf`** Whether to load rules from config file.

`ironic_inspector.policy.`**`list_policies`**()

Get list of all policies defined in code.

Used to register them all at runtime, and by oslo-config-generator to generate sample policy files.

ironic_inspector.process module

Handling introspection data from the ramdisk.

`ironic_inspector.process.`**`get_introspection_data`**(*uuid*, *processed=True*, *get_json=False*)

Get introspection data from the storage backend.

Parameters

- **`uuid`** node UUID
- **`processed`** Indicates the type of introspection data to be read, set True to request processed introspection data.
- **`get_json`** Specify whether return the introspection data in json format, string value is returned if False.

Raises utils.Error

`ironic_inspector.process.`**`process`**(*introspection_data*)

Process data from the ramdisk.

This function heavily relies on the hooks to do the actual data processing.

`ironic_inspector.process.`**`reapply`**(*node_uuid*, *data=None*)

Re-apply introspection steps.

Re-apply preprocessing, postprocessing and introspection rules on stored data.

Parameters

- **`node_uuid`** node UUID
- **`data`** unprocessed introspection data to be reapplied

Raises utils.Error

`ironic_inspector.process.`**`store_introspection_data`**(*node_uuid*, *data*, *processed=True*)

Store introspection data to the storage backend.

Parameters

- **`node_uuid`** node UUID
- **`data`** Introspection data to be saved
- **`processed`** The type of introspection data, set to True means the introspection data is processed, otherwise unprocessed.

Raises utils.Error

ironic_inspector.rules module

Support for introspection rules.

class `ironic_inspector.rules.`**`IntrospectionRule`**(*uuid*, *conditions*, *actions*, *description*, *scope=None*)

Bases: `object`

High-level class representing an introspection rule.

`apply_actions`(*node_info*, *data=None*)
Run actions on a node.

Parameters

- **`node_info`** NodeInfo instance
- **`data`** introspection data

`as_dict`(*short=False*)

`check_conditions`(*node_info*, *data*)
Check if conditions are true for a given node.

Parameters

- **`node_info`** a NodeInfo object
- **`data`** introspection data

Returns True if conditions match, otherwise False

`check_scope`(*node_info*)
Check if nodes scope falls under rule._scope and rule is applicable

Parameters **`node_info`** a NodeInfo object

Returns True if conditions match, otherwise False

property description

`ironic_inspector.rules.`**`actions_schema`**()

`ironic_inspector.rules.`**`apply`**(*node_info*, *data*)
Apply rules to a node.

`ironic_inspector.rules.`**`conditions_schema`**()

`ironic_inspector.rules.`**`create`**(*conditions_json*, *actions_json*, *uuid=None*, *description=None*, *scope=None*)
Create a new rule in database.

Parameters

- **`conditions_json`** list of dicts with the following keys: * op - operator * field - JSON path to field to compare Other keys are stored as is.
- **`actions_json`** list of dicts with the following keys: * action - action type Other keys are stored as is.
- **`uuid`** rule UUID, will be generated if empty
- **`description`** human-readable rule description

- **`scope`** if scope on node and rule matches, rule applies; if its empty, rule applies to all nodes.

Returns new IntrospectionRule object

Raises utils.Error on failure

`ironic_inspector.rules.`**`delete`**`(`*uuid*`)`
Delete a rule by its UUID.

`ironic_inspector.rules.`**`delete_all`**`()`
Delete all rules.

`ironic_inspector.rules.`**`get`**`(`*uuid*`)`
Get a rule by its UUID.

`ironic_inspector.rules.`**`get_all`**`()`
List all rules.

ironic_inspector.utils module

`class` `ironic_inspector.utils.`**`DeferredBasicAuthMiddleware`**`(`*app*, *auth_file*`)`
Bases: `object`

Middleware which sets X-Identity-Status header based on authentication

`exception` `ironic_inspector.utils.`**`Error`**`(`*msg*, *code=400*, *log_level='error'*, ***kwargs*`)`
Bases: `Exception`

Inspector exception.

`exception` `ironic_inspector.utils.`**`IntrospectionDataNotFound`**`(`*msg*, *code=404*, ***kwargs*`)`
Bases: *`ironic_inspector.utils.NotFoundInCacheError`*

Introspection data not found.

`exception` `ironic_inspector.utils.`**`IntrospectionDataStoreDisabled`**`(`*msg*, *code=400*, *log_level='error'*, ***kwargs*`)`
Bases: *`ironic_inspector.utils.Error`*

Introspection data store is disabled.

`exception` `ironic_inspector.utils.`**`NoAvailableConductor`**`(`*msg*, ***kwargs*`)`
Bases: *`ironic_inspector.utils.Error`*

No available conductor in the service group.

`exception` `ironic_inspector.utils.`**`NodeStateInvalidEvent`**`(`*msg*, *code=400*, *log_level='error'*, ***kwargs*`)`
Bases: *`ironic_inspector.utils.Error`*

Invalid event attempted.

exception `ironic_inspector.utils.`**`NodeStateRaceCondition`**(**args*, ***kwargs*)

Bases: `ironic_inspector.utils.Error`

State mismatch between the DB and a node_info.

exception `ironic_inspector.utils.`**`NotFoundInCacheError`**(*msg*, *code=404*, ***kwargs*)

Bases: `ironic_inspector.utils.Error`

Exception when node was not found in cache during processing.

class `ironic_inspector.utils.`**`ProcessingLoggerAdapter`**(*logger*, *extra*)

Bases: `oslo_log.log.KeywordArgumentAdapter`

`process`(*msg*, *kwargs*)

Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, youll only need to override this one method in a LoggerAdapter subclass for your specific needs.

`ironic_inspector.utils.`**`add_auth_middleware`**(*app*)

Add authentication middleware to Flask application.

Parameters **`app`** application.

`ironic_inspector.utils.`**`add_basic_auth_middleware`**(*app*)

Add HTTP Basic authentication middleware to Flask application.

Parameters **`app`** application.

`ironic_inspector.utils.`**`add_cors_middleware`**(*app*)

Create a CORS wrapper

Attach ironic-inspector-specific defaults that must be included in all CORS responses.

Parameters **`app`** application

`ironic_inspector.utils.`**`check_auth`**(*request*, *rule=None*, *target=None*)

Check authentication on request.

Parameters

- **`request`** Flask request
- **`rule`** policy rule to check the request against
- **`target`** dict-like structure to check rule against

Raises utils.Error if access is denied

`ironic_inspector.utils.`**`executor`**()

Return the current futures executor.

`ironic_inspector.utils.`**`getProcessingLogger`**(*name*)

`ironic_inspector.utils.`**`get_inventory`**(*data*, *node_info=None*)

Get and validate the hardware inventory from introspection data.

`ironic_inspector.utils.`**`get_ipmi_address_from_data`**(*introspection_data*) `ironic_inspector.utils.`**`get_ipmi_v6address_from_data`**(*introspection_data*)

`ironic_inspector.utils.`**`get_pxe_mac`**(*introspection_data*)

`ironic_inspector.utils.`**`get_valid_macs`**(*data*)

Get a list of valid MACs from the introspection data.

`ironic_inspector.utils.`**`iso_timestamp`**(*timestamp=None*, *tz=<UTC>*)

Return an ISO8601-formatted timestamp (tz: UTC) or None.

Parameters

- **`timestamp`** such as time.time() or None
- **`tz`** timezone

Returns an ISO8601-formatted timestamp, or None

`ironic_inspector.utils.`**`processing_logger_prefix`**(*data=None*, *node_info=None*)

Calculate prefix for logging.

Tries to use: * node UUID, node._state * node PXE MAC, * node BMC address

Parameters

- **`data`** introspection data
- **`node_info`** NodeInfo or ironic node object

Returns logging prefix as a string

ironic_inspector.version module

ironic_inspector.wsgi_service module

`class` `ironic_inspector.wsgi_service.`**`WSGIService`**

Bases: `oslo_service.service.Service`

Provides ability to launch API from wsgi app.

`reset`()

Reset server greenpool size to default.

Returns None

`start`()

Start serving this service using loaded configuration.

Returns None

`stop`()

Stop serving this API.

Returns None

`wait`()

Wait for the service to stop serving this API.

Returns None

Module contents

4.1.8 Ironic Inspector CI

Its important to understand the role of each job in the CI. To facilitate that, we have created the documentation below.

Jobs description

The description of each jobs that runs in the CI when you submit a patch for *openstack/ironic-inspector* is shown in the following table.

Note: All jobs are configured to use a pre-build tinyipa ramdisk, a wholedisk image that is downloaded from a Swift temporary url, *pxe* boot and *ipmi* driver.

Table 1: Table. OpenStack Ironic Inspector CI jobs description

Job name	Description
ironic-inspector-grenade	Deploys Ironic and Ironic Inspector in DevStack and runs upgrade for all enabled services.
ironic-inspector-tempest	Deploys Ironic and Ironic Inspector in DevStack. Runs tempest tests that match the regex *InspectorBasicTest* and deploys 1 virtual baremetal.
ironic-inspector-tempest-discovery	Deploys Ironic and Ironic Inspector in DevStack. Runs tempest tests that match the regex *InspectorDiscoveryTest* and deploys 1 virtual baremetal.
ironic-inspector-tempest-python3	Deploys Ironic and Ironic Inspector in DevStack under Python3. Runs tempest tests that match the regex *Inspector* and deploys 1 virtual baremetal.
openstack-tox-functional-py36	Run tox-based functional tests for Ironic Inspector under Python3.6
bifrost-integration-tinyipa-ubuntu-xenial	Tests the integration between Ironic Inspector and Bifrost.
ironic-inspector-tox-bandit	Runs bandit security tests in a tox environment to find known issues in the Ironic Inspector code.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX