Ironic Lib Documentation

Release 4.6.5.dev2

OpenStack Foundation

Mar 18, 2024

CONTENTS

Ironic-lib is a library for use by projects under Bare Metal governance only. This documentation is intended for developer use only. If you are looking for documentation for deployers, please see the ironic documentation.

CHAPTER

ONE

WELCOME TO IRONIC-LIB!

1.1 Overview

Ironic-lib is a library for use by projects under Bare Metal governance only. This documentation is intended for developer use only. If you are looking for documentation for deployers, please see the ironic documentation.

1.2 Metrics

Ironic-lib provides a pluggable metrics library as of the 2.0.0 release. Current provided backends are the default, noop, which discards all data, and statsd, which emits metrics to a statsd daemon over the network. The metrics backend to be used is configured via `CONF.metrics.backend`. How this configuration is set in practice may vary by project.

The typical usage of metrics is to initialize and cache a metrics logger, using the *get_metrics_logger()* method in *ironic_lib.metrics_utils*, then use that object to decorate functions or create context managers to gather metrics. The general convention is to provide the name of the module as the first argument to set it as the prefix, then set the actual metric name to the method name. For example:

```
from ironic_lib import metrics_utils

METRICS = metrics_utils.get_metrics_logger(__name__)

@METRICS.timer('my_simple_method')
def my_simple_method(arg, matey):
    pass

def my_complex_method(arg, matey):
    with METRICS.timer('complex_method_pt_1'):
        do_some_work()

    with METRICS.timer('complex_method_pt_2'):
        do_more_work()
```

There are three different kinds of metrics:

- **Timers** measure how long the code in the decorated method or context manager takes to execute, and emits the value as a timer metric. These are useful for measuring performance of a given block of code.

- **Counters** increment a counter each time a decorated method or context manager is executed. These are useful for counting the number of times a method is called, or the number of times an event occurs.
- **Gauges** return the value of a decorated method as a metric. This is useful when you want to monitor the value returned by a method over time.

Additionally, metrics can be sent directly, rather than using a context manager or decorator, when appropriate. When used in this way, ironic-lib will simply emit the value provided as the requested metric type. For example:

```
from ironic_lib import metrics_utils

METRICS = metrics_utils.get_metrics_logger(__name__)

def my_node_failure_method(node):
    if node.failed:
        METRICS.send_counter(node.uuid, 1)
```

The provided statsd backend natively supports all three metric types. For more information about how statsd changes behavior based on the metric type, see statsd metric types

CHAPTER

TWO

AUTOGENERATED API REFERENCE

2.1 ironic_lib

2.1.1 ironic_lib package

Subpackages

ironic_lib.common package

Submodules

ironic_lib.common.i18n module

Module contents

ironic_lib.json_rpc package

Submodules

ironic_lib.json_rpc.client module

A simple JSON RPC client.

This client is compatible with any JSON RPC 2.0 implementation, including ours.

class ironic_lib.json_rpc.client.**Client**(*serializer*, *version_cap=None*)

Bases: object

JSON RPC client with ironic exception handling.

allowed_exception_namespaces = ['ironic_lib.exception.', 'ironic.common.exce

can_send_version(*version*)

prepare(*topic*, *version=None*)

ironic_lib.json_rpc.server module

Implementation of JSON RPC for communication between API and conductors.

This module implementa a subset of JSON RPC 2.0 as defined in https://www.jsonrpc.org/specification. Main differences: * No support for batched requests. * No support for positional arguments passing. * No JSON RPC 1.0 fallback.

class ironic_lib.json_rpc.server.**EmptyContext**(*src*)

Bases: object

request_id = None

to_dict()

exception ironic_lib.json_rpc.server.**InvalidParams**(*message=None*, ***kwargs*)

Bases: *ironic_lib.json_rpc.server.JsonRpcError*

code = -32602

exception ironic_lib.json_rpc.server.**InvalidRequest**(*message=None*, ***kwargs*)

Bases: *ironic_lib.json_rpc.server.JsonRpcError*

code = -32600

exception ironic_lib.json_rpc.server.**JsonRpcError**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.json_rpc.server.**MethodNotFound**(*message=None*, ***kwargs*)

Bases: *ironic_lib.json_rpc.server.JsonRpcError*

code = -32601

exception ironic_lib.json_rpc.server.**ParseError**(*message=None*, ***kwargs*)

Bases: *ironic_lib.json_rpc.server.JsonRpcError*

code = -32700

class ironic_lib.json_rpc.server.**WSGIService**(*manager*, *serializer*, *context_class=<class 'ironic_lib.json_rpc.server.EmptyContext'>*)

Bases: oslo_service.service.Service

Provides ability to launch JSON RPC as a WSGI application.

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

ironic_lib.json_rpc.**auth_strategy**()

ironic_lib.json_rpc.**list_opts**()

ironic_lib.json_rpc.**register_opts**(*conf*)

Submodules

ironic_lib.auth_basic module

class ironic_lib.auth_basic.**BasicAuthMiddleware**(*app*, *auth_file*)
Bases: object

Middleware which performs HTTP basic authentication on requests

format_exception(*e*)

ironic_lib.auth_basic.**auth_entry**(*entry*, *password*)
Compare a password with a single user auth file entry

Param entry: Line from auth user file to use for authentication

Param password: Password encoded as bytes

Returns A dictionary of WSGI environment values to append to the request

Raises Unauthorized, if the entry doesnt match supplied password or if the entry is crypted with a method other than bcrypt

ironic_lib.auth_basic.**authenticate**(*auth_file*, *username*, *password*)
Finds username and password match in Apache style user auth file

The user auth file format is expected to comply with Apache documentation[1] however the bcrypt password digest is the *only* digest format supported.

[1] https://httpd.apache.org/docs/current/misc/password_encryptions.html

Param auth_file: Path to user auth file

Param username: Username to authenticate

Param password: Password encoded as bytes

Returns A dictionary of WSGI environment values to append to the request

Raises Unauthorized, if no file entries match supplied username/password

ironic_lib.auth_basic.**parse_entry**(*entry*)
Extrace the username and crypted password from a user auth file entry

Param entry: Line from auth user file to use for authentication

Returns a tuple of username and crypted password

Raises ConfigInvalid if the password is not in the supported bcrypt format

ironic_lib.auth_basic.**parse_header**(*env*)
Parse WSGI environment for Authorization header of type Basic

Param env: WSGI environment to get header from

Returns Token portion of the header value

Raises Unauthorized, if header is missing or if the type is not Basic

ironic_lib.auth_basic.**parse_token**(*token*)
Parse the token portion of the Authentication header value

Param token: Token value from basic authorization header

Returns tuple of username, password

Raises Unauthorized, if username and password could not be parsed for any reason

ironic_lib.auth_basic.**unauthorized**(*message=None*)
Raise an Unauthorized exception to prompt for basic authentication

Param message: Optional message for esception

Raises Unauthorized with WWW-Authenticate header set

ironic_lib.auth_basic.**validate_auth_file**(*auth_file*)
Read the auth user file and validate its correctness

Param auth_file: Path to user auth file

Raises ConfigInvalid on validation error

ironic_lib.capabilities module

Code for working with capabilities.

ironic_lib.capabilities.**combine**(*capabilities_dict*, *skip_none=False*)
Combine capabilities into the old format.

Parameters

- **capabilities_dict** Capabilities as a mapping.
- **skip_none** If True, skips all items with value of None.

Returns Capabilities as a string key1:value1,key2:value2.

ironic_lib.capabilities.**parse**(*capabilities*, *compat=True*, *skip_malformed=False*)
Extract capabilities from provided object.

The capabilities value can either be a dict, or a json str, or a key1:value1,key2:value2 formatted string (if compat is True). If None, an empty dictionary is returned.

Parameters

- **capabilities** The capabilities value. Can either be a dict, or a json str, or a key1:value1,key2:value2 formatted string (if compat is True).

- **compat** Whether to parse the old format key1:value1,key2:value2.
- **skip_malformed** Whether to skip malformed items or raise ValueError.

Returns A dictionary with the capabilities if found and well formatted, otherwise an empty dictionary.

Raises TypeError if the capabilities are of invalid type.

Raises ValueError on a malformed capability if skip_malformed is False or on invalid JSON with compat is False.

ironic_lib.capabilities.**update_and_combine**(*capabilities*, *new_values*, *skip_malformed=False*, *skip_none=False*)

Parses capabilities, updated them with new values and re-combines.

Parameters

- **capabilities** The capabilities value. Can either be a dict, or a json str, or a key1:value1,key2:value2 formatted string (if compat is True).
- **new_values** New values as a dictionary.
- **skip_malformed** Whether to skip malformed items or raise ValueError.
- **skip_none** If True, skips all items with value of None.

Returns Capabilities in the old format (key1:value1,key2:value2).

Raises TypeError if the capabilities are of invalid type.

Raises ValueError on a malformed capability if skip_malformed is False.

ironic_lib.disk_partitioner module

class ironic_lib.disk_partitioner.**DiskPartitioner**(*device*, *disk_label='msdos'*, *alignment='optimal'*)

Bases: object

add_partition(*size*, *part_type='primary'*, *fs_type=''*, *boot_flag=None*, *extra_flags=None*)

Add a partition.

Parameters

- **size** The size of the partition in MiB.
- **part_type** The type of the partition. Valid values are: primary, logical, or extended.
- **fs_type** The filesystem type. Valid types are: ext2, fat32, fat16, HFS, linux-swap, NTFS, reiserfs, ufs. If blank (), it will create a Linux native partition (83).
- **boot_flag** Boot flag that needs to be configured on the partition. Ignored if None. It can take values bios_grub, boot.
- **extra_flags** List of flags to set on the partition. Ignored if None.

Returns The partition number.

commit()
Write to the disk.

get_partitions()
Get the partitioning layout.

Returns An iterator with the partition number and the partition layout.

ironic_lib.disk_partitioner.**list_opts**()
Entry point for oslo-config-generator.

ironic_lib.disk_utils module

ironic_lib.disk_utils.**block_uuid**(*dev*)
Get UUID of a block device.

Try to fetch the UUID, if that fails, try to fetch the PARTUUID.

ironic_lib.disk_utils.**convert_image**(*source*, *dest*, *out_format*, *run_as_root=False*)
Convert image to other format.

ironic_lib.disk_utils.**count_mbr_partitions**(*device*)
Count the number of primary and logical partitions on a MBR

Parameters **device** The device path.

Returns A tuple with the number of primary partitions and logical partitions.

Raise ValueError if the device does not have a valid MBR partition table.

ironic_lib.disk_utils.**create_config_drive_partition**(*node_uuid*, *device*, *configdrive*)
Create a partition for config drive

Checks if the device is GPT or MBR partitioned and creates config drive partition accordingly.

Parameters

- **node_uuid** UUID of the Node.
- **device** The device path.
- **configdrive** Base64 encoded Gzipped configdrive content or configdrive HTTP URL.

Raises InstanceDeployFailure if config drive size exceeds maximum limit or if it fails to create config drive.

ironic_lib.disk_utils.**dd**(*src*, *dst*, *conv_flags=None*)
Execute dd from src to dst.

ironic_lib.disk_utils.**destroy_disk_metadata**(*dev*, *node_uuid*)
Destroy metadata structures on nodes disk.

Ensure that nodes disk magic strings are wiped without zeroing the entire drive. To do this we use the wipefs tool from util-linux.

Parameters

- **dev** Path for the device to work on.
- **node_uuid** Nodes uuid. Used for logging.

ironic_lib.disk_utils.**find_efi_partition**(*device*)
Looks for the EFI partition on a given device.

A boot partition on a GPT disk is assumed to be an EFI partition as well.

Parameters **device** the name of the device

Returns the EFI partition record from *list_partitions* or None

ironic_lib.disk_utils.**fix_gpt_partition**(*device*, *node_uuid*)
Fix GPT partition

Fix GPT table information when image is written to a disk which has a bigger extend (e.g. 30GB image written on a 60Gb physical disk).

Parameters

- **device** The device path.
- **node_uuid** UUID of the Node.

Raises InstanceDeployFailure if exception is caught.

ironic_lib.disk_utils.**get_dev_block_size**(*dev*)
Get the device size in 512 byte sectors.

ironic_lib.disk_utils.**get_device_information**(*device*, *probe=False*, *fields=None*)
Get information about a device using blkid.

Can be applied to all block devices: disks, RAID, partitions.

Parameters

- **device** Device name.
- **probe** DEPRECATED, do not use.
- **fields** A list of fields to request (all by default).

Returns A dictionary with requested fields as keys.

Raises ProcessExecutionError

ironic_lib.disk_utils.**get_disk_identifier**(*dev*)
Get the disk identifier from the disk being exposed by the ramdisk.

This disk identifier is appended to the pxe config which will then be used by chain.c32 to detect the correct disk to chainload. This is helpful in deployments to nodes with multiple disks.

http://www.syslinux.org/wiki/index.php/Comboot/chain.c32#mbr:

Parameters **dev** Path for the already populated disk device.

Raises **OSError** When the hexdump binary is unavailable.

Returns The Disk Identifier.

ironic_lib.disk_utils.**get_image_mb**(*image_path*, *virtual_size=True*)
Get size of an image in Megabyte.

ironic_lib.disk_utils.**get_partition_table_type**(*device*)
Get partition table type, msdos or gpt.

Parameters **device** the name of the device

Returns dos, gpt or None

ironic_lib.disk_utils.**get_uefi_disk_identifier**(*dev*)
Get the uuid from the disk being exposed by the ramdisk.

DEPRECATED: use find_efi_partition with get_device_information instead.

Parameters **dev** Path for the already populated disk device.

Raises ***InstanceDeployFailure*** Image is not UEFI bootable.

Returns The UUID of the partition.

ironic_lib.disk_utils.**is_block_device**(*dev*)
Check whether a device is block or not.

ironic_lib.disk_utils.**is_iscsi_device**(*dev*, *node_uuid=None*)
Check whether the device path belongs to an iSCSI device.

If node UUID is provided, checks that the device belongs to this UUID.

ironic_lib.disk_utils.**is_last_char_digit**(*dev*)
check whether device name ends with a digit

ironic_lib.disk_utils.**list_opts**()
Entry point for oslo-config-generator.

ironic_lib.disk_utils.**list_partitions**(*device*)
Get partitions information from given device.

Parameters **device** The device path.

Returns list of dictionaries (one per partition) with keys: number, start, end, size (in MiB), filesystem, partition_name, flags, path.

ironic_lib.disk_utils.**make_partitions**(*dev*, *root_mb*, *swap_mb*, *ephemeral_mb*, *configdrive_mb*, *node_uuid*, *commit=True*, *boot_option='netboot'*, *boot_mode='bios'*, *disk_label=None*, *cpu_arch=''*)
Partition the disk device.

Create partitions for root, swap, ephemeral and configdrive on a disk device.

Parameters

- **dev** Path for the device to work on.
- **root_mb** Size of the root partition in mebibytes (MiB).
- **swap_mb** Size of the swap partition in mebibytes (MiB). If 0, no partition will be created.
- **ephemeral_mb** Size of the ephemeral partition in mebibytes (MiB). If 0, no partition will be created.
- **configdrive_mb** Size of the configdrive partition in mebibytes (MiB). If 0, no partition will be created.

- **commit** True/False. Default for this setting is True. If False partitions will not be written to disk.
- **boot_option** Can be local or netboot. netboot by default.
- **boot_mode** Can be bios or uefi. bios by default.
- **node_uuid** Nodes uuid. Used for logging.
- **disk_label** The disk label to be used when creating the partition table. Valid values are: msdos, gpt or None; If None Ironic will figure it out according to the boot_mode parameter.
- **cpu_arch** Architecture of the node the disk device belongs to. When using the default value of None, no architecture specific steps will be taken. This default should be used for x86_64. When set to ppc64*, architecture specific steps are taken for booting a partition image locally.

Returns A dictionary containing the partition type as Key and partition path as Value for the partitions created by this method.

ironic_lib.disk_utils.**partition_index_to_path**(*device*, *index*)
Guess a partition path based on its device and index.

Parameters

- **device** Device path.
- **index** Partition index.

ironic_lib.disk_utils.**partprobe**(*device*, *attempts=None*)
Probe partitions on the given device.

Parameters

- **device** The block device containing paritions that is attempting to be updated.
- **attempts** Number of attempts to run partprobe, the default is read from the configuration.

Returns True on success, False otherwise.

ironic_lib.disk_utils.**populate_image**(*src*, *dst*, *conv_flags=None*)

ironic_lib.disk_utils.**qemu_img_info**(*path*)
Return an object containing the parsed output from qemu-img info.

ironic_lib.disk_utils.**trigger_device_rescan**(*device*, *attempts=None*)
Sync and trigger device rescan.

Disk partition performed via parted, when performed on a ramdisk do not have to honor the fsync mechanism. In essence, fsync is used on the file representing the block device, which falls to the kernel filesystem layer to trigger a sync event. On a ramdisk using ramfs, this is an explicit non-operation.

As a result of this, we need to trigger a system wide sync operation which will trigger cache to flush to disk, after which partition changes should be visible upon re-scan.

When ramdisks are not in use, this also helps ensure that data has been safely flushed across the wire, such as on iscsi connections.

Parameters

- **device** The block device containing paritions that is attempting to be updated.
- **attempts** Number of attempts to run partprobe, the default is read from the configuration.

Returns True on success, False otherwise.

ironic_lib.disk_utils.**udev_settle**()

Wait for the udev event queue to settle.

Wait for the udev event queue to settle to make sure all devices are detected once the machine boots up.

Returns True on success, False otherwise.

ironic_lib.disk_utils.**work_on_disk**(*dev*, *root_mb*, *swap_mb*, *ephemeral_mb*, *ephemeral_format*, *image_path*, *node_uuid*, *preserve_ephemeral=False*, *configdrive=None*, *boot_option='netboot'*, *boot_mode='bios'*, *tempdir=None*, *disk_label=None*, *cpu_arch=''*, *conv_flags=None*)

Create partitions and copy an image to the root partition.

Parameters

- **dev** Path for the device to work on.
- **root_mb** Size of the root partition in megabytes.
- **swap_mb** Size of the swap partition in megabytes.
- **ephemeral_mb** Size of the ephemeral partition in megabytes. If 0, no ephemeral partition will be created.
- **ephemeral_format** The type of file system to format the ephemeral partition.
- **image_path** Path for the instances disk image. If None, the root partition is prepared but not populated.
- **node_uuid** nodes uuid. Used for logging.
- **preserve_ephemeral** If True, no filesystem is written to the ephemeral block device, preserving whatever content it had (if the partition table has not changed).
- **configdrive** Optional. Base64 encoded Gzipped configdrive content or configdrive HTTP URL.
- **boot_option** Can be local or netboot. netboot by default.
- **boot_mode** Can be bios or uefi. bios by default.
- **tempdir** A temporary directory
- **disk_label** The disk label to be used when creating the partition table. Valid values are: msdos, gpt or None; If None Ironic will figure it out according to the boot_mode parameter.

- **cpu_arch** Architecture of the node the disk device belongs to. When using the default value of None, no architecture specific steps will be taken. This default should be used for x86_64. When set to ppc64*, architecture specific steps are taken for booting a partition image locally.
- **conv_flags** Flags that need to be sent to the dd command, to control the conversion of the original file when copying to the host. It can contain several options separated by commas.

Returns a dictionary containing the following keys: root uuid: UUID of root partition efi system partition uuid: UUID of the uefi system partition (if boot mode is uefi). *partitions*: mapping of partition types to their device paths. NOTE: If key exists but value is None, it means partition doesnt exist.

ironic_lib.exception module

Ironic base exception handling.

Includes decorator for re-raising Ironic-type exceptions.

SHOULD include dedicated exception logging.

exception ironic_lib.exception.**BadRequest**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

code = 400

exception ironic_lib.exception.**CatalogNotFound**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**ConfigInvalid**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**FileSystemNotSupported**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**InstanceDeployFailure**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**InvalidMetricConfig**(*message=None*, ***kwargs*)

Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**IronicException**(*message=None*, ***kwargs*)

Bases: Exception

Base Ironic Exception

To correctly use this class, inherit from it and define a _msg_fmt property. That _msg_fmt will get printfd with the keyword arguments provided to the constructor.

If you need to access the message from an exception you should use str(exc)

code = 500

headers = {}

safe = False

exception ironic_lib.exception.**KeystoneFailure**(*message=None*, ***kwargs*)
Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**KeystoneUnauthorized**(*message=None*, ***kwargs*)
Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**ServiceLookupFailure**(*message=None*, ***kwargs*)
Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**ServiceRegistrationFailure**(*message=None*, ***kwargs*)
Bases: *ironic_lib.exception.IronicException*

exception ironic_lib.exception.**Unauthorized**(*message=None*, ***kwargs*)
Bases: *ironic_lib.exception.IronicException*

code = 401

headers = {'WWW-Authenticate': 'Basic realm="Baremetal API"'}

ironic_lib.exception.**list_opts**()
Entry point for oslo-config-generator.

ironic_lib.keystone module

Central place for handling Keystone authorization and service lookup.

ironic_lib.keystone.**add_auth_opts**(*options*, *service_type=None*)
Add auth options to sample config

As these are dynamically registered at runtime, this adds options for most used auth_plugins when generating sample config.

ironic_lib.keystone.**get_adapter**(*group*, ***adapter_kwargs*)
Loads adapter from options in a configuration file section.

The adapter_kwargs will be passed directly to keystoneauth1 Adapter and will override the values loaded from config. Consult keystoneauth1 docs for available adapter options.

Parameters **group** name of the config section to load adapter options from

ironic_lib.keystone.**get_auth**(*group*, ***auth_kwargs*)
Loads auth plugin from options in a configuration file section.

The auth_kwargs will be passed directly to keystoneauth1 auth plugin and will override the values loaded from config. Note that the accepted kwargs will depend on auth plugin type as defined by [group]auth_type option. Consult keystoneauth1 docs for available auth plugins and their options.

Parameters **group** name of the config section to load auth plugin options from

ironic_lib.keystone.**get_endpoint**(*group*, ***adapter_kwargs*)
Get an endpoint from an adapter.

The adapter_kwargs will be passed directly to keystoneauth1 Adapter and will override the values loaded from config. Consult keystoneauth1 docs for available adapter options.

Parameters **group** name of the config section to load adapter options from

Raises CatalogNotFound if the endpoint is not found

ironic_lib.keystone.**get_service_auth**(*context*, *endpoint*, *service_auth*)
Create auth plugin wrapping both user and service auth.

When properly configured and using auth_token middleware, requests with valid service auth will not fail if the user token is expired.

Ideally we would use the plugin provided by auth_token middleware however this plugin isnt serialized yet.

ironic_lib.keystone.**get_session**(*group*, ***session_kwargs*)
Loads session object from options in a configuration file section.

The session_kwargs will be passed directly to keystoneauth1 Session and will override the values loaded from config. Consult keystoneauth1 docs for available options.

Parameters **group** name of the config section to load session options from

ironic_lib.keystone.**ks_exceptions**(*f*)
Wraps keystoneclient functions and centralizes exception handling.

ironic_lib.keystone.**register_auth_opts**(*conf*, *group*, *service_type=None*)
Register session- and auth-related options

Registers only basic auth options shared by all auth plugins. The rest are registered at runtime depending on auth plugin used.

ironic_lib.mdns module

Multicast DNS implementation for API discovery.

This implementation follows RFC 6763 as clarified by the API SIG guideline https://review.opendev.org/651222.

class ironic_lib.mdns.**Zeroconf**
Bases: object

Multicast DNS implementation client and server.

Uses threading internally, so there is no start method. It starts automatically on creation.

> **Warning:** The underlying library does not yet support IPv6.

close()
Shut down mDNS and unregister services.

> **Note:** If another server is running for the same services, it will re-register them immediately.

get_endpoint(*service_type*, *skip_loopback=True*, *skip_link_local=False*)
Get an endpoint and its properties from mDNS.

If the requested endpoint is already in the built-in server cache, and its TTL is not exceeded, the cached value is returned.

Parameters

- **service_type** OpenStack service type.
- **skip_loopback** Whether to ignore loopback addresses.
- **skip_link_local** Whether to ignore link local V6 addresses.

Returns tuple (endpoint URL, properties as a dict).

Raises *ServiceLookupFailure* if the service cannot be found.

register_service(*service_type*, *endpoint*, *params=None*)
Register a service.

This call announces the new services via multicast and instructs the built-in server to respond to queries about it.

Parameters

- **service_type** OpenStack service type, e.g. baremetal.
- **endpoint** full endpoint to reach the service.
- **params** optional properties as a dictionary.

Raises *ServiceRegistrationFailure* if the service cannot be registered, e.g. because of conflicts.

ironic_lib.mdns.**get_endpoint**(*service_type*)
Get an endpoint and its properties from mDNS.

If the requested endpoint is already in the built-in server cache, and its TTL is not exceeded, the cached value is returned.

Parameters **service_type** OpenStack service type.

Returns tuple (endpoint URL, properties as a dict).

Raises *ServiceLookupFailure* if the service cannot be found.

ironic_lib.mdns.**list_opts**()
Entry point for oslo-config-generator.

ironic_lib.metrics module

class ironic_lib.metrics.**Counter**(*metrics*, *name*, *sample_rate*)
Bases: object

A counter decorator and context manager.

This metric type increments a counter every time the decorated method or context manager is executed. It is bound to this MetricLogger. For example:

```
from ironic_lib import metrics_utils

METRICS = metrics_utils.get_metrics_logger()

@METRICS.counter('foo')
def foo(bar, baz):
    print bar, baz
```

(continues on next page)

(continued from previous page)

```
with METRICS.counter('foo'):
    do_something()
```

class ironic_lib.metrics.**Gauge**(*metrics*, *name*)

Bases: object

A gauge decorator.

This metric type returns the value of the decorated method as a metric every time the method is executed. It is bound to this MetricLogger. For example:

```
from ironic_lib import metrics_utils

METRICS = metrics_utils.get_metrics_logger()

@METRICS.gauge('foo')
def add_foo(bar, baz):
    return (bar + baz)
```

class ironic_lib.metrics.**MetricLogger**(*prefix=''*, *delimiter='.'*)

Bases: object

Abstract class representing a metrics logger.

A MetricLogger sends data to a backend (noop or statsd). The data can be a gauge, a counter, or a timer.

The data sent to the backend is composed of:

- a full metric name
- a numeric value

The format of the full metric name is: _prefix<delim>name

where:

- _prefix: [global_prefix<delim>][uuid<delim>][host_name<delim>]prefix
- name: the name of this metric
- <delim>: the delimiter. Default is .

counter(*name*, *sample_rate=None*)

gauge(*name*)

get_metric_name(*name*)

Get the full metric name.

The format of the full metric name is: _prefix<delim>name

where:

- _prefix: [global_prefix<delim>][uuid<delim>][host_name<delim>] prefix
- name: the name of this metric
- <delim>: the delimiter. Default is .

Parameters **name** The metric name.

Returns The full metric name, with logger prefix, as a string.

send_counter(*name*, *value*, *sample_rate=None*)
Send counter metric data.

Counters are used to count how many times an event occurred. The backend will increment the counter name by the value value.

Optionally, specify sample_rate in the interval [0.0, 1.0] to sample data probabilistically where:

```
P(send metric data) = sample_rate
```

If sample_rate is None, then always send metric data, but do not have the backend send sample rate information (if supported).

Parameters

- **name** Metric name
- **value** Metric numeric value that will be sent to the backend
- **sample_rate** Probabilistic rate at which the values will be sent. Value must be None or in the interval [0.0, 1.0].

send_gauge(*name*, *value*)
Send gauge metric data.

Gauges are simple values. The backend will set the value of gauge name to value.

Parameters

- **name** Metric name
- **value** Metric numeric value that will be sent to the backend

send_timer(*name*, *value*)
Send timer data.

Timers are used to measure how long it took to do something.

Parameters

- **m_name** Metric name
- **m_value** Metric numeric value that will be sent to the backend

timer(*name*)

class ironic_lib.metrics.**NoopMetricLogger**(*prefix=''*, *delimiter='.'*)
Bases: *ironic_lib.metrics.MetricLogger*

Noop metric logger that throws away all metric data.

class ironic_lib.metrics.**Timer**(*metrics*, *name*)
Bases: object

A timer decorator and context manager.

This metric type times the decorated method or code running inside the context manager, and emits the time as the metric value. It is bound to this MetricLogger. For example:

```
from ironic_lib import metrics_utils

METRICS = metrics_utils.get_metrics_logger()

@METRICS.timer('foo')
def foo(bar, baz):
    print bar, baz

with METRICS.timer('foo'):
    do_something()
```

ironic_lib.metrics_statsd module

class ironic_lib.metrics_statsd.**StatsdMetricLogger**(*prefix*, *delimiter='.'*, *host=None*, *port=None*)

Bases: *ironic_lib.metrics.MetricLogger*

Metric logger that reports data via the statsd protocol.

COUNTER_TYPE = 'c'

GAUGE_TYPE = 'g'

TIMER_TYPE = 'ms'

ironic_lib.metrics_statsd.**list_opts**()

Entry point for oslo-config-generator.

ironic_lib.metrics_utils module

ironic_lib.metrics_utils.**get_metrics_logger**(*prefix=''*, *backend=None*, *host=None*, *delimiter='.'*)

Return a metric logger with the specified prefix.

The format of the prefix is: [global_prefix<delim>][host_name<delim>]prefix where <delim> is the delimiter (default is .)

Parameters

- **prefix** Prefix for this metric logger. Value should be a string or None.
- **backend** Backend to use for the metrics system. Possible values are noop and statsd.
- **host** Name of this node.
- **delimiter** Delimiter to use for the metrics name.

Returns The new MetricLogger.

ironic_lib.metrics_utils.**list_opts**()

Entry point for oslo-config-generator.

ironic_lib.utils module

Utilities and helper functions.

ironic_lib.utils.**dd**(*src*, *dst*, **args*)
Execute dd from src to dst.

Parameters

- **src** the input file for dd command.
- **dst** the output file for dd command.
- **args** a tuple containing the arguments to be passed to dd command.

Raises processutils.ProcessExecutionError if it failed to run the process.

ironic_lib.utils.**execute**(**cmd*, *use_standard_locale=False*, *log_stdout=True*, ***kwargs*)
Convenience wrapper around oslos execute() method.

Executes and logs results from a system command. See docs for oslo_concurrency.processutils.execute for usage.

Parameters

- **cmd** positional arguments to pass to processutils.execute()
- **use_standard_locale** Defaults to False. If set to True, execute command with standard locale added to environment variables.
- **log_stdout** Defaults to True. If set to True, logs the output.
- **kwargs** keyword arguments to pass to processutils.execute()

Returns (stdout, stderr) from process execution

Raises UnknownArgumentError on receiving unknown arguments

Raises ProcessExecutionError

Raises OSError

ironic_lib.utils.**find_devices_by_hints**(*devices*, *root_device_hints*)
Find all devices that match the root device hints.

Try to find devices that match the root device hints. In order for a device to be matched it needs to satisfy all the given hints.

Parameters

- **devices**

  A list of dictionaries representing the devices containing one or more of the following keys:

  name (String) The device name, e.g /dev/sda

  size (Integer) Size of the device in *bytes*

  model (String) Device model

  vendor (String) Device vendor name

serial (String) Device serial number

wwn (String) Unique storage identifier

wwn_with_extension (String): Unique storage identifier with the vendor extension appended

wwn_vendor_extension (String): United vendor storage identifier

rotational (Boolean) Whether its a rotational device or not. Useful to distinguish HDDs (rotational) and SSDs (not rotational).

hctl (String): The SCSI address: Host, channel, target and lun. For example: 1:0:0:0.

by_path (String): The alternative device name, e.g. /dev/disk/by-path/pci-0000:00

- **root_device_hints** A dictionary with the root device hints.

Raises ValueError, if some information is invalid.

Returns A generator with all matching devices as dictionaries.

ironic_lib.utils.**get_route_source**(*dest*, *ignore_link_local=True*)
Get the IP address to send packages to destination.

ironic_lib.utils.**is_http_url**(*url*)

ironic_lib.utils.**list_opts**()
Entry point for oslo-config-generator.

ironic_lib.utils.**match_root_device_hints**(*devices*, *root_device_hints*)
Try to find a device that matches the root device hints.

Try to find a device that matches the root device hints. In order for a device to be matched it needs to satisfy all the given hints.

Parameters

- **devices**

A list of dictionaries representing the devices containing one or more of the following keys:

name (String) The device name, e.g /dev/sda

size (Integer) Size of the device in *bytes*

model (String) Device model

vendor (String) Device vendor name

serial (String) Device serial number

wwn (String) Unique storage identifier

wwn_with_extension (String): Unique storage identifier with the vendor extension appended

wwn_vendor_extension (String): United vendor storage identifier

rotational (Boolean) Whether its a rotational device or not. Useful to distinguish HDDs (rotational) and SSDs (not rotational).

hctl (String): The SCSI address: Host, channel, target and lun. For example: 1:0:0:0.

by_path (String): The alternative device name, e.g. /dev/disk/by-path/pci-0000:00

- **root_device_hints** A dictionary with the root device hints.

Raises ValueError, if some information is invalid.

Returns The first device to match all the hints or None.

ironic_lib.utils.**mkfs**(*fs*, *path*, *label=None*)

Format a file or block device

Parameters

- **fs** Filesystem type (examples include swap, ext3, ext4 btrfs, etc.)
- **path** Path to file or block device to format
- **label** Volume label to use

ironic_lib.utils.**mounted**(*source*, *dest=None*, *opts=None*, *fs_type=None*, *mount_attempts=1*, *umount_attempts=3*)

A context manager for a temporary mount.

Parameters

- **source** A device to mount.
- **dest** Mount destination. If not specified, a temporary directory will be created and removed afterwards. An existing destination is not removed.
- **opts** Mount options (`-o` argument).
- **fs_type** File system type (`-t` argument).
- **mount_attempts** A number of attempts to mount the device.
- **umount_attempts** A number of attempts to unmount the device.

Returns A generator yielding the destination.

ironic_lib.utils.**parse_device_tags**(*output*)

Parse tags from the lsblk/blkid output.

Parses format KEY=VALUE KEY2=VALUE2.

Returns a generator yielding dicts with information from each line.

ironic_lib.utils.**parse_root_device_hints**(*root_device*)

Parse the root_device property of a node.

Parses and validates the root_device property of a node. These are hints for how a nodes root device is created. The size hint should be a positive integer. The rotational hint should be a Boolean value.

Parameters **root_device** the root_device dictionary from the nodes property.

Returns a dictionary with the root device hints parsed or None if there are no hints.

Raises ValueError, if some information is invalid.

ironic_lib.utils.**try_execute**(**cmd*, ***kwargs*)
The same as execute but returns None on error.

Executes and logs results from a system command. See docs for oslo_concurrency.processutils.execute for usage.

Instead of raising an exception on failure, this method simply returns None in case of failure.

Parameters

- **cmd** positional arguments to pass to processutils.execute()
- **kwargs** keyword arguments to pass to processutils.execute()

Raises UnknownArgumentError on receiving unknown arguments

Returns tuple of (stdout, stderr) or None in some error cases

ironic_lib.utils.**unlink_without_raise**(*path*)

ironic_lib.utils.**wait_for_disk_to_become_available**(*device*)
Wait for a disk device to become available.

Waits for a disk device to become available for use by waiting until all process locks on the device have been released.

Timeout and iteration settings come from the configuration options used by the in-library disk_partitioner: `check_device_interval` and `check_device_max_retries`.

Params device The path to the device.

Raises IronicException If the disk fails to become available.

ironic_lib.version module

Module contents

- genindex
- search

PYTHON MODULE INDEX

INDEX

T

U

V

W

Z