Client

Creating a client

To communicate with the Docker daemon, you first need to instantiate a client. The easiest way to do that is by calling the function `from_env()`. It can also be configured manually by instantiating a `DockerClient` class.

`from_env()`

Return a client configured from environment variables.

The environment variables used are the same as those used by the Docker command-line client. They are:

`DOCKER_HOST`
The URL to the Docker host.
`DOCKER_TLS_VERIFY`
Verify the host against a CA certificate.
`DOCKER_CERT_PATH`
A path to a directory containing TLS certificates to use when connecting to the Docker host.

Parameters:

- **version** (*str*) – The version of the API to use. Set to `auto` to automatically detect the server's version. Default: `auto`
- **timeout** (*int*) – Default timeout for API calls, in seconds.
- **max_pool_size** (*int*) – The maximum number of connections to save in the pool.
- **environment** (*dict*) – The environment to read environment variables from. Default: the value of `os.environ`

- **credstore_env** (*dict*) – Override environment variables when calling the credential store process.
- **use_ssh_client** (*bool*) – If set to *True*, an ssh connection is made via shelling out to the ssh client. Ensure the ssh client is installed and configured on the host.

Example

```
>>> import docker
>>> client = docker.from_env()
```

Client reference

`class DockerClient`

A client for communicating with a Docker server.

Example

```
>>> import docker
>>> client =
docker.DockerClient(base_url='unix://var/run/docker.sock')
```

Parameters:

- **base_url** (*str*) – URL to the Docker server. For example, `unix:///var/run/docker.sock` or `tcp://127.0.0.1:1234`.
- **version** (*str*) – The version of the API to use. Set to `auto` to automatically detect the server's version. Default: `1.35`
- **timeout** (*int*) – Default timeout for API calls, in seconds.
- **tls** (bool or `TLSConfig`) – Enable TLS. Pass `True` to enable it with default

options, or pass a `TLSConfig` object to use custom configuration.
- **user_agent** (*str*) – Set a custom user agent for requests to the server.
- **credstore_env** (*dict*) – Override environment variables when calling the credential store process.
- **use_ssh_client** (*bool*) – If set to *True*, an ssh connection is made via shelling out to the ssh client. Ensure the ssh client is installed and configured on the host.
- **max_pool_size** (*int*) – The maximum number of connections to save in the pool.

`configs`

An object for managing configs on the server. See the configs documentation for full details.

`containers`

An object for managing containers on the server. See the containers documentation for full details.

`images`

An object for managing images on the server. See the images documentation for full details.

`networks`

An object for managing networks on the server. See the networks documentation for full details.

`nodes`

An object for managing nodes on the server. See the nodes documentation for full details.

`plugins`

An object for managing plugins on the server. See the plugins documentation for full details.

`secrets`

An object for managing secrets on the server. See the secrets documentation for full details.

`services`

An object for managing services on the server. See the services documentation for full details.

`swarm`

An object for managing a swarm on the server. See the swarm documentation for full details.

`volumes`

An object for managing volumes on the server. See the volumes documentation for full details.

`close()`

Closes all adapters and as such the session

`df()`

Get data usage information.

Returns:
A dictionary representing different resource categories and their respective data usage.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`events()`

Get real-time events from the server. Similar to the `docker events` command.

Parameters:

- **since** (*UTC datetime or int*) – Get events from this point
- **until** (*UTC datetime or int*) – Get events until this point
- **filters** (*dict*) – Filter the events by event time, container or image

- **decode** (*bool*) – If set to true, stream will be decoded into dicts on the fly. False by default.

Returns:
A `docker.types.daemon.CancellableStream` generator
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> for event in
client.events(decode=True)
...   print(event)
{u'from': u'image/with:tag',
 u'id': u'container-id',
 u'status': u'start',
 u'time': 1423339459}
...
```

or

```
>>> events = client.events()
>>> for event in events:
...   print(event)
>>> # and cancel from another thread
>>> events.close()
```

`info()`

Display system-wide information. Identical to the `docker info` command.

Returns:
The info as a dict
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`login()`

Authenticate with a registry. Similar to the `docker login` command.

Parameters:

- **username** (*str*) – The registry username
- **password** (*str*) – The plaintext password
- **email** (*str*) – The email for the registry account
- **registry** (*str*) – URL to the registry. E.g. `https://index.docker.io/v1/`
- **reauth** (*bool*) – Whether or not to refresh existing authentication on the Docker server.
- **dockercfg_path** (*str*) – Use a custom path for the Docker config file (default `$HOME/.docker/config.json` if present, otherwise `$HOME/.dockercfg`)

Returns:
The response from the login request
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`ping()`

Checks the server is responsive. An exception will be raised if it isn’t responding.

Returns:
(bool) The response from the server.
Raises:
docker.errors.APIError – If the server returns an error.

`version()`

Returns version information from the server. Similar to the `docker version` command.

Returns:

The server version information
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Configs

Manage configs on the server.

Methods available on `client.configs`:

`create`(***kwargs*)
Create a config

Parameters:

- **name** (*string*) – Name of the config
- **data** (*bytes*) – Config data to be stored
- **labels** (*dict*) – A mapping of labels to assign to the config
- **templating** (*dict*) – dictionary containing the name of the templating driver to be used expressed as { name: <templating_driver_name>}

Returns (dict): ID of the newly created config

`get`(*config_id*)
Get a config.

Parameters:
config_id (*str*) – Config ID.
Returns:
The config.
Return type:
(`Config`)
Raises:

- **docker.errors.NotFound** – If the config does not exist.
- **docker.errors.APIError** – If the server returns an error.

`list`(***kwargs*)
List configs. Similar to the `docker config ls` command.

Parameters:
filters (*dict*) – Server-side list filtering options.
Returns:
The configs.
Return type:
(list of `Config`)
Raises:

docker.errors.APIError – If the server returns an error.

Config objects

class **`Config`**

A config.

`id`

The ID of the object.

`name`

`attrs`

The raw representation of this object from the server.

`reload()`

Load this object from the server again and update `attrs` with the new data.

`remove()`

Remove this config.

Raises:

docker.errors.APIError – If config failed to remove.

Containers

Run and manage containers on the server.

Methods available on `client.containers`:

`run`(*`image, command=None, **kwargs`*)

Run a container. By default, it will wait for the container to finish and return its logs, similar to `docker run`.

If the `detach` argument is `True`, it will start the container and immediately return a **`Container`** object, similar to `docker run -d`.

Example

Run a container and get its output:

```
>>> import docker
>>> client = docker.from_env()
>>> client.containers.run('alpine', 'echo hello world')
b'hello world\n'
```

Run a container and detach:

```
>>> container = client.containers.run('bfirsh/reticulate-
splines',
                                      detach=True)
>>> container.logs()
'Reticulating spline 1...\nReticulating spline 2...\n'
```

Parameters:

- **image** (*str*) – The image to run.
- **command** (*str or list*) – The command to run in the container.
- **auto_remove** (*bool*) – enable auto-removal of the container on daemon side when the container's process exits.
- **blkio_weight_device** – Block IO weight (relative device weight) in the form of: `[{"Path": "device_path", "Weight": weight}]`.
- **blkio_weight** – Block IO weight (relative weight), accepts a weight value between 10 and 1000.
- **cap_add** (*list of str*) – Add kernel capabilities. For example, `["SYS_ADMIN", "MKNOD"]`.
- **cap_drop** (*list of str*) – Drop kernel capabilities.
- **cgroup_parent** (*str*) – Override the default parent cgroup.
- **cgroupns** (*str*) – Override the default cgroup namespace mode for the container. One of: - `private` the container runs in its own private cgroup namespace.
    - `host` use the host system's cgroup namespace.
- **cpu_count** (*int*) – Number of usable CPUs (Windows only).
- **cpu_percent** (*int*) – Usable percentage of the available CPUs (Windows only).
- **cpu_period** (*int*) – The length of a CPU period in microseconds.
- **cpu_quota** (*int*) – Microseconds of CPU time that the container can get in a CPU period.
- **cpu_rt_period** (*int*) – Limit CPU real-time period in microseconds.
- **cpu_rt_runtime** (*int*) – Limit CPU real-time runtime in microseconds.
- **cpu_shares** (*int*) – CPU shares (relative weight).
- **cpuset_cpus** (*str*) – CPUs in which to allow execution (`0-3`, `0,1`).

- **cpuset_mems** (*str*) – Memory nodes (MEMs) in which to allow execution (`0-3`, `0,1`). Only effective on NUMA systems.
- **detach** (*bool*) – Run container in the background and return a `Container` object.
- **device_cgroup_rules** (`list`) – A list of cgroup rules to apply to the container.
- **device_read_bps** – Limit read rate (bytes per second) from a device in the form of: *[{"Path": "device_path", "Rate": rate}]*
- **device_read_iops** – Limit read rate (IO per second) from a device.
- **device_write_bps** – Limit write rate (bytes per second) from a device.
- **device_write_iops** – Limit write rate (IO per second) from a device.
- **devices** (`list`) –
  Expose host devices to the container, as a list of strings in the form `<path_on_host>:<path_in_container>:<cgroup_permissions>`.
  For example, `/dev/sda:/dev/xvda:rwm` allows the container to have read-write access to the host's `/dev/sda` via a node named `/dev/xvda` inside the container.
- **device_requests** (`list`) – Expose host resources such as GPUs to the container, as a list of `docker.types.DeviceRequest` instances.
- **dns** (`list`) – Set custom DNS servers.
- **dns_opt** (`list`) – Additional options to be added to the container's `resolv.conf` file.
- **dns_search** (`list`) – DNS search domains.
- **domainname** (*str or list*) – Set custom DNS search domains.
- **entrypoint** (*str or list*) – The entrypoint for the container.
- **environment** (*dict or list*) – Environment variables to set inside the container, as a dictionary or a list of strings in the format `["SOMEVARIABLE=xxx"]`.
- **extra_hosts** (*dict*) – Additional hostnames to resolve inside the container, as a mapping of hostname to IP address.
- **group_add** (`list`) – List of additional group names and/or IDs that the container process will run as.
- **healthcheck** (*dict*) –
  Specify a test to perform to check that the container is healthy. The dict takes the following keys:
  - test (`list` or str): Test to perform to determine

container health. Possible values:

  - Empty list: Inherit healthcheck from parent image
  - `["NONE"]`: Disable healthcheck
  - `["CMD", args...]`: exec arguments directly.
  - `["CMD-SHELL", command]`: Run command in the system's default shell.

If a string is provided, it will be used as a `CMD-SHELL` command.

    - interval (int): The time to wait between checks in nanoseconds. It should be 0 or at least 1000000 (1 ms).
    - timeout (int): The time to wait before considering the check to have hung. It should be 0 or at least 1000000 (1 ms).
    - retries (int): The number of consecutive failures needed to
consider a container as unhealthy.
    - start_period (int): Start period for the container to
initialize before starting health-retries countdown in nanoseconds. It should be 0 or at least 1000000 (1 ms).

- **hostname** (*str*) – Optional hostname for the container.
- **init** (*bool*) – Run an init inside the container that forwards signals and reaps processes
- **init_path** (*str*) – Path to the docker-init binary
- **ipc_mode** (*str*) – Set the IPC mode for the container.
- **isolation** (*str*) – Isolation technology to use. Default: *None*.
- **kernel_memory** (*int or str*) – Kernel memory limit
- **labels** (*dict or list*) – A dictionary of name-value labels (e.g. `{"label1": "value1", "label2": "value2"}`) or a list of names of labels to set with empty values (e.g. `["label1", "label2"]`)
- **links** (*dict*) – Mapping of links using the `{'container': 'alias'}` format. The alias is optional. Containers declared in this dict will be linked to the new container using the provided alias. Default: `None`.
- **log_config** (*LogConfig*) – Logging configuration.
- **lxc_conf** (*dict*) – LXC config.
- **mac_address** (*str*) – MAC address to assign to the container.
- **mem_limit** (*int or str*) – Memory limit. Accepts float values (which represent the memory limit of the created container in bytes) or a string with a units identification char (`100000b`, `1000k`, `128m`, `1g`). If a string is specified without a units character, bytes are assumed as an intended unit.
- **mem_reservation** (*int or str*) – Memory soft limit.

- **mem_swappiness** (*int*) – Tune a container’s memory swappiness behavior. Accepts number between 0 and 100.
- **memswap_limit** (*str or int*) – Maximum amount of memory + swap a container is allowed to consume.
- **mounts** (`list`) – Specification for mounts to be added to the container. More powerful alternative to `volumes`. Each item in the list is expected to be a **`docker.types.Mount`** object.
- **name** (*str*) – The name for this container.
- **nano_cpus** (*int*) – CPU quota in units of 1e-9 CPUs.
- **network** (*str*) – Name of the network this container will be connected to at creation time. You can connect to additional networks using **`Network.connect()`**. Incompatible with `network_mode`.
- **network_disabled** (*bool*) – Disable networking.
- **network_mode** (*str*) –
  One of:
    - `bridge` Create a new network stack for the container on the bridge network.
    - `none` No networking for this container.
    - `container:<name|id>` Reuse another container’s network stack.
    - `host` Use the host network stack. This mode is incompatible with `ports`.

  Incompatible with `network`.
- **networking_config** (*Dict[str, EndpointConfig]*) –
  Dictionary of EndpointConfig objects for each container network. The key is the name of the network. Defaults to `None`.
  Used in conjuction with `network`.
  Incompatible with `network_mode`.
- **oom_kill_disable** (*bool*) – Whether to disable OOM killer.
- **oom_score_adj** (*int*) – An integer value containing the score given to the container in order to tune OOM killer preferences.
- **pid_mode** (*str*) – If set to `host`, use the host PID namespace inside the container.
- **pids_limit** (*int*) – Tune a container’s pids limit. Set `-1` for unlimited.
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`. Only used if the method needs to pull the requested image.
- **ports** (*dict*) –
  Ports to bind inside the container.

The keys of the dictionary are the ports to bind inside the container, either as an integer or a string in the form `port/protocol`, where the protocol is either `tcp`, `udp`, or `sctp`.
The values of the dictionary are the corresponding ports to open on the host, which can be either:
  - The port number, as an integer. For example, `{'2222/tcp': 3333}` will expose port 2222 inside the container as port 3333 on the host.
  - `None`, to assign a random host port. For example, `{'2222/tcp': None}`.
  - A tuple of `(address, port)` if you want to specify the host interface. For example, `{'1111/tcp': ('127.0.0.1', 1111)}`.
  - A list of integers, if you want to bind multiple host ports to a single container port. For example, `{'1111/tcp': [1234, 4567]}`.

Incompatible with `host` network mode.
- **privileged** (*bool*) – Give extended privileges to this container.
- **publish_all_ports** (*bool*) – Publish all ports to the host.
- **read_only** (*bool*) – Mount the container's root filesystem as read only.
- **remove** (*bool*) – Remove the container when it has finished running. Default: `False`.
- **restart_policy** (*dict*) –
  Restart the container when it exits. Configured as a dictionary with keys:
  - `Name` One of `on-failure`, or `always`.
  - `MaximumRetryCount` Number of times to restart the container on failure.

  For example: `{"Name": "on-failure", "MaximumRetryCount": 5}`
- **runtime** (*str*) – Runtime to use with this container.
- **security_opt** (`list`) – A list of string values to customize labels for MLS systems, such as SELinux.
- **shm_size** (*str or int*) – Size of /dev/shm (e.g. `1G`).
- **stdin_open** (*bool*) – Keep `STDIN` open even if not attached.
- **stdout** (*bool*) – Return logs from `STDOUT` when `detach=False`. Default: `True`.
- **stderr** (*bool*) – Return logs from `STDERR` when `detach=False`. Default: `False`.
- **stop_signal** (*str*) – The stop signal to use to stop the container (e.g. `SIGINT`).

- **storage_opt** (*dict*) – Storage driver options per container as a key-value mapping.
- **stream** (*bool*) – If true and `detach` is false, return a log generator instead of a string. Ignored if `detach` is true. Default: `False`.
- **sysctls** (*dict*) – Kernel parameters to set in the container.
- **tmpfs** (*dict*) –
  Temporary filesystems to mount, as a dictionary mapping a path inside the container to options for that path.
  For example:
  ```
  {
      '/mnt/vol2': '',
      '/mnt/vol1': 'size=3G,uid=1000'
  }
  ```
- **tty** (*bool*) – Allocate a pseudo-TTY.
- **ulimits** (`list`) – Ulimits to set inside the container, as a list of **`docker.types.Ulimit`** instances.
- **use_config_proxy** (*bool*) – If `True`, and if the docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being built.
- **user** (*str or int*) – Username or UID to run commands as inside the container.
- **userns_mode** (*str*) – Sets the user namespace mode for the container when user namespace remapping option is enabled. Supported values are: `host`
- **uts_mode** (*str*) – Sets the UTS namespace mode for the container. Supported values are: `host`
- **version** (*str*) – The version of the API to use. Set to `auto` to automatically detect the server's version. Default: `1.35`
- **volume_driver** (*str*) – The name of a volume driver/plugin.
- **volumes** (*dict or list*) –
  A dictionary to configure volumes mounted inside the container. The key is either the host path or a volume name, and the value is a dictionary with the keys:
  - `bind` The path to mount the volume inside the container
  - `mode` Either `rw` to mount the volume read/write, or `ro` to mount it read-only.

  For example:
  ```
  {'/home/user1/': {'bind': '/mnt/vol2', 'mode': 'rw'},
   '/var/www': {'bind': '/mnt/vol1', 'mode': 'ro'}}
  ```

Or a list of strings which each one of its elements specifies a mount volume.
For example:
`['/home/user1/:/mnt/vol2','/var/www:/mnt/vol1']`

- **volumes_from** (`list`) – List of container names or IDs to get volumes from.
- **working_dir** (*str*) – Path to the working directory.

Returns:
The container logs, either `STDOUT`, `STDERR`, or both, depending on the value of the `stdout` and `stderr` arguments.
`STDOUT` and `STDERR` may be read only if either `json-file` or `journald` logging driver used. Thus, if you are using none of these drivers, a `None` object is returned instead. See the Engine API documentation for full details.
If `detach` is `True`, a **`Container`** object is returned instead.
Raises:

- **docker.errors.ContainerError** – If the container exits with a non-zero exit code and `detach` is `False`.
- **docker.errors.ImageNotFound** – If the specified image does not exist.
- **docker.errors.APIError** – If the server returns an error.

`create`(*`image, command=None, **kwargs`*)
Create a container without starting it. Similar to `docker create`.

Takes the same arguments as **`run()`**, except for `stdout`, `stderr`, and `remove`.

Returns:
A **`Container`** object.
Raises:

- **docker.errors.ImageNotFound** – If the specified image does not exist.
- **docker.errors.APIError** – If the server returns an error.

`get`(*`id_or_name`*)
Get a container by name or ID.

Parameters:
container_id (*str*) – Container name or ID.
Returns:
A **`Container`** object.
Raises:

- **docker.errors.NotFound** – If the container does not exist.
- **docker.errors.APIError** – If the server returns an error.

`list(**kwargs)`

List containers. Similar to the `docker ps` command.

Parameters:

- **all** (*bool*) – Show all containers. Only running containers are shown by default
- **since** (*str*) – Show only containers created since Id or Name, include non-running ones
- **before** (*str*) – Show only container created before Id or Name, include non-running ones
- **limit** (*int*) – Show *limit* last created containers, include non-running ones
- **filters** (*dict*) –
  Filters to be processed on the image list. Available filters:
    - *exited* (int): Only containers with specified exit code
    - *status* (str): One of `restarting`, `running`,
`paused`, `exited`
    - *label* (str|list): format either `"key"`, `"key=value"`
or a list of such.
    - *id* (str): The id of the container.
    - *name* (str): The name of the container.
    - *ancestor* (str): Filter by container ancestor. Format of
`<image-name>[:tag]`, `<image-id>`, or `<image@digest>`.
    - *before* (str): Only containers created before a particular
container. Give the container name or id.
    - *since* (str): Only containers created after a particular
container. Give container name or id.

  A comprehensive list can be found in the documentation for docker ps.
- **sparse** (*bool*) – Do not inspect containers. Returns partial information, but guaranteed not to block.
  Use **`Container.reload()`** on resulting objects to retrieve all attributes. Default: `False`
- **ignore_removed** (*bool*) – Ignore failures due to missing containers when attempting to inspect containers from the original list. Set to `True` if race conditions are likely. Has no effect if `sparse=True`. Default: `False`

Returns:

(list of **`Container`**)

Raises:
docker.errors.APIError – If the server returns an error.

`prune(filters=None)`

Delete stopped containers

Parameters:
filters (*dict*) – Filters to process on the prune list.
Returns:
A dict containing a list of deleted container IDs and the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Container objects

class **Container**

Local representation of a container object. Detailed configuration may be accessed through the `attrs` attribute. Note that local attributes are cached; users may call `reload()` to query the Docker daemon for the current properties, causing `attrs` to be refreshed.

attrs

id

The ID of the object.

image

The image of the container.

labels

The labels of a container as dictionary.

name

The name of the container.

short_id

The ID of the object, truncated to 12 characters.

status

The status of the container. For example, `running`, or `exited`.

The raw representation of this object from the server.

`attach(**kwargs)`

Attach to this container.

`logs()` is a wrapper around this method, which you can use instead if you want to fetch/stream container output without first retrieving the entire backlog.

Parameters:

- **stdout** (*bool*) – Include stdout.
- **stderr** (*bool*) – Include stderr.
- **stream** (*bool*) – Return container output progressively as an iterator of strings, rather than a single string.
- **logs** (*bool*) – Include the container's previous output.

Returns:
By default, the container's output as a single string.
If `stream=True`, an iterator of output strings.
Raises:
docker.errors.APIError – If the server returns an error.

`attach_socket(**kwargs)`

Like `attach()`, but returns the underlying socket-like object for the HTTP request.

Parameters:

- **params** (*dict*) – Dictionary of request parameters (e.g. `stdout`, `stderr`, `stream`).
- **ws** (*bool*) – Use websockets instead of raw HTTP.

Raises:
docker.errors.APIError – If the server returns an error.

`commit(repository=None, tag=None, **kwargs)`

Commit a container to an image. Similar to the `docker commit` command.

Parameters:

- **repository** (*str*) – The repository to push the image to
- **tag** (*str*) – The tag to push
- **message** (*str*) – A commit message
- **author** (*str*) – The name of the author
- **pause** (*bool*) – Whether to pause the container before committing
- **changes** (*str*) – Dockerfile instructions to apply while committing
- **conf** (*dict*) –
  The configuration for the container. See the Engine API documentation for full details.

Raises:
docker.errors.APIError – If the server returns an error.

`diff()`

Inspect changes on a container's filesystem.

Returns:

(list) A list of dictionaries containing the attributes *Path* and *Kind*.

Raises:

docker.errors.APIError – If the server returns an error.

`exec_run(cmd, stdout=True, stderr=True, stdin=False, tty=False, privileged=False, user='', detach=False, stream=False, socket=False, environment=None, workdir=None, demux=False)`

Run a command inside this container. Similar to `docker exec`.

Parameters:

- **cmd** (*str or list*) – Command to be executed
- **stdout** (*bool*) – Attach to stdout. Default: `True`
- **stderr** (*bool*) – Attach to stderr. Default: `True`
- **stdin** (*bool*) – Attach to stdin. Default: `False`
- **tty** (*bool*) – Allocate a pseudo-TTY. Default: False
- **privileged** (*bool*) – Run as privileged.
- **user** (*str*) – User to execute command as. Default: root
- **detach** (*bool*) – If true, detach from the exec command. Default: False
- **stream** (*bool*) – Stream response data. Default: False
- **socket** (*bool*) – Return the connection socket to allow custom read/write operations. Default: False
- **environment** (*dict or list*) – A dictionary or a list of strings in the following format `["PASSWORD=xxx"]` or `{"PASSWORD": "xxx"}`.
- **workdir** (*str*) – Path to working directory for this exec session
- **demux** (*bool*) – Return stdout and stderr separately

Returns:

A tuple of (exit_code, output)

exit_code: (int):

Exit code for the executed command or `None` if either `stream` or `socket` is `True`.

output: (generator, bytes, or tuple):

If `stream=True`, a generator yielding response chunks. If `socket=True`, a socket object for the connection. If `demux=True`, a tuple of two bytes: stdout and stderr. A bytestring containing response data otherwise.

Return type:

(ExecResult)

Raises:

docker.errors.APIError – If the server returns an error.

`export(chunk_size=2097152)`

Export the contents of the container’s filesystem as a tar archive.

Parameters:

chunk_size (*int*) – The number of bytes returned by each iteration of the generator. If `None`, data will be streamed as it is received. Default: 2 MB

Returns:

The filesystem tar archive

Return type:

(str)

Raises:

docker.errors.APIError – If the server returns an error.

`get_archive`(*path, chunk_size=2097152, encode_stream=False*)

Retrieve a file or folder from the container in the form of a tar archive.

Parameters:

- **path** (*str*) – Path to the file or folder to retrieve
- **chunk_size** (*int*) – The number of bytes returned by each iteration of the generator. If `None`, data will be streamed as it is received. Default: 2 MB
- **encode_stream** (*bool*) – Determines if data should be encoded (gzip-compressed) during transmission. Default: False

Returns:

First element is a raw tar data stream. Second element is a dict containing `stat` information on the specified `path`.

Return type:

(tuple)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> f = open('./sh_bin.tar', 'wb')
>>> bits, stat = container.get_archive('/bin/sh')
>>> print(stat)
{'name': 'sh', 'size': 1075464, 'mode': 493,
 'mtime': '2018-10-01T15:37:48-07:00', 'linkTarget': ''}
>>> for chunk in bits:
...     f.write(chunk)
>>> f.close()
```

`kill`(*signal=None*)

Kill or send a signal to the container.

Parameters:

signal (*str or int*) – The signal to send. Defaults to `SIGKILL`

Raises:

docker.errors.APIError – If the server returns an error.

`logs(**kwargs)`

Get logs from this container. Similar to the `docker logs` command.

The `stream` parameter makes the `logs` function return a blocking generator you can iterate over to retrieve log output as it happens.

Parameters:

- **stdout** (*bool*) – Get `STDOUT`. Default `True`
- **stderr** (*bool*) – Get `STDERR`. Default `True`
- **stream** (*bool*) – Stream the response. Default `False`
- **timestamps** (*bool*) – Show timestamps. Default `False`
- **tail** (*str or int*) – Output specified number of lines at the end of logs. Either an integer of number of lines or the string `all`. Default `all`
- **since** (*datetime, int, or float*) – Show logs since a given datetime, integer epoch (in seconds) or float (in nanoseconds)
- **follow** (*bool*) – Follow log output. Default `False`
- **until** (*datetime, int, or float*) – Show logs that occurred before the given datetime, integer epoch (in seconds), or float (in nanoseconds)

Returns:

Logs from the container.

Return type:

(generator of bytes or bytes)

Raises:

docker.errors.APIError – If the server returns an error.

`pause()`

Pauses all processes within this container.

Raises:

docker.errors.APIError – If the server returns an error.

`put_archive(path, data)`

Insert a file or folder in this container using a tar archive as source.

Parameters:

- **path** (*str*) – Path inside the container where the file(s) will be extracted. Must exist.
- **data** (*bytes or stream*) – tar data to be extracted

Returns:

True if the call succeeds.

Return type:

(bool)

Raises:

APIError –

reload()

Load this object from the server again and update `attrs` with the new data.

remove(*kwargs*)**

Remove this container. Similar to the `docker rm` command.

Parameters:

- **v** (*bool*) – Remove the volumes associated with the container
- **link** (*bool*) – Remove the specified link and not the underlying container
- **force** (*bool*) – Force the removal of a running container (uses `SIGKILL`)

Raises:

docker.errors.APIError – If the server returns an error.

rename(*name*)

Rename this container. Similar to the `docker rename` command.

Parameters:

name (*str*) – New name for the container

Raises:

docker.errors.APIError – If the server returns an error.

resize(*height, width*)

Resize the tty session.

Parameters:

- **height** (*int*) – Height of tty session
- **width** (*int*) – Width of tty session

Raises:

docker.errors.APIError – If the server returns an error.

restart(*kwargs*)**

Restart this container. Similar to the `docker restart` command.

Parameters:

timeout (*int*) – Number of seconds to try to stop for before killing the container. Once killed it will then be restarted. Default is 10 seconds.

Raises:

docker.errors.APIError – If the server returns an error.

start(*kwargs*)**

Start this container. Similar to the `docker start` command, but doesn’t support attach options.

Raises:

docker.errors.APIError – If the server returns an error.

stats(*kwargs*)**

Stream statistics for this container. Similar to the `docker stats` command.

Parameters:

- **decode** (*bool*) – If set to true, stream will be decoded into dicts on the fly. Only applicable if `stream` is True. False by default.
- **stream** (*bool*) – If set to false, only the current stats will be returned instead of a stream. True by default.

Raises:

docker.errors.APIError – If the server returns an error.

`stop(**kwargs)`

Stops a container. Similar to the `docker stop` command.

Parameters:

timeout (*int*) – Timeout in seconds to wait for the container to stop before sending a `SIGKILL`. Default: 10

Raises:

docker.errors.APIError – If the server returns an error.

`top(**kwargs)`

Display the running processes of the container.

Parameters:

ps_args (*str*) – An optional arguments passed to ps (e.g. `aux`)

Returns:

The output of the top

Return type:

(str)

Raises:

docker.errors.APIError – If the server returns an error.

`unpause()`

Unpause all processes within the container.

Raises:

docker.errors.APIError – If the server returns an error.

`update(**kwargs)`

Update resource configuration of the containers.

Parameters:

- **blkio_weight** (*int*) – Block IO (relative weight), between 10 and 1000
- **cpu_period** (*int*) – Limit CPU CFS (Completely Fair Scheduler) period
- **cpu_quota** (*int*) – Limit CPU CFS (Completely Fair Scheduler) quota
- **cpu_shares** (*int*) – CPU shares (relative weight)
- **cpuset_cpus** (*str*) – CPUs in which to allow execution
- **cpuset_mems** (*str*) – MEMs in which to allow execution
- **mem_limit** (*int or str*) – Memory limit

- **mem_reservation** (*int or str*) – Memory soft limit
- **memswap_limit** (*int or str*) – Total memory (memory + swap), -1 to disable swap
- **kernel_memory** (*int or str*) – Kernel memory limit
- **restart_policy** (*dict*) – Restart policy dictionary

Returns:
Dictionary containing a `Warnings` key.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`wait(**kwargs)`
Block until the container stops, then return its exit code. Similar to the `docker wait` command.

Parameters:
- **timeout** (*int*) – Request timeout
- **condition** (*str*) – Wait until a container state reaches the given condition, either `not-running` (default), `next-exit`, or `removed`

Returns:
The API's response as a Python dictionary, including the container's exit code under the `StatusCode` attribute.
Return type:
(dict)
Raises:
- **requests.exceptions.ReadTimeout** – If the timeout is exceeded.
- **docker.errors.APIError** – If the server returns an error.

Images

Manage images on the server.

Methods available on `client.images`:

`build(**kwargs)`
Build an image and return it. Similar to the `docker build` command. Either `path` or `fileobj` must be set.

If you already have a tar file for the Docker build context (including a Dockerfile), pass a readable file-like object to `fileobj` and also pass `custom_context=True`. If the stream is also compressed, set `encoding` to the correct value (e.g `gzip`).

If you want to get the raw output of the build, use the **`build()`** method in the low-level API.

Parameters:

- **path** (*str*) – Path to the directory containing the Dockerfile
- **fileobj** – A file object to use as the Dockerfile. (Or a file-like object)
- **tag** (*str*) – A tag to add to the final image
- **quiet** (*bool*) – Whether to return the status
- **nocache** (*bool*) – Don't use the cache when set to `True`
- **rm** (*bool*) – Remove intermediate containers.
  The `docker build` command now defaults to `--rm=true`, but we have kept the old default of *False* to preserve backward compatibility
- **timeout** (*int*) – HTTP timeout
- **custom_context** (*bool*) – Optional if using `fileobj`
- **encoding** (*str*) – The encoding for a stream. Set to `gzip` for compressing
- **pull** (*bool*) – Downloads any updates to the FROM image in Dockerfiles
- **forcerm** (*bool*) – Always remove intermediate containers, even after unsuccessful builds
- **dockerfile** (*str*) – path within the build context to the Dockerfile
- **buildargs** (*dict*) – A dictionary of build arguments
- **container_limits** (*dict*) –
  A dictionary of limits applied to each container created by the build process. Valid keys:
  - memory (int): set memory limit for build
  - memswap (int): Total memory (memory + swap), -1 to disable

swap

  - cpushares (int): CPU shares (relative weight)
  - cpusetcpus (str): CPUs in which to allow execution, e.g.,

`"0-3"`, `"0,1"`

- **shmsize** (*int*) – Size of */dev/shm* in bytes. The size must be greater than 0. If omitted the system uses 64MB
- **labels** (*dict*) – A dictionary of labels to set on the image

- **cache_from** (*list*) – A list of images used for build cache resolution
- **target** (*str*) – Name of the build-stage to build in a multi-stage Dockerfile
- **network_mode** (*str*) – networking mode for the run commands during build
- **squash** (*bool*) – Squash the resulting images layers into a single layer.
- **extra_hosts** (*dict*) – Extra hosts to add to /etc/hosts in building containers, as a mapping of hostname to IP address.
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`.
- **isolation** (*str*) – Isolation technology used during build. Default: *None*.
- **use_config_proxy** (*bool*) – If `True`, and if the docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being built.

Returns:
The first item is the **`Image`** object for the
image that was built. The second item is a generator of the build logs as JSON-decoded objects.
Return type:
(tuple)
Raises:

- **docker.errors.BuildError** – If there is an error during the build.
- **docker.errors.APIError** – If the server returns any other error.
- **TypeError** – If neither `path` nor `fileobj` is specified.

get(*name*)

Gets an image.

Parameters:
name (*str*) – The name of the image.
Returns:
The image.
Return type:
(**`Image`**)
Raises:

- **docker.errors.ImageNotFound** – If the image does not exist.
- **docker.errors.APIError** – If the server returns an error.

`get_registry_data(name, auth_config=None)`

Gets the registry data for an image.

Parameters:

- **name** (*str*) – The name of the image.
- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.

Returns:
The data object.
Return type:
(`RegistryData`)
Raises:
docker.errors.APIError – If the server returns an error.

`list(**kwargs)`

List images on the server.

Parameters:

- **name** (*str*) – Only show images belonging to the repository `name`
- **all** (*bool*) – Show intermediate image layers. By default, these are filtered out.
- **filters** (*dict*) – Filters to be processed on the image list. Available filters: - `dangling` (bool) - *label* (str|list): format either `"key"`, `"key=value"` or a list of such.

Returns:
The images.
Return type:
(list of `Image`)
Raises:
docker.errors.APIError – If the server returns an error.

`load(data)`

Load an image that was previously saved using `save()` (or `docker save`). Similar to `docker load`.

Parameters:
data (*binary*) – Image data to be loaded.
Returns:
The images.
Return type:
(list of `Image`)

Raises:
docker.errors.APIError – If the server returns an error.

`prune(`*`filters=None`*`)`

Delete unused images

Parameters:
filters (*dict*) – Filters to process on the prune list. Available filters: - dangling (bool): When set to true (or 1), prune only unused and untagged images.
Returns:
A dict containing a list of deleted image IDs and the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`pull(`*`repository, tag=None, all_tags=False, **kwargs`*`)`

Pull an image of the given name and return it. Similar to the `docker pull` command. If `tag` is `None` or empty, it is set to `latest`. If `all_tags` is set, the `tag` parameter is ignored and all image tags will be pulled.

If you want to get the raw pull output, use the **`pull()`** method in the low-level API.

Parameters:

- **repository** (*str*) – The repository to pull
- **tag** (*str*) – The tag to pull
- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`
- **all_tags** (*bool*) – Pull all image tags

Returns:
The image that has been pulled.
If `all_tags` is True, the method will return a list of **`Image`** objects belonging to this repository.
Return type:
(**`Image`** or list)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> # Pull the image tagged `latest` in the busybox repo
>>> image = client.images.pull('busybox')

>>> # Pull all tags in the busybox repo
>>> images = client.images.pull('busybox', all_tags=True)
```

`push(repository, tag=None, **kwargs)`

Push an image or a repository to the registry. Similar to the `docker push` command.

Parameters:

- **repository** (*str*) – The repository to push to
- **tag** (*str*) – An optional tag to push
- **stream** (*bool*) – Stream the output as a blocking generator
- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.
- **decode** (*bool*) – Decode the JSON data from the server into dicts. Only applies with `stream=True`

Returns:

The output from the server.

Return type:

(generator or str)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> resp = client.api.push(
...     'yourname/app',
...     stream=True,
...     decode=True,
... )
... for line in resp:
...   print(line)
{'status': 'Pushing repository yourname/app (1 tags)'}
{'status': 'Pushing','progressDetail': {}, 'id': '511136ea3c5a'}
{'status': 'Image already pushed, skipping', 'progressDetail':{},
 'id': '511136ea3c5a'}
...
```

`remove(*args, **kwargs)`

Remove an image. Similar to the `docker rmi` command.

Parameters:

- **image** (*str*) – The image to remove
- **force** (*bool*) – Force removal of the image
- **noprune** (*bool*) – Do not delete untagged parents

`search(*args, **kwargs)`

Search for images on Docker Hub. Similar to the `docker search` command.

Parameters:

- **term** (*str*) – A term to search for.
- **limit** (*int*) – The maximum number of results to return.

Returns:

The response of the search.

Return type:

(list of dicts)

Raises:

docker.errors.APIError – If the server returns an error.

Image objects

class **`Image`**

An image on the server.

`attrs`

The raw representation of this object from the server.

`id`

The ID of the object.

`labels`

The labels of an image as dictionary.

`short_id`

The ID of the image truncated to 12 characters, plus the `sha256:` prefix.

`tags`

The image’s tags.

`history()`

Show the history of an image.

Returns:

The history of the image.

Return type:

(list)
Raises:
docker.errors.APIError – If the server returns an error.

`reload()`

Load this object from the server again and update `attrs` with the new data.

`save(chunk_size=2097152, named=False)`

Get a tarball of an image. Similar to the `docker save` command.

Parameters:

- **chunk_size** (*int*) – The generator will return up to that much data per iteration, but may return less. If `None`, data will be streamed as it is received. Default: 2 MB
- **named** (*str or bool*) – If `False` (default), the tarball will not retain repository and tag information for this image. If set to `True`, the first tag in the **`tags`** list will be used to identify the image. Alternatively, any element of the **`tags`** list can be used as an argument to use that specific tag as the saved identifier.

Returns:
A stream of raw archive data.
Return type:
(generator)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> image = cli.images.get("busybox:latest")
>>> f = open('/tmp/busybox-latest.tar', 'wb')
>>> for chunk in image.save():
>>>   f.write(chunk)
>>> f.close()
```

`tag(repository, tag=None, **kwargs)`

Tag this image into a repository. Similar to the `docker tag` command.

Parameters:

- **repository** (*str*) – The repository to set for the tag
- **tag** (*str*) – The tag name
- **force** (*bool*) – Force

Raises:
docker.errors.APIError – If the server returns an error.
Returns:
`True` if successful
Return type:

(bool)

RegistryData objects

class **`RegistryData`**

Image metadata stored on the registry, including available platforms.

`attrs`

The raw representation of this object from the server.

`id`

The ID of the object.

`short_id`

The ID of the image truncated to 12 characters, plus the `sha256:` prefix.

`has_platform`(*`platform`*)

Check whether the given platform identifier is available for this digest.

Parameters:

platform (*str or dict*) – A string using the `os[/arch[/variant]]` format, or a platform dictionary.

Returns:

`True` if the platform is recognized as available, `False` otherwise.

Return type:

(bool)

Raises:

docker.errors.InvalidArgument – If the platform argument is not a valid descriptor.

`pull`(*`platform=None`*)

Pull the image digest.

Parameters:

- **platform** (*str*) – The platform to pull the image for.
- **Default** – `None`

Returns:

A reference to the pulled image.

Return type:

(`Image`)

`reload``()`

Load this object from the server again and update `attrs` with the new data.

Networks

Create and manage networks on the server. For more information about networks, see the Engine documentation.

Methods available on `client.networks`:

`create(name, *args, **kwargs)`
Create a network. Similar to the `docker network create`.

Parameters:

- **name** (*str*) – Name of the network
- **driver** (*str*) – Name of the driver used to create the network
- **options** (*dict*) – Driver options as a key-value dictionary
- **ipam** (*IPAMConfig*) – Optional custom IP scheme for the network.
- **check_duplicate** (*bool*) – Request daemon to check for networks with same name. Default: `None`.
- **internal** (*bool*) – Restrict external access to the network. Default `False`.
- **labels** (*dict*) – Map of labels to set on the network. Default `None`.
- **enable_ipv6** (*bool*) – Enable IPv6 on the network. Default `False`.
- **attachable** (*bool*) – If enabled, and the network is in the global scope, non-service containers on worker nodes will be able to connect to the network.
- **scope** (*str*) – Specify the network's scope (`local`, `global` or `swarm`)
- **ingress** (*bool*) – If set, create an ingress network which provides the routing-mesh in swarm mode.

Returns:
The network that was created.

Return type:
(`Network`)

Raises:
docker.errors.APIError – If the server returns an error.

Example

A network using the bridge driver:

```
>>> client.networks.create("network1", driver="bridge")
```

You can also create more advanced networks with custom IPAM configurations. For example, setting the subnet to `192.168.52.0/24` and gateway address to `192.168.52.254`.

```
>>> ipam_pool = docker.types.IPAMPool(
    subnet='192.168.52.0/24',
    gateway='192.168.52.254'
)
>>> ipam_config = docker.types.IPAMConfig(
    pool_configs=[ipam_pool]
)
>>> client.networks.create(
    "network1",
    driver="bridge",
    ipam=ipam_config
)
```

get(*network_id*, **args*, ***kwargs*)

Get a network by its ID.

Parameters:

- **network_id** (*str*) – The ID of the network.
- **verbose** (*bool*) – Retrieve the service details across the cluster in swarm mode.
- **scope** (*str*) – Filter the network by scope (`swarm`, `global` or `local`).

Returns:
(`Network`) The network.

Raises:

- **docker.errors.NotFound** – If the network does not exist.
- **docker.errors.APIError** – If the server returns an error.

list(**args*, ***kwargs*)

List networks. Similar to the `docker network ls` command.

Parameters:

- **names** (`list`) – List of names to filter by.
- **ids** (`list`) – List of ids to filter by.
- **filters** (*dict*) –
  Filters to be processed on the network list. Available filters: - `driver=[<driver-name>]` Matches a network's driver. - *label* (str|list): format either `"key"`, `"key=value"`

or a list of such.
    - `type=["custom"|"builtin"]` Filters networks by type.
- **greedy** (*bool*) – Fetch more details for each network individually. You might want this to get the containers attached to them.

Returns:
(list of `Network`) The networks on the server.
Raises:
docker.errors.APIError – If the server returns an error.

`prune(`*`filters=None`*`)`

Delete unused networks

Parameters:
filters (*dict*) – Filters to process on the prune list.
Returns:
A dict containing a list of deleted network names and
the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Network objects

`class` **`Network`**

A Docker network.

`id`

The ID of the object.

`short_id`

The ID of the object, truncated to 12 characters.

`name`

The name of the network.

`containers`

The containers that are connected to the network, as a list of `Container` objects.

`attrs`

The raw representation of this object from the server.

`connect``(`*`container, *args, **kwargs`*`)`

Connect a container to this network.

Parameters:

- **container** (*str*) – Container to connect to this network, as either an ID, name, or `Container` object.
- **aliases** (`list`) – A list of aliases for this endpoint. Names in that list can be used within the network to reach the container. Defaults to `None`.
- **links** (`list`) – A list of links for this endpoint. Containers declared in this list will be linkedto this container. Defaults to `None`.
- **ipv4_address** (*str*) – The IP address of this container on the network, using the IPv4 protocol. Defaults to `None`.
- **ipv6_address** (*str*) – The IP address of this container on the network, using the IPv6 protocol. Defaults to `None`.
- **link_local_ips** (`list`) – A list of link-local (IPv4/IPv6) addresses.
- **driver_opt** (*dict*) – A dictionary of options to provide to the network driver. Defaults to `None`.

Raises:

docker.errors.APIError – If the server returns an error.

`disconnect``(`*`container, *args, **kwargs`*`)`

Disconnect a container from this network.

Parameters:

- **container** (*str*) – Container to disconnect from this network, as either an ID, name, or `Container` object.
- **force** (*bool*) – Force the container to disconnect from a network. Default: `False`

Raises:

docker.errors.APIError – If the server returns an error.

`reload``()`

Load this object from the server again and update `attrs` with the new data.

`remove``()`

Remove this network.

Raises:

docker.errors.APIError – If the server returns an error.

Nodes

Get and list nodes in a swarm. Before you can use these methods, you first need to join or initialize a swarm.

Methods available on `client.nodes`:

get(*id_or_name*)

Get a node.

Parameters:
node_id (*string*) – ID of the node to be inspected.
Returns:
A `Node` object.
Raises:
docker.errors.APIError – If the server returns an error.

list(***kwargs*)

List swarm nodes.

Parameters:
filters (*dict*) – Filters to process on the nodes list. Valid filters: `id`, `name`, `membership` and `role`. Default: `None`
Returns:
A list of `Node` objects.
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> client.nodes.list(filters={'role': 'manager'})
```

Node objects

class **Node**

A node in a swarm.

id
The ID of the object.
short_id
The ID of the object, truncated to 12 characters.
attrs
The raw representation of this object from the server.
version
The version number of the service. If this is not the same as the server, the `update()` function will not work and you will need to call `reload()` before calling it again.
reload()
Load this object from the server again and update `attrs` with the new data.

`update`(*node_spec*)

Update the node’s configuration.

Parameters:
node_spec (*dict*) – Configuration settings to update. Any values not provided will be removed. Default: `None`
Returns:
True if the request went through.
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> node_spec = {'Availability': 'active',
                 'Name': 'node-name',
                 'Role': 'manager',
                 'Labels': {'foo': 'bar'}
                }
>>> node.update(node_spec)
```

Plugins

Manage plugins on the server.

Methods available on `client.plugins`:

`get`(*name*)

Gets a plugin.

Parameters:
name (*str*) – The name of the plugin.
Returns:
The plugin.
Return type:
(`Plugin`)
Raises:

- **docker.errors.NotFound** –
- **exist.** –
- **docker.errors.APIError** – If the server returns an error.

`install`(*remote_name*, *local_name=None*)

Pull and install a plugin.

Parameters:

- **remote_name** (*string*) – Remote reference for the plugin to install. The `:latest` tag is optional, and is the default if omitted.
- **local_name** (*string*) – Local name for the pulled plugin. The `:latest` tag is optional, and is the default if omitted. Optional.

Returns:
The installed plugin

Return type:
(`Plugin`)

Raises:
docker.errors.APIError – If the server returns an error.

`list`()

List plugins installed on the server.

Returns:
The plugins.

Return type:
(list of `Plugin`)

Raises:
docker.errors.APIError – If the server returns an error.

Plugin objects

class **`Plugin`**

A plugin on the server.

`id`

The ID of the object.

`short_id`

The ID of the object, truncated to 12 characters.

`name`

The plugin’s name.

`enabled`

Whether the plugin is enabled.

`settings`

A dictionary representing the plugin’s configuration.

`attrs`

The raw representation of this object from the server.

`configure(options)`

Update the plugin's settings.

Parameters:

options (*dict*) – A key-value mapping of options.

Raises:

docker.errors.APIError – If the server returns an error.

`disable(force=False)`

Disable the plugin.

Parameters:

force (*bool*) – Force disable. Default: False

Raises:

docker.errors.APIError – If the server returns an error.

`enable(timeout=0)`

Enable the plugin.

Parameters:

timeout (*int*) – Timeout in seconds. Default: 0

Raises:

docker.errors.APIError – If the server returns an error.

`reload()`

Load this object from the server again and update `attrs` with the new data.

`push()`

Push the plugin to a remote registry.

Returns:

A dict iterator streaming the status of the upload.

Raises:

docker.errors.APIError – If the server returns an error.

`remove(force=False)`

Remove the plugin from the server.

Parameters:

force (*bool*) – Remove even if the plugin is enabled. Default: False

Raises:

docker.errors.APIError – If the server returns an error.

`upgrade(remote=None)`

Upgrade the plugin.

Parameters:

remote (*string*) – Remote reference to upgrade to. The `:latest` tag is optional and is the default if omitted. Default: this plugin's name.
Returns:
A generator streaming the decoded API logs

Secrets

Manage secrets on the server.

Methods available on `client.secrets`:

`create(**kwargs)`
Create a secret

Parameters:

- **name** (*string*) – Name of the secret
- **data** (*bytes*) – Secret data to be stored
- **labels** (*dict*) – A mapping of labels to assign to the secret
- **driver** (*DriverConfig*) – A custom driver configuration. If unspecified, the default `internal` driver will be used

Returns (dict): ID of the newly created secret

`get(secret_id)`
Get a secret.

Parameters:
secret_id (*str*) – Secret ID.
Returns:
The secret.
Return type:
(`Secret`)
Raises:

- **docker.errors.NotFound** – If the secret does not exist.
- **docker.errors.APIError** – If the server returns an error.

`list(**kwargs)`
List secrets. Similar to the `docker secret ls` command.

Parameters:
filters (*dict*) – Server-side list filtering options.
Returns:
The secrets.
Return type:
(list of `Secret`)
Raises:
docker.errors.APIError – If the server returns an error.

Secret objects

class **`Secret`**

A secret.

`id`
The ID of the object.
`name`
`attrs`
The raw representation of this object from the server.
`reload()`
Load this object from the server again and update `attrs` with the new data.
`remove()`
Remove this secret.

Raises:
docker.errors.APIError – If secret failed to remove.

Services

Manage services on a swarm. For more information about services, see the Engine documentation.

Before you can use any of these methods, you first need to join or initialize a swarm.

Methods available on `client.services`:

`create`(*image, command=None, **kwargs*)
Create a service. Similar to the `docker service create` command.

Parameters:

- **image** (*str*) – The image name to use for the containers.
- **command** (*list of str or str*) – Command to run.
- **args** (*list of str*) – Arguments to the command.
- **constraints** (*list of str*) – `Placement` constraints.
- **preferences** (*list of tuple*) – `Placement` preferences.
- **maxreplicas** (*int*) – `Placement` maxreplicas or (int) representing maximum number of replicas per node.
- **platforms** (*list of tuple*) – A list of platform constraints expressed as `(arch, os)` tuples.
- **container_labels** (*dict*) – Labels to apply to the container.
- **endpoint_spec** (*EndpointSpec*) – Properties that can be configured to access and load balance a service. Default: `None`.
- **env** (*list of str*) – Environment variables, in the form `KEY=val`.
- **hostname** (*string*) – Hostname to set on the container.
- **init** (*boolean*) – Run an init inside the container that forwards signals and reaps processes
- **isolation** (*string*) – Isolation technology used by the service's containers. Only used for Windows containers.
- **labels** (*dict*) – Labels to apply to the service.
- **log_driver** (*str*) – Log driver to use for containers.
- **log_driver_options** (*dict*) – Log driver options.
- **mode** (*ServiceMode*) – Scheduling mode for the service. Default:`None`
- **mounts** (*list of str*) – Mounts for the containers, in the form `source:target:options`, where options is either `ro` or `rw`.
- **name** (*str*) – Name to give to the service.
- **networks** (`list`) – List of network names or IDs or `NetworkAttachmentConfig` to attach the service to. Default: `None`.
- **resources** (*Resources*) – Resource limits and reservations.
- **restart_policy** (*RestartPolicy*) – Restart policy for containers.
- **secrets** (list of `SecretReference`) – List of secrets accessible to containers for this service.
- **stop_grace_period** (*int*) – Amount of time to wait for containers to terminate before forcefully killing them.
- **update_config** (*UpdateConfig*) – Specification for the update strategy of the service. Default: `None`
- **rollback_config** (*RollbackConfig*) – Specification for the rollback strategy of the service. Default: `None`
- **user** (*str*) – User to run commands as.
- **workdir** (*str*) – Working directory for commands to run.

- **tty** (*boolean*) – Whether a pseudo-TTY should be allocated.
- **groups** (`list`) – A list of additional groups that the container process will run as.
- **open_stdin** (*boolean*) – Open `stdin`
- **read_only** (*boolean*) – Mount the container's root filesystem as read only.
- **stop_signal** (*string*) – Set signal to stop the service's containers
- **healthcheck** (*Healthcheck*) – Healthcheck configuration for this service.
- **hosts** (`dict`) – A set of host to IP mappings to add to the container's *hosts* file.
- **dns_config** (*DNSConfig*) – Specification for DNS related configurations in resolver configuration file.
- **configs** (`list`) – List of `ConfigReference` that will be exposed to the service.
- **privileges** (*Privileges*) – Security options for the service's containers.
- **cap_add** (`list`) – A list of kernel capabilities to add to the default set for the container.
- **cap_drop** (`list`) – A list of kernel capabilities to drop from the default set for the container.
- **sysctls** (`dict`) – A dict of sysctl values to add to the container

Returns:
The created service.
Return type:
`Service`
Raises:
docker.errors.APIError – If the server returns an error.

get(*service_id, insert_defaults=None*)

Get a service.

Parameters:

- **service_id** (*str*) – The ID of the service.
- **insert_defaults** (*boolean*) – If true, default values will be merged into the output.

Returns:
The service.
Return type:
`Service`
Raises:

- **docker.errors.NotFound** – If the service does not exist.

- **docker.errors.APIError** – If the server returns an error.
- **docker.errors.InvalidVersion** – If one of the arguments is not supported with the current API version.

`list(**kwargs)`

List services.

Parameters:

- **filters** (*dict*) – Filters to process on the nodes list. Valid filters: `id`, `name` , `label` and `mode`. Default: `None`.
- **status** (*bool*) – Include the service task count of running and desired tasks. Default: `None`.

Returns:
The services.

Return type:
list of `Service`

Raises:
docker.errors.APIError – If the server returns an error.

Service objects

class **`Service`**

A service.

`id`
The ID of the object.

`short_id`
The ID of the object, truncated to 12 characters.

`name`
The service’s name.

`version`
The version number of the service. If this is not the same as the server, the `update()` function will not work and you will need to call `reload()` before calling it again.

`attrs`
The raw representation of this object from the server.

`force_update()`
Force update the service even if no changes require it.

Returns:
`True` if successful.

Return type:

bool

`logs``(`*`**kwargs`*`)`

Get log stream for the service. Note: This method works only for services with the `json-file` or `journald` logging drivers.

Parameters:

- **details** (*bool*) – Show extra details provided to logs. Default: `False`
- **follow** (*bool*) – Keep connection open to read logs as they are sent by the Engine. Default: `False`
- **stdout** (*bool*) – Return logs from `stdout`. Default: `False`
- **stderr** (*bool*) – Return logs from `stderr`. Default: `False`
- **since** (*int*) – UNIX timestamp for the logs staring point. Default: 0
- **timestamps** (*bool*) – Add timestamps to every log line.
- **tail** (*string or int*) – Number of log lines to be returned, counting from the current end of the logs. Specify an integer or `'all'` to output all log lines. Default: `all`

Returns:

Logs for the service.

Return type:

generator

`reload``()`

Load this object from the server again and update `attrs` with the new data.

`remove``()`

Stop and remove the service.

Raises:

docker.errors.APIError – If the server returns an error.

`scale``(`*`replicas`*`)`

Scale service container.

Parameters:

replicas (*int*) – The number of containers that should be running.

Returns:

`True` if successful.

Return type:

bool

`tasks``(`*`filters=None`*`)`

List the tasks in this service.

Parameters:

filters (*dict*) – A map of filters to process on the tasks list. Valid filters: `id`, `name`, `node`, `label`, and `desired-state`.

Returns:
List of task dictionaries.
Return type:
`list`
Raises:
docker.errors.APIError – If the server returns an error.

`update(**kwargs)`

Update a service’s configuration. Similar to the `docker service update` command.

Takes the same parameters as `create()`.

Raises:
docker.errors.APIError – If the server returns an error.

Swarm

Manage [Docker Engine’s swarm mode](https://docs.docker.com/engine/swarm/).

To use any swarm methods, you first need to make the Engine part of a swarm. This can be done by either initializing a new swarm with `init()`, or joining an existing swarm with `join()`.

These methods are available on `client.swarm`:

`get_unlock_key()`

Get the unlock key for this Swarm manager.

Returns:
A `dict` containing an `UnlockKey` member

`init()`

Initialize a new swarm on this Engine.

Parameters:

- **advertise_addr** (*str*) –
  Externally reachable address advertised to other nodes. This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port

number, like `eth0:4567`. If the port number is omitted, the port number from the listen address is used.
If not specified, it will be automatically detected when possible.

- **listen_addr** (*str*) – Listen address used for inter-manager communication, as well as determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP). This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the default swarm listening port is used. Default: `0.0.0.0:2377`
- **force_new_cluster** (*bool*) – Force creating a new Swarm, even if already part of one. Default: False
- **default_addr_pool** (*list of str*) – Default Address Pool specifies default subnet pools for global scope networks. Each pool should be specified as a CIDR block, like '10.0.0.0/8'. Default: None
- **subnet_size** (*int*) – SubnetSize specifies the subnet size of the networks created from the default subnet pool. Default: None
- **data_path_addr** (*string*) – Address or interface to use for data path traffic. For example, 192.168.1.1, or an interface, like eth0.
- **data_path_port** (*int*) – Port number to use for data path traffic. Acceptable port range is 1024 to 49151. If set to `None` or 0, the default port 4789 will be used. Default: None
- **task_history_retention_limit** (*int*) – Maximum number of tasks history stored.
- **snapshot_interval** (*int*) – Number of logs entries between snapshot.
- **keep_old_snapshots** (*int*) – Number of snapshots to keep beyond the current snapshot.
- **log_entries_for_slow_followers** (*int*) – Number of log entries to keep around to sync up slow followers after a snapshot is created.
- **heartbeat_tick** (*int*) – Amount of ticks (in seconds) between each heartbeat.
- **election_tick** (*int*) – Amount of ticks (in seconds) needed without a leader to trigger a new election.
- **dispatcher_heartbeat_period** (*int*) – The delay for an agent to send a heartbeat to the dispatcher.
- **node_cert_expiry** (*int*) – Automatic expiry for nodes certificates.

- **external_ca** (*dict*) – Configuration for forwarding signing requests to an external certificate authority. Use `docker.types.SwarmExternalCA`.
- **name** (*string*) – Swarm's name
- **labels** (*dict*) – User-defined key/value metadata.
- **signing_ca_cert** (*str*) – The desired signing CA certificate for all swarm node TLS leaf certificates, in PEM format.
- **signing_ca_key** (*str*) – The desired signing CA key for all swarm node TLS leaf certificates, in PEM format.
- **ca_force_rotate** (*int*) – An integer whose purpose is to force swarm to generate a new signing CA certificate and key, if none have been specified.
- **autolock_managers** (*boolean*) – If set, generate a key and use it to lock data stored on the managers.
- **log_driver** (*DriverConfig*) – The default log driver to use for tasks created in the orchestrator.

Returns:
The ID of the created node.
Return type:
(str)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> client.swarm.init(
    advertise_addr='eth0', listen_addr='0.0.0.0:5000',
    force_new_cluster=False, default_addr_pool=['10.20.0.0/16],
    subnet_size=24, snapshot_interval=5000,
    log_entries_for_slow_followers=1200
)
```

`join()`

Make this Engine join a swarm that has already been created.

Parameters:
- **remote_addrs** (`list`) – Addresses of one or more manager nodes already participating in the Swarm to join.
- **join_token** (*string*) – Secret token for joining this Swarm.
- **listen_addr** (*string*) – Listen address used for inter-manager communication if the node gets promoted to manager, as well as

determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP). Default: `'0.0.0.0:2377`

- **advertise_addr** (*string*) – Externally reachable address advertised to other nodes. This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the port number from the listen address is used. If AdvertiseAddr is not specified, it will be automatically detected when possible. Default: `None`
- **data_path_addr** (*string*) – Address or interface to use for data path traffic. For example, 192.168.1.1, or an interface, like eth0.

Returns:
`True` if the request went through.
Raises:
docker.errors.APIError – If the server returns an error.

`leave()`

Leave a swarm.

Parameters:
force (*bool*) – Leave the swarm even if this node is a manager.
Default: `False`
Returns:
`True` if the request went through.
Raises:
docker.errors.APIError – If the server returns an error.

`unlock()`

Unlock a locked swarm.

Parameters:
key (*string*) – The unlock key as provided by `get_unlock_key()`
Raises:

- **docker.errors.InvalidArgument** – If the key argument is in an incompatible format
- **docker.errors.APIError** – If the server returns an error.

Returns:
True if the request was successful.

Example

```
>>> key = client.api.get_unlock_key()
>>> client.unlock_swarm(key)
```

`update()`

Update the swarm's configuration.

It takes the same arguments as **`init()`**, except `advertise_addr`, `listen_addr`, and `force_new_cluster`. In addition, it takes these arguments:

Parameters:

- **rotate_worker_token** (*bool*) – Rotate the worker join token. Default: `False`.
- **rotate_manager_token** (*bool*) – Rotate the manager join token. Default: `False`.
- **rotate_manager_unlock_key** (*bool*) – Rotate the manager unlock key. Default: `False`.

Raises:

docker.errors.APIError – If the server returns an error.

`reload()`

Inspect the swarm on the server and store the response in **`attrs`**.

Raises:

docker.errors.APIError – If the server returns an error.

`version`

The version number of the swarm. If this is not the same as the server, the **`update()`** function will not work and you will need to call **`reload()`** before calling it again.

`attrs`

The raw representation of this object from the server.

Volumes

Manage volumes on the server.

Methods available on `client.volumes`:

`create(`*`name=None, **kwargs`*`)`

Create a volume.

Parameters:

- **name** (*str*) – Name of the volume. If not specified, the engine generates a name.
- **driver** (*str*) – Name of the driver used to create the volume

- **driver_opts** (*dict*) – Driver options as a key-value dictionary
- **labels** (*dict*) – Labels to set on the volume

Returns:
The volume created.
Return type:
(`Volume`)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> volume = client.volumes.create(name='foobar', driver='local',
        driver_opts={'foo': 'bar', 'baz': 'false'},
        labels={"key": "value"})
```

`get(`*`volume_id`*`)`
Get a volume.

Parameters:
volume_id (*str*) – Volume name.
Returns:
The volume.
Return type:
(`Volume`)
Raises:

- **docker.errors.NotFound** – If the volume does not exist.
- **docker.errors.APIError** – If the server returns an error.

`list(`*`**kwargs`*`)`
List volumes. Similar to the `docker volume ls` command.

Parameters:
filters (*dict*) – Server-side list filtering options.
Returns:
The volumes.
Return type:
(list of `Volume`)
Raises:
docker.errors.APIError – If the server returns an error.

`prune(`*`filters=None`*`)`
Delete unused volumes

Parameters:

filters (*dict*) – Filters to process on the prune list.
Returns:
A dict containing a list of deleted volume names and
the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Volume objects

class **Volume**

A volume.

id

The ID of the object.

short_id

The ID of the object, truncated to 12 characters.

name

The name of the volume.

attrs

The raw representation of this object from the server.

reload()

Load this object from the server again and update `attrs` with the new data.

remove(*force=False*)

Remove this volume.

Parameters:
force (*bool*) – Force removal of volumes that were already removed out of band by the volume driver plugin.
Raises:
docker.errors.APIError – If volume failed to remove.

Low-level API

The main object-orientated API is built on top of **`APIClient`**. Each method on **`APIClient`** maps one-to-one with a REST API endpoint, and returns the response that the API responds with.

It's possible to use **`APIClient`** directly. Some basic things (e.g. running a container) consist of several API calls and are complex to do with the low-level API, but it's useful if you need extra flexibility and power.

class **APIClient**(*base_url=None*, *version=None*, *timeout=60*, *tls=False*, *user_agent='docker-sdk-python/7.1.0'*, *num_pools=None*, *credstore_env=None*, *use_ssh_client=False*, *max_pool_size=10*)

A low-level client for the Docker Engine API.

Example

```
>>> import docker
>>> client =
docker.APIClient(base_url='unix://var/run/docker.sock')
>>> client.version()
{u'ApiVersion': u'1.33',
 u'Arch': u'amd64',
 u'BuildTime': u'2017-11-19T18:46:37.000000000+00:00',
 u'GitCommit': u'f4ffd2511c',
 u'GoVersion': u'go1.9.2',
 u'KernelVersion': u'4.14.3-1-ARCH',
 u'MinAPIVersion': u'1.12',
 u'Os': u'linux',
 u'Version': u'17.10.0-ce'}
```

Parameters:

- **base_url** (*str*) – URL to the Docker server. For example, `unix:///var/run/docker.sock` or `tcp://127.0.0.1:1234`.
- **version** (*str*) – The version of the API to use. Set to `auto` to automatically detect the server's version. Default: `1.35`
- **timeout** (*int*) – Default timeout for API calls, in seconds.
- **tls** (bool or **`TLSConfig`**) – Enable TLS. Pass `True` to enable it with default options, or pass a **`TLSConfig`** object to use custom configuration.
- **user_agent** (*str*) – Set a custom user agent for requests to the server.

- **credstore_env** (*dict*) – Override environment variables when calling the credential store process.
- **use_ssh_client** (*bool*) – If set to *True*, an ssh connection is made via shelling out to the ssh client. Ensure the ssh client is installed and configured on the host.
- **max_pool_size** (*int*) – The maximum number of connections to save in the pool.

Configs

`configs`(*`filters=None`*)

List configs

Parameters:

- **filters** (*list. Available*) – A map of filters to process on the configs
- **filters** – `names`

Returns (list): A list of configs

`create_config`(*`name, data, labels=None, templating=None`*)

Create a config

Parameters:

- **name** (*string*) – Name of the config
- **data** (*bytes*) – Config data to be stored
- **labels** (*dict*) – A mapping of labels to assign to the config
- **templating** (*dict*) – dictionary containing the name of the templating driver to be used expressed as { name: <templating_driver_name>}

Returns (dict): ID of the newly created config

`inspect_config`(*`id`*)

Retrieve config metadata

Parameters:

id (*string*) – Full ID of the config to inspect

Returns (dict): A dictionary of metadata

Raises:

docker.errors.NotFound – if no config with that ID exists

`remove_config`(*`id`*)

Remove a config

Parameters:
id (*string*) – Full ID of the config to remove

Returns (boolean): True if successful

Raises:
docker.errors.NotFound – if no config with that ID exists

Containers

attach(*container, stdout=True, stderr=True, stream=False, logs=False, demux=False*)
Attach to a container.

The `.logs()` function is a wrapper around this method, which you can use instead if you want to fetch/stream container output without first retrieving the entire backlog.

Parameters:

- **container** (*str*) – The container to attach to.
- **stdout** (*bool*) – Include stdout.
- **stderr** (*bool*) – Include stderr.
- **stream** (*bool*) – Return container output progressively as an iterator of strings, rather than a single string.
- **logs** (*bool*) – Include the container's previous output.
- **demux** (*bool*) – Keep stdout and stderr separate.

Returns:
By default, the container's output as a single string (two if `demux=True`: one for stdout and one for stderr).
If `stream=True`, an iterator of output strings. If `demux=True`, two iterators are returned: one for stdout and one for stderr.
Raises:
docker.errors.APIError – If the server returns an error.
attach_socket(*container, params=None, ws=False*)
Like `attach`, but returns the underlying socket-like object for the HTTP request.

Parameters:

- **container** (*str*) – The container to attach to.
- **params** (*dict*) – Dictionary of request parameters (e.g. `stdout`, `stderr`, `stream`). For `detachKeys`, ~/.docker/config.json is used by default.
- **ws** (*bool*) – Use websockets instead of raw HTTP.

Raises:

docker.errors.APIError – If the server returns an error.

`commit``(`*`container, repository=None, tag=None, message=None, author=None, pause=True, changes=None, conf=None`*`)`

Commit a container to an image. Similar to the `docker commit` command.

Parameters:

- **container** (*str*) – The image hash of the container
- **repository** (*str*) – The repository to push the image to
- **tag** (*str*) – The tag to push
- **message** (*str*) – A commit message
- **author** (*str*) – The name of the author
- **pause** (*bool*) – Whether to pause the container before committing
- **changes** (*str*) – Dockerfile instructions to apply while committing
- **conf** (*dict*) – The configuration for the container. See the Engine API documentation for full details.

Raises:

docker.errors.APIError – If the server returns an error.

`containers``(`*`quiet=False, all=False, trunc=False, latest=False, since=None, before=None, limit=-1, size=False, filters=None`*`)`

List containers. Similar to the `docker ps` command.

Parameters:

- **quiet** (*bool*) – Only display numeric Ids
- **all** (*bool*) – Show all containers. Only running containers are shown by default
- **trunc** (*bool*) – Truncate output
- **latest** (*bool*) – Show only the latest created container, include non-running ones.
- **since** (*str*) – Show only containers created since Id or Name, include non-running ones
- **before** (*str*) – Show only container created before Id or Name, include non-running ones
- **limit** (*int*) – Show *limit* last created containers, include non-running ones
- **size** (*bool*) – Display sizes
- **filters** (*dict*) –
  Filters to be processed on the image list. Available filters:
    - *exited* (int): Only containers with specified exit code
    - *status* (str): One of `restarting`, `running`,
`paused`, `exited`
    - *label* (str|list): format either `"key"`, `"key=value"`
or a list of such.
    - *id* (str): The id of the container.

- *name* (str): The name of the container.
- *ancestor* (str): Filter by container ancestor. Format of `<image-name>[:tag]`, `<image-id>`, or `<image@digest>`.
- *before* (str): Only containers created before a particular container. Give the container name or id.
- *since* (str): Only containers created after a particular container. Give container name or id.

A comprehensive list can be found in the documentation for docker ps.

Returns:
A list of dicts, one per container
Raises:
docker.errors.APIError – If the server returns an error.

`create_container`(*`image, command=None, hostname=None, user=None, detach=False, stdin_open=False, tty=False, ports=None, environment=None, volumes=None, network_disabled=False, name=None, entrypoint=None, working_dir=None, domainname=None, host_config=None, mac_address=None, labels=None, stop_signal=None, networking_config=None, healthcheck=None, stop_timeout=None, runtime=None, use_config_proxy=True, platform=None`*)

Creates a container. Parameters are similar to those for the `docker run` command except it doesn't support the attach options (`-a`).

The arguments that are passed directly to this function are host-independent configuration options. Host-specific configuration is passed with the *host_config* argument. You'll normally want to use this method in combination with the **`create_host_config()`** method to generate `host_config`.

Port bindings

Port binding is done in two parts: first, provide a list of ports to open inside the container with the `ports` parameter, then declare bindings with the `host_config` parameter. For example:

```
container_id = client.api.create_container(
    'busybox', 'ls', ports=[1111, 2222],
    host_config=client.api.create_host_config(port_bindings={
        1111: 4567,
        2222: None
    })
)
```

You can limit the host address on which the port will be exposed like such:

```
client.api.create_host_config(
    port_bindings={1111: ('127.0.0.1', 4567)}
)
```

Or without host port assignment:

```
client.api.create_host_config(port_bindings={1111: ('127.0.0.1',)})
```

If you wish to use UDP instead of TCP (default), you need to declare ports as such in both the config and host config:

```
container_id = client.api.create_container(
    'busybox', 'ls', ports=[(1111, 'udp'), 2222],
    host_config=client.api.create_host_config(port_bindings={
        '1111/udp': 4567, 2222: None
    })
)
```

To bind multiple host ports to a single container port, use the following syntax:

```
client.api.create_host_config(port_bindings={
    1111: [1234, 4567]
})
```

You can also bind multiple IPs to a single container port:

```
client.api.create_host_config(port_bindings={
    1111: [
        ('192.168.0.100', 1234),
        ('192.168.0.101', 1234)
    ]
})
```

Using volumes

Volume declaration is done in two parts. Provide a list of paths to use as mountpoints inside the container with the `volumes` parameter, and declare mappings from paths on the host in the `host_config` section.

```
container_id = client.api.create_container(
    'busybox', 'ls', volumes=['/mnt/vol1', '/mnt/vol2'],
    host_config=client.api.create_host_config(binds={
        '/home/user1/': {
            'bind': '/mnt/vol2',
            'mode': 'rw',
        },
        '/var/www': {
            'bind': '/mnt/vol1',
            'mode': 'ro',
        },
        '/autofs/user1': {
            'bind': '/mnt/vol3',
            'mode': 'rw',
            'propagation': 'shared'
        }
    })
)
```

You can alternatively specify binds as a list. This code is equivalent to the example above:

```
container_id = client.api.create_container(
    'busybox', 'ls', volumes=['/mnt/vol1', '/mnt/vol2',
'/mnt/vol3'],
    host_config=client.api.create_host_config(binds=[
        '/home/user1/:/mnt/vol2',
        '/var/www:/mnt/vol1:ro',
        '/autofs/user1:/mnt/vol3:rw,shared',
    ])
)
```

Networking

You can specify networks to connect the container to by using the `networking_config` parameter. At the time of creation, you can only connect a container to a single networking, but you can create more connections by using **`connect_container_to_network()`**.

For example:

```
networking_config = client.api.create_networking_config({
    'network1': client.api.create_endpoint_config(
        ipv4_address='172.28.0.124',
        aliases=['foo', 'bar'],
        links=['container2']
    )
})

ctnr = client.api.create_container(
    img, command, networking_config=networking_config
)
```

Parameters:

- **image** (*str*) – The image to run
- **command** (*str or list*) – The command to be run in the container
- **hostname** (*str*) – Optional hostname for the container
- **user** (*str or int*) – Username or UID
- **detach** (*bool*) – Detached mode: run container in the background and return container ID
- **stdin_open** (*bool*) – Keep STDIN open even if not attached
- **tty** (*bool*) – Allocate a pseudo-TTY
- **ports** (*list of ints*) – A list of port numbers
- **environment** (*dict or list*) – A dictionary or a list of strings in the following format `["PASSWORD=xxx"]` or `{"PASSWORD": "xxx"}`.
- **volumes** (*str or list*) – List of paths inside the container to use as volumes.
- **network_disabled** (*bool*) – Disable networking
- **name** (*str*) – A name for the container
- **entrypoint** (*str or list*) – An entrypoint
- **working_dir** (*str*) – Path to the working directory
- **domainname** (*str*) – The domain name to use for the container
- **host_config** (*dict*) – A dictionary created with **`create_host_config()`**.
- **mac_address** (*str*) – The Mac Address to assign the container
- **labels** (*dict or list*) – A dictionary of name-value labels (e.g. `{"label1": "value1", "label2": "value2"}`) or a list of names of labels to set with empty values (e.g. `["label1", "label2"]`)
- **stop_signal** (*str*) – The stop signal to use to stop the container (e.g. `SIGINT`).
- **stop_timeout** (*int*) – Timeout to stop the container, in seconds. Default: 10
- **networking_config** (*dict*) – A networking configuration generated by **`create_networking_config()`**.
- **runtime** (*str*) – Runtime to use with this container.
- **healthcheck** (*dict*) – Specify a test to perform to check that the container is healthy.
- **use_config_proxy** (*bool*) – If `True`, and if the docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being created.
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`.

Returns:

A dictionary with an image ‘Id’ key and a ‘Warnings’ key.

Raises:

- **docker.errors.ImageNotFound** – If the specified image does not exist.
- **docker.errors.APIError** – If the server returns an error.

`create_container_config`(**args*, ***kwargs*)

`create_container_from_config`(*config*, *name=None*, *platform=None*)

`create_endpoint_config`(**args*, ***kwargs*)

Create an endpoint config dictionary to be used with **`create_networking_config()`**.

Parameters:

- **aliases** (`list`) – A list of aliases for this endpoint. Names in that list can be used within the network to reach the container. Defaults to `None`.
- **links** (*dict*) – Mapping of links for this endpoint using the `{'container': 'alias'}` format. The alias is optional. Containers declared in this dict will be linked to this container using the provided alias. Defaults to `None`.
- **ipv4_address** (*str*) – The IP address of this container on the network, using the IPv4 protocol. Defaults to `None`.
- **ipv6_address** (*str*) – The IP address of this container on the network, using the IPv6 protocol. Defaults to `None`.
- **link_local_ips** (`list`) – A list of link-local (IPv4/IPv6) addresses.
- **driver_opt** (*dict*) – A dictionary of options to provide to the network driver. Defaults to `None`.

Returns:

(dict) An endpoint config.

Example

```
>>> endpoint_config = client.api.create_endpoint_config(
    aliases=['web', 'app'],
    links={'app_db': 'db', 'another': None},
    ipv4_address='132.65.0.123'
)
```

`create_host_config`(**args*, ***kwargs*)

Create a dictionary for the `host_config` argument to **`create_container()`**.

Parameters:

- **auto_remove** (*bool*) – enable auto-removal of the container on daemon side when the container's process exits.
- **binds** (*dict*) – Volumes to bind. See **`create_container()`** for more information.
- **blkio_weight_device** – Block IO weight (relative device weight) in the form of: `[{"Path": "device_path", "Weight": weight}]`.
- **blkio_weight** – Block IO weight (relative weight), accepts a weight value between 10 and 1000.

- **cap_add** (*list of str*) – Add kernel capabilities. For example, `["SYS_ADMIN", "MKNOD"]`.
- **cap_drop** (*list of str*) – Drop kernel capabilities.
- **cpu_period** (*int*) – The length of a CPU period in microseconds.
- **cpu_quota** (*int*) – Microseconds of CPU time that the container can get in a CPU period.
- **cpu_shares** (*int*) – CPU shares (relative weight).
- **cpuset_cpus** (*str*) – CPUs in which to allow execution (`0-3`, `0,1`).
- **cpuset_mems** (*str*) – Memory nodes (MEMs) in which to allow execution (`0-3`, `0,1`). Only effective on NUMA systems.
- **device_cgroup_rules** (`list`) – A list of cgroup rules to apply to the container.
- **device_read_bps** – Limit read rate (bytes per second) from a device in the form of: *[{"Path": "device_path", "Rate": rate}]*
- **device_read_iops** – Limit read rate (IO per second) from a device.
- **device_write_bps** – Limit write rate (bytes per second) from a device.
- **device_write_iops** – Limit write rate (IO per second) from a device.
- **devices** (`list`) – Expose host devices to the container, as a list of strings in the form `<path_on_host>:<path_in_container>:<cgroup_permissions>`. For example, `/dev/sda:/dev/xvda:rwm` allows the container to have read-write access to the host's `/dev/sda` via a node named `/dev/xvda` inside the container.
- **device_requests** (`list`) – Expose host resources such as GPUs to the container, as a list of **`docker.types.DeviceRequest`** instances.
- **dns** (`list`) – Set custom DNS servers.
- **dns_opt** (`list`) – Additional options to be added to the container's `resolv.conf` file
- **dns_search** (`list`) – DNS search domains.
- **extra_hosts** (*dict*) – Additional hostnames to resolve inside the container, as a mapping of hostname to IP address.
- **group_add** (`list`) – List of additional group names and/or IDs that the container process will run as.
- **init** (*bool*) – Run an init inside the container that forwards signals and reaps processes
- **ipc_mode** (*str*) – Set the IPC mode for the container.
- **isolation** (*str*) – Isolation technology to use. Default: `None`.
- **links** (*dict*) – Mapping of links using the `{'container': 'alias'}` format. The alias is optional. Containers declared in this dict will be linked to the new container using the provided alias. Default: `None`.
- **log_config** (*LogConfig*) – Logging configuration
- **lxc_conf** (*dict*) – LXC config.

- **mem_limit** (*float or str*) – Memory limit. Accepts float values (which represent the memory limit of the created container in bytes) or a string with a units identification char (`100000b`, `1000k`, `128m`, `1g`). If a string is specified without a units character, bytes are assumed as an
- **mem_reservation** (*float or str*) – Memory soft limit.
- **mem_swappiness** (*int*) – Tune a container's memory swappiness behavior. Accepts number between 0 and 100.
- **memswap_limit** (*str or int*) – Maximum amount of memory + swap a container is allowed to consume.
- **mounts** (`list`) – Specification for mounts to be added to the container. More powerful alternative to `binds`. Each item in the list is expected to be a **`docker.types.Mount`** object.
- **network_mode** (*str*) –
  One of:
  - `bridge` Create a new network stack for the container on the bridge network.
  - `none` No networking for this container.
  - `container:<name|id>` Reuse another container's network stack.
  - `host` Use the host network stack. This mode is incompatible with `port_bindings`.
- **oom_kill_disable** (*bool*) – Whether to disable OOM killer.
- **oom_score_adj** (*int*) – An integer value containing the score given to the container in order to tune OOM killer preferences.
- **pid_mode** (*str*) – If set to `host`, use the host PID namespace inside the container.
- **pids_limit** (*int*) – Tune a container's pids limit. Set `-1` for unlimited.
- **port_bindings** (*dict*) – See **`create_container()`** for more information. Imcompatible with `host` in `network_mode`.
- **privileged** (*bool*) – Give extended privileges to this container.
- **publish_all_ports** (*bool*) – Publish all ports to the host.
- **read_only** (*bool*) – Mount the container's root filesystem as read only.
- **restart_policy** (*dict*) –
  Restart the container when it exits. Configured as a dictionary with keys:
  - `Name` One of `on-failure`, or `always`.
  - `MaximumRetryCount` Number of times to restart the container on failure.
- **security_opt** (`list`) – A list of string values to customize labels for MLS systems, such as SELinux.
- **shm_size** (*str or int*) – Size of /dev/shm (e.g. `1G`).
- **storage_opt** (*dict*) – Storage driver options per container as a key-value mapping.

- **sysctls** (*dict*) – Kernel parameters to set in the container.
- **tmpfs** (*dict*) –
  Temporary filesystems to mount, as a dictionary mapping a path inside the container to options for that path.
  For example:
  ```
  {
      '/mnt/vol2': '',
      '/mnt/vol1': 'size=3G,uid=1000'
  }
  ```
- **ulimits** (`list`) – Ulimits to set inside the container, as a list of **`docker.types.Ulimit`** instances.
- **userns_mode** (*str*) – Sets the user namespace mode for the container when user namespace remapping option is enabled. Supported values are: `host`
- **uts_mode** (*str*) – Sets the UTS namespace mode for the container. Supported values are: `host`
- **volumes_from** (`list`) – List of container names or IDs to get volumes from.
- **runtime** (*str*) – Runtime to use with this container.

Returns:
(dict) A dictionary which can be passed to the `host_config` argument to **`create_container()`**.

Example

```
>>> client.api.create_host_config(
...     privileged=True,
...     cap_drop=['MKNOD'],
...     volumes_from=['nostalgic_newton'],
... )
{'CapDrop': ['MKNOD'], 'LxcConf': None, 'Privileged': True,
'VolumesFrom': ['nostalgic_newton'], 'PublishAllPorts': False}
```

`create_networking_config``(*args, **kwargs)`

Create a networking config dictionary to be used as the `networking_config` parameter in **`create_container()`**.

Parameters:
endpoints_config (*dict*) – A dictionary mapping network names to endpoint configurations generated by **`create_endpoint_config()`**.
Returns:
(dict) A networking config.

Example

```
>>> client.api.create_network('network1')
>>> networking_config = client.api.create_networking_config({
    'network1': client.api.create_endpoint_config()
})
>>> container = client.api.create_container(
    img, command, networking_config=networking_config
)
```

`diff(container)`

Inspect changes on a container's filesystem.

Parameters:

container (*str*) – The container to diff

Returns:

(list) A list of dictionaries containing the attributes *Path* and *Kind*.

Raises:

docker.errors.APIError – If the server returns an error.

`export(container, chunk_size=2097152)`

Export the contents of a filesystem as a tar archive.

Parameters:

- **container** (*str*) – The container to export
- **chunk_size** (*int*) – The number of bytes returned by each iteration of the generator. If `None`, data will be streamed as it is received. Default: 2 MB

Returns:

The archived filesystem data stream

Return type:

(generator)

Raises:

docker.errors.APIError – If the server returns an error.

`get_archive(container, path, chunk_size=2097152, encode_stream=False)`

Retrieve a file or folder from a container in the form of a tar archive.

Parameters:

- **container** (*str*) – The container where the file is located
- **path** (*str*) – Path to the file or folder to retrieve
- **chunk_size** (*int*) – The number of bytes returned by each iteration of the generator. If `None`, data will be streamed as it is received. Default: 2 MB
- **encode_stream** (*bool*) – Determines if data should be encoded (gzip-compressed) during transmission. Default: False

Returns:

First element is a raw tar data stream. Second element is a dict containing `stat` information on the specified `path`.

Return type:

(tuple)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> c = docker.APIClient()
>>> f = open('./sh_bin.tar', 'wb')
>>> bits, stat = c.api.get_archive(container, '/bin/sh')
>>> print(stat)
{'name': 'sh', 'size': 1075464, 'mode': 493,
 'mtime': '2018-10-01T15:37:48-07:00', 'linkTarget': ''}
>>> for chunk in bits:
...    f.write(chunk)
>>> f.close()
```

`inspect_container`(*`container`*)

Identical to the *docker inspect* command, but only for containers.

Parameters:

container (*str*) – The container to inspect

Returns:

Similar to the output of *docker inspect*, but as a single dict

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

`kill`(*`container, signal=None`*)

Kill a container or send a signal to a container.

Parameters:

- **container** (*str*) – The container to kill
- **signal** (*str or int*) – The signal to send. Defaults to `SIGKILL`

Raises:

docker.errors.APIError – If the server returns an error.

`logs`(*`container, stdout=True, stderr=True, stream=False, timestamps=False, tail='all', since=None, follow=None, until=None`*)

Get logs from a container. Similar to the `docker logs` command.

The `stream` parameter makes the `logs` function return a blocking generator you can iterate over to retrieve log output as it happens.

Parameters:

- **container** (*str*) – The container to get logs from
- **stdout** (*bool*) – Get `STDOUT`. Default `True`
- **stderr** (*bool*) – Get `STDERR`. Default `True`
- **stream** (*bool*) – Stream the response. Default `False`
- **timestamps** (*bool*) – Show timestamps. Default `False`
- **tail** (*str or int*) – Output specified number of lines at the end of logs. Either an integer of number of lines or the string `all`. Default `all`
- **since** (*datetime, int, or float*) – Show logs since a given datetime, integer epoch (in seconds) or float (in fractional seconds)
- **follow** (*bool*) – Follow log output. Default `False`
- **until** (*datetime, int, or float*) – Show logs that occurred before the given datetime, integer epoch (in seconds), or float (in fractional seconds)

Returns:

(generator of bytes or bytes)

Raises:

docker.errors.APIError – If the server returns an error.

`pause(`*`container`*`)`

Pauses all processes within a container.

Parameters:

container (*str*) – The container to pause

Raises:

docker.errors.APIError – If the server returns an error.

`port(`*`container, private_port`*`)`

Lookup the public-facing port that is NAT-ed to `private_port`. Identical to the `docker port` command.

Parameters:

- **container** (*str*) – The container to look up
- **private_port** (*int*) – The private port to inspect

Returns:

The mapping for the host ports

Return type:

(list of dict)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
$ docker run -d -p 80:80 ubuntu:14.04 /bin/sleep 30
7174d6347063a83f412fad6124c99cffd25ffe1a0807eb4b7f9cec76ac8cb43b

>>> client.api.port('7174d6347063', 80)
[{'HostIp': '0.0.0.0', 'HostPort': '80'}]
```

`prune_containers`(*`filters=None`*)

Delete stopped containers

Parameters:
filters (*dict*) – Filters to process on the prune list.
Returns:
A dict containing a list of deleted container IDs and
the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`put_archive`(*`container, path, data`*)

Insert a file or folder in an existing container using a tar archive as source.

Parameters:

- **container** (*str*) – The container where the file(s) will be extracted
- **path** (*str*) – Path inside the container where the file(s) will be extracted. Must exist.
- **data** (*bytes or stream*) – tar data to be extracted

Returns:
True if the call succeeds.
Return type:
(bool)
Raises:
docker.errors.APIError – If the server returns an error.

`remove_container`(*`container, v=False, link=False, force=False`*)

Remove a container. Similar to the `docker rm` command.

Parameters:

- **container** (*str*) – The container to remove
- **v** (*bool*) – Remove the volumes associated with the container
- **link** (*bool*) – Remove the specified link and not the underlying container
- **force** (*bool*) – Force the removal of a running container (uses `SIGKILL`)

Raises:
docker.errors.APIError – If the server returns an error.

`rename`(*`container, name`*)

Rename a container. Similar to the `docker rename` command.

Parameters:

- **container** (*str*) – ID of the container to rename
- **name** (*str*) – New name for the container

Raises:

docker.errors.APIError – If the server returns an error.

`resize`(*`container, height, width`*)

Resize the tty session.

Parameters:

- **container** (*str or dict*) – The container to resize
- **height** (*int*) – Height of tty session
- **width** (*int*) – Width of tty session

Raises:

docker.errors.APIError – If the server returns an error.

`restart`(*`container, timeout=10`*)

Restart a container. Similar to the `docker restart` command.

Parameters:

- **container** (*str or dict*) – The container to restart. If a dict, the `Id` key is used.
- **timeout** (*int*) – Number of seconds to try to stop for before killing the container. Once killed it will then be restarted. Default is 10 seconds.

Raises:

docker.errors.APIError – If the server returns an error.

`start`(*`container, *args, **kwargs`*)

Start a container. Similar to the `docker start` command, but doesn't support attach options.

Deprecation warning: Passing configuration options in `start` is no longer supported. Users are expected to provide host config options in the `host_config` parameter of **`create_container()`**.

Parameters:

container (*str*) – The container to start

Raises:

- **docker.errors.APIError** – If the server returns an error.
- **docker.errors.DeprecatedMethod** – If any argument besides `container` are provided.

Example

```
>>> container = client.api.create_container(
...     image='busybox:latest',
...     command='/bin/sleep 30')
>>> client.api.start(container=container.get('Id'))
```

`stats`(*container*, *decode=None*, *stream=True*, *one_shot=None*)

Stream statistics for a specific container. Similar to the `docker stats` command.

Parameters:

- **container** (*str*) – The container to stream statistics from
- **decode** (*bool*) – If set to true, stream will be decoded into dicts on the fly. Only applicable if `stream` is True. False by default.
- **stream** (*bool*) – If set to false, only the current stats will be returned instead of a stream. True by default.
- **one_shot** (*bool*) – If set to true, Only get a single stat instead of waiting for 2 cycles. Must be used with stream=false. False by default.

Raises:

docker.errors.APIError – If the server returns an error.

`stop`(*container*, *timeout=None*)

Stops a container. Similar to the `docker stop` command.

Parameters:

- **container** (*str*) – The container to stop
- **timeout** (*int*) – Timeout in seconds to wait for the container to stop before sending a `SIGKILL`. If None, then the StopTimeout value of the container will be used. Default: None

Raises:

docker.errors.APIError – If the server returns an error.

`top`(*container*, *ps_args=None*)

Display the running processes of a container.

Parameters:

- **container** (*str*) – The container to inspect
- **ps_args** (*str*) – An optional arguments passed to ps (e.g. `aux`)

Returns:

The output of the top

Return type:

(str)

Raises:

docker.errors.APIError – If the server returns an error.

`unpause`(*container*)

Unpause all processes within a container.

Parameters:
container (*str*) – The container to unpause
update_container(*container, blkio_weight=None, cpu_period=None, cpu_quota=None, cpu_shares=None, cpuset_cpus=None, cpuset_mems=None, mem_limit=None, mem_reservation=None, memswap_limit=None, kernel_memory=None, restart_policy=None*)
Update resource configs of one or more containers.

Parameters:

- **container** (*str*) – The container to inspect
- **blkio_weight** (*int*) – Block IO (relative weight), between 10 and 1000
- **cpu_period** (*int*) – Limit CPU CFS (Completely Fair Scheduler) period
- **cpu_quota** (*int*) – Limit CPU CFS (Completely Fair Scheduler) quota
- **cpu_shares** (*int*) – CPU shares (relative weight)
- **cpuset_cpus** (*str*) – CPUs in which to allow execution
- **cpuset_mems** (*str*) – MEMs in which to allow execution
- **mem_limit** (*float or str*) – Memory limit
- **mem_reservation** (*float or str*) – Memory soft limit
- **memswap_limit** (*int or str*) – Total memory (memory + swap), -1 to disable swap
- **kernel_memory** (*int or str*) – Kernel memory limit
- **restart_policy** (*dict*) – Restart policy dictionary

Returns:
Dictionary containing a `Warnings` key.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.
wait(*container, timeout=None, condition=None*)
Block until a container stops, then return its exit code. Similar to
the `docker wait` command.

Parameters:

- **container** (*str or dict*) – The container to wait on. If a dict, the `Id` key is used.
- **timeout** (*int*) – Request timeout
- **condition** (*str*) – Wait until a container state reaches the given condition, either `not-running` (default), `next-exit`, or `removed`

Returns:
The API’s response as a Python dictionary, including
the container’s exit code under the `StatusCode` attribute.
Return type:

(dict)

Raises:

- **requests.exceptions.ReadTimeout** – If the timeout is exceeded.
- **docker.errors.APIError** – If the server returns an error.

Images

`get_image`(*image, chunk_size=2097152*)

Get a tarball of an image. Similar to the `docker save` command.

Parameters:

- **image** (*str*) – Image name to get
- **chunk_size** (*int*) – The number of bytes returned by each iteration of the generator. If `None`, data will be streamed as it is received. Default: 2 MB

Returns:

A stream of raw archive data.

Return type:

(generator)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> image = client.api.get_image("busybox:latest")
>>> f = open('/tmp/busybox-latest.tar', 'wb')
>>> for chunk in image:
>>>   f.write(chunk)
>>> f.close()
```

`history`(*image*)

Show the history of an image.

Parameters:

image (*str*) – The image to show history for

Returns:

The history of the image

Return type:

(list)

Raises:

docker.errors.APIError – If the server returns an error.

`images`(*name=None, quiet=False, all=False, filters=None*)

List images. Similar to the `docker images` command.

Parameters:

- **name** (*str*) – Only show images belonging to the repository `name`
- **quiet** (*bool*) – Only return numeric IDs as a list.
- **all** (*bool*) – Show intermediate image layers. By default, these are filtered out.
- **filters** (*dict*) –
  Filters to be processed on the image list. Available filters: - `dangling` (bool) - *label* (str|list): format either `"key"`, `"key=value"`
  or a list of such.

Returns:
A list if `quiet=True`, otherwise a dict.

Return type:
(dict or list)

Raises:
docker.errors.APIError – If the server returns an error.

`import_image`(*`src=None, repository=None, tag=None, image=None, changes=None, stream_src=False`*)

Import an image. Similar to the `docker import` command.

If `src` is a string or unicode string, it will first be treated as a path to a tarball on the local system. If there is an error reading from that file, `src` will be treated as a URL instead to fetch the image from. You can also pass an open file handle as `src`, in which case the data will be read from that file.

If `src` is unset but `image` is set, the `image` parameter will be taken as the name of an existing image to import from.

Parameters:

- **src** (*str or file*) – Path to tarfile, URL, or file-like object
- **repository** (*str*) – The repository to create
- **tag** (*str*) – The tag to apply
- **image** (*str*) – Use another image like the `FROM` Dockerfile parameter

`import_image_from_data`(*`data, repository=None, tag=None, changes=None`*)

Like **`import_image()`**, but allows importing in-memory bytes data.

Parameters:

- **data** (*bytes collection*) – Bytes collection containing valid tar data
- **repository** (*str*) – The repository to create
- **tag** (*str*) – The tag to apply

`import_image_from_file`(*`filename, repository=None, tag=None, changes=None`*)

Like **`import_image()`**, but only supports importing from a tar file on disk.

Parameters:

- **filename** (*str*) – Full path to a tar file.
- **repository** (*str*) – The repository to create
- **tag** (*str*) – The tag to apply

Raises:

IOError – File does not exist.

`import_image_from_image`(*image, repository=None, tag=None, changes=None*)

Like **`import_image()`**, but only supports importing from another image, like the `FROM` Dockerfile parameter.

Parameters:

- **image** (*str*) – Image name to import from
- **repository** (*str*) – The repository to create
- **tag** (*str*) – The tag to apply

`import_image_from_stream`(*stream, repository=None, tag=None, changes=None*)

`import_image_from_url`(*url, repository=None, tag=None, changes=None*)

Like **`import_image()`**, but only supports importing from a URL.

Parameters:

- **url** (*str*) – A URL pointing to a tar file.
- **repository** (*str*) – The repository to create
- **tag** (*str*) – The tag to apply

`inspect_distribution`(*image, auth_config=None*)

Get image digest and platform information by contacting the registry.

Parameters:

- **image** (*str*) – The image name to inspect
- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.

Returns:

A dict containing distribution data

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

`inspect_image`(*image*)

Get detailed information about an image. Similar to the `docker inspect` command, but only for images.

Parameters:
image (*str*) – The image to inspect
Returns:
Similar to the output of `docker inspect`, but as a
Return type:
(dict)

single dict

Raises:
docker.errors.APIError – If the server returns an error.

`load_image`(*`data, quiet=None`*)

Load an image that was previously saved using **`get_image()`** (or `docker save`). Similar to `docker load`.

Parameters:

- **data** (*binary*) – Image data to be loaded.
- **quiet** (*boolean*) – Suppress progress details in response.

Returns:
Progress output as JSON objects. Only available for
API version >= 1.23
Return type:
(generator)
Raises:
docker.errors.APIError – If the server returns an error.

`prune_images`(*`filters=None`*)

Delete unused images

Parameters:
filters (*dict*) – Filters to process on the prune list. Available filters: - dangling (bool): When set to true (or 1), prune only unused and untagged images.
Returns:
A dict containing a list of deleted image IDs and
the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`pull`(*`repository, tag=None, stream=False, auth_config=None, decode=False, platform=None, all_tags=False`*)

Pulls an image. Similar to the `docker pull` command.

Parameters:

- **repository** (*str*) – The repository to pull
- **tag** (*str*) – The tag to pull. If `tag` is `None` or empty, it is set to `latest`.
- **stream** (*bool*) – Stream the output as a generator. Make sure to consume the generator, otherwise pull might get cancelled.
- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.
- **decode** (*bool*) – Decode the JSON data from the server into dicts. Only applies with `stream=True`
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`
- **all_tags** (*bool*) – Pull all image tags, the `tag` parameter is ignored.

Returns:
The output

Return type:
(generator or str)

Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> resp = client.api.pull('busybox', stream=True, decode=True)
... for line in resp:
...     print(json.dumps(line, indent=4))
{
    "status": "Pulling image (latest) from busybox",
    "progressDetail": {},
    "id": "e72ac664f4f0"
}
{
    "status": "Pulling image (latest) from busybox, endpoint: ...",
    "progressDetail": {},
    "id": "e72ac664f4f0"
}
```

push(*repository, tag=None, stream=False, auth_config=None, decode=False*)

Push an image or a repository to the registry. Similar to the `docker push` command.

Parameters:

- **repository** (*str*) – The repository to push to
- **tag** (*str*) – An optional tag to push
- **stream** (*bool*) – Stream the output as a blocking generator

- **auth_config** (*dict*) – Override the credentials that are found in the config for this request. `auth_config` should contain the `username` and `password` keys to be valid.
- **decode** (*bool*) – Decode the JSON data from the server into dicts. Only applies with `stream=True`

Returns:

The output from the server.

Return type:

(generator or str)

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> resp = client.api.push(
...     'yourname/app',
...     stream=True,
...     decode=True,
... )
... for line in resp:
...   print(line)
{'status': 'Pushing repository yourname/app (1 tags)'}
{'status': 'Pushing','progressDetail': {}, 'id': '511136ea3c5a'}
{'status': 'Image already pushed, skipping', 'progressDetail':{},
 'id': '511136ea3c5a'}
...
```

`remove_image(image, force=False, noprune=False)`

Remove an image. Similar to the `docker rmi` command.

Parameters:

- **image** (*str*) – The image to remove
- **force** (*bool*) – Force removal of the image
- **noprune** (*bool*) – Do not delete untagged parents

`search(term, limit=None)`

Search for images on Docker Hub. Similar to the `docker search` command.

Parameters:

- **term** (*str*) – A term to search for.
- **limit** (*int*) – The maximum number of results to return.

Returns:

The response of the search.

Return type:

(list of dicts)
Raises:
docker.errors.APIError – If the server returns an error.
tag(*image, repository, tag=None, force=False*)
Tag an image into a repository. Similar to the `docker tag` command.

Parameters:

- **image** (*str*) – The image to tag
- **repository** (*str*) – The repository to set for the tag
- **tag** (*str*) – The tag name
- **force** (*bool*) – Force

Returns:
`True` if successful
Return type:
(bool)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> client.api.tag('ubuntu', 'localhost:5000/ubuntu', 'latest',
                   force=True)
```

Building images

build(*path=None, tag=None, quiet=False, fileobj=None, nocache=False, rm=False, timeout=None, custom_context=False, encoding=None, pull=False, forcerm=False, dockerfile=None, container_limits=None, decode=False, buildargs=None, gzip=False, shmsize=None, labels=None, cache_from=None, target=None, network_mode=None, squash=None, extra_hosts=None, platform=None, isolation=None, use_config_proxy=True*)
Similar to the `docker build` command. Either `path` or `fileobj` needs to be set. `path` can be a local path (to a directory containing a Dockerfile) or a remote URL. `fileobj` must be a readable file-like object to a Dockerfile.

If you have a tar file for the Docker build context (including a Dockerfile) already, pass a readable file-like object to `fileobj` and also pass `custom_context=True`. If the stream is compressed also, set `encoding` to the correct value (e.g `gzip`).

Example

```
>>> from io import BytesIO
>>> from docker import APIClient
>>> dockerfile = '''
... # Shared Volume
... FROM busybox:buildroot-2014.02
... VOLUME /data
... CMD ["/bin/sh"]
... '''
>>> f = BytesIO(dockerfile.encode('utf-8'))
>>> cli = APIClient(base_url='tcp://127.0.0.1:2375')
>>> response = [line for line in cli.build(
...     fileobj=f, rm=True, tag='yourname/volume'
... )]
>>> response
['{"stream":" ---\u003e a9eb17255234\n"}',
 '{"stream":"Step 1 : VOLUME /data\n"}',
 '{"stream":" ---\u003e Running in abdc1e6896c6\n"}',
 '{"stream":" ---\u003e 713bca62012e\n"}',
 '{"stream":"Removing intermediate container abdc1e6896c6\n"}',
 '{"stream":"Step 2 : CMD [\"/bin/sh\"]\n"}',
 '{"stream":" ---\u003e Running in dba30f2a1a7e\n"}',
 '{"stream":" ---\u003e 032b8b2855fc\n"}',
 '{"stream":"Removing intermediate container dba30f2a1a7e\n"}',
 '{"stream":"Successfully built 032b8b2855fc\n"}']
```

Parameters:

- **path** (*str*) – Path to the directory containing the Dockerfile
- **fileobj** – A file object to use as the Dockerfile. (Or a file-like object)
- **tag** (*str*) – A tag to add to the final image
- **quiet** (*bool*) – Whether to return the status
- **nocache** (*bool*) – Don't use the cache when set to `True`
- **rm** (*bool*) – Remove intermediate containers. The `docker build` command now defaults to `--rm=true`, but we have kept the old default of *False* to preserve backward compatibility
- **timeout** (*int*) – HTTP timeout
- **custom_context** (*bool*) – Optional if using `fileobj`
- **encoding** (*str*) – The encoding for a stream. Set to `gzip` for compressing
- **pull** (*bool*) – Downloads any updates to the FROM image in Dockerfiles
- **forcerm** (*bool*) – Always remove intermediate containers, even after unsuccessful builds
- **dockerfile** (*str*) – path within the build context to the Dockerfile
- **gzip** (*bool*) – If set to `True`, gzip compression/encoding is used
- **buildargs** (*dict*) – A dictionary of build arguments

- **container_limits** (*dict*) –
  A dictionary of limits applied to each container created by the build process. Valid keys:
    - memory (int): set memory limit for build
    - memswap (int): Total memory (memory + swap), -1 to disable
swap
    - cpushares (int): CPU shares (relative weight)
    - cpusetcpus (str): CPUs in which to allow execution, e.g.,
`"0-3"`, `"0,1"`
- **decode** (*bool*) – If set to `True`, the returned stream will be decoded into dicts on the fly. Default `False`
- **shmsize** (*int*) – Size of */dev/shm* in bytes. The size must be greater than 0. If omitted the system uses 64MB
- **labels** (*dict*) – A dictionary of labels to set on the image
- **cache_from** (`list`) – A list of images used for build cache resolution
- **target** (*str*) – Name of the build-stage to build in a multi-stage Dockerfile
- **network_mode** (*str*) – networking mode for the run commands during build
- **squash** (*bool*) – Squash the resulting images layers into a single layer.
- **extra_hosts** (*dict*) – Extra hosts to add to /etc/hosts in building containers, as a mapping of hostname to IP address.
- **platform** (*str*) – Platform in the format `os[/arch[/variant]]`
- **isolation** (*str*) – Isolation technology used during build. Default: *None*.
- **use_config_proxy** (*bool*) – If `True`, and if the docker client configuration file (`~/.docker/config.json` by default) contains a proxy configuration, the corresponding environment variables will be set in the container being built.

Returns:
A generator for the build output.
Raises:
- **docker.errors.APIError** – If the server returns an error.
- **TypeError** – If neither `path` nor `fileobj` is specified.

`prune_builds`(*`filters=None, keep_storage=None, all=None`*)
Delete the builder cache

Parameters:
- **filters** (*dict*) – Filters to process on the prune list. Needs Docker API v1.39+ Available filters: - dangling (bool): When set to true (or 1), prune only unused and untagged images. - until (str): Can be Unix timestamps, date formatted timestamps, or Go duration strings (e.g. 10m, 1h30m) computed relative to the daemon’s local time.
- **keep_storage** (*int*) – Amount of disk space in bytes to keep for cache. Needs Docker API v1.39+

- **all** (*bool*) – Remove all types of build cache. Needs Docker API v1.39+

Returns:
A dictionary containing information about the operation's result. The `SpaceReclaimed` key indicates the amount of bytes of disk space reclaimed.

Return type:
(dict)

Raises:
docker.errors.APIError – If the server returns an error.

Networks

`connect_container_to_network`(*`container, net_id, ipv4_address=None, ipv6_address=None, aliases=None, links=None, link_local_ips=None, driver_opt=None, mac_address=None`*)

Connect a container to a network.

Parameters:

- **container** (*str*) – container-id/name to be connected to the network
- **net_id** (*str*) – network id
- **aliases** (`list`) – A list of aliases for this endpoint. Names in that list can be used within the network to reach the container. Defaults to `None`.
- **links** (`list`) – A list of links for this endpoint. Containers declared in this list will be linked to this container. Defaults to `None`.
- **ipv4_address** (*str*) – The IP address of this container on the network, using the IPv4 protocol. Defaults to `None`.
- **ipv6_address** (*str*) – The IP address of this container on the network, using the IPv6 protocol. Defaults to `None`.
- **link_local_ips** (`list`) – A list of link-local (IPv4/IPv6) addresses.
- **mac_address** (*str*) – The MAC address of this container on the network. Defaults to `None`.

`create_network`(*`name, driver=None, options=None, ipam=None, check_duplicate=None, internal=False, labels=None, enable_ipv6=False, attachable=None, scope=None, ingress=None`*)

Create a network. Similar to the `docker network create`.

Parameters:

- **name** (*str*) – Name of the network
- **driver** (*str*) – Name of the driver used to create the network
- **options** (*dict*) – Driver options as a key-value dictionary
- **ipam** (*IPAMConfig*) – Optional custom IP scheme for the network.

- **check_duplicate** (*bool*) – Request daemon to check for networks with same name. Default: `None`.
- **internal** (*bool*) – Restrict external access to the network. Default `False`.
- **labels** (*dict*) – Map of labels to set on the network. Default `None`.
- **enable_ipv6** (*bool*) – Enable IPv6 on the network. Default `False`.
- **attachable** (*bool*) – If enabled, and the network is in the global scope, non-service containers on worker nodes will be able to connect to the network.
- **scope** (*str*) – Specify the network's scope (`local`, `global` or `swarm`)
- **ingress** (*bool*) – If set, create an ingress network which provides the routing-mesh in swarm mode.

Returns:

The created network reference object

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

Example

A network using the bridge driver:

```
>>> client.api.create_network("network1", driver="bridge")
```

You can also create more advanced networks with custom IPAM configurations. For example, setting the subnet to `192.168.52.0/24` and gateway address to `192.168.52.254`.

```
>>> ipam_pool = docker.types.IPAMPool(
    subnet='192.168.52.0/24',
    gateway='192.168.52.254'
)
>>> ipam_config = docker.types.IPAMConfig(
    pool_configs=[ipam_pool]
)
>>> client.api.create_network("network1", driver="bridge",
                                 ipam=ipam_config)
```

`disconnect_container_from_network`(*`container, net_id, force=False`*)

Disconnect a container from a network.

Parameters:

- **container** (*str*) – container ID or name to be disconnected from the network

- **net_id** (*str*) – network ID
- **force** (*bool*) – Force the container to disconnect from a network. Default: `False`

`inspect_network`(*`net_id, verbose=None, scope=None`*)

Get detailed information about a network.

Parameters:

- **net_id** (*str*) – ID of network
- **verbose** (*bool*) – Show the service details across the cluster in swarm mode.
- **scope** (*str*) – Filter the network by scope (`swarm`, `global` or `local`).

`networks`(*`names=None, ids=None, filters=None`*)

List networks. Similar to the `docker network ls` command.

Parameters:

- **names** (**`list`**) – List of names to filter by
- **ids** (**`list`**) – List of ids to filter by
- **filters** (*dict*) – Filters to be processed on the network list. Available filters: - `driver=[<driver-name>]` Matches a network's driver. - `label=[<key>]`, `label=[<key>=<value>]` or a list of such.
    - `type=["custom"|"builtin"]` Filters networks by type.

Returns:

List of network objects.

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

`prune_networks`(*`filters=None`*)

Delete unused networks

Parameters:

filters (*dict*) – Filters to process on the prune list.

Returns:

A dict containing a list of deleted network names and the amount of disk space reclaimed in bytes.

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

`remove_network`(*`net_id`*)

Remove a network. Similar to the `docker network rm` command.

Parameters:
net_id (*str*) – The network’s id

Volumes

create_volume(*name=None, driver=None, driver_opts=None, labels=None*)
Create and register a named volume

Parameters:

- **name** (*str*) – Name of the volume
- **driver** (*str*) – Name of the driver used to create the volume
- **driver_opts** (*dict*) – Driver options as a key-value dictionary
- **labels** (*dict*) – Labels to set on the volume

Returns:
The created volume reference object
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> volume = client.api.create_volume(
...     name='foobar',
...     driver='local',
...     driver_opts={'foo': 'bar', 'baz': 'false'},
...     labels={"key": "value"},
... )
... print(volume)
{u'Driver': u'local',
u'Labels': {u'key': u'value'},
u'Mountpoint': u'/var/lib/docker/volumes/foobar/_data',
u'Name': u'foobar',
u'Scope': u'local'}
```

inspect_volume(*name*)
Retrieve volume info by name.

Parameters:
name (*str*) – volume name
Returns:
Volume information dictionary

Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> client.api.inspect_volume('foobar')
{u'Driver': u'local',
 u'Mountpoint': u'/var/lib/docker/volumes/foobar/_data',
 u'Name': u'foobar'}
```

`prune_volumes`(*`filters=None`*)

Delete unused volumes

Parameters:
filters (*dict*) – Filters to process on the prune list.
Returns:
A dict containing a list of deleted volume names and
the amount of disk space reclaimed in bytes.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`remove_volume`(*`name, force=False`*)

Remove a volume. Similar to the `docker volume rm` command.

Parameters:

- **name** (*str*) – The volume's name
- **force** (*bool*) – Force removal of volumes that were already removed out of band by the volume driver plugin.

Raises:
docker.errors.APIError – If volume failed to remove.

`volumes`(*`filters=None`*)

List volumes currently registered by the docker daemon. Similar to
the `docker volume ls` command.

Parameters:
filters (*dict*) – Server-side list filtering options.
Returns:
Dictionary with list of volume objects as value of the `Volumes` key.
Return type:
(dict)

Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> client.api.volumes()
{u'Volumes': [{u'Driver': u'local',
   u'Mountpoint': u'/var/lib/docker/volumes/foobar/_data',
   u'Name': u'foobar'},
  {u'Driver': u'local',
   u'Mountpoint': u'/var/lib/docker/volumes/baz/_data',
   u'Name': u'baz'}]}
```

Executing commands in containers

`exec_create`(*`container, cmd, stdout=True, stderr=True, stdin=False, tty=False, privileged=False, user='', environment=None, workdir=None, detach_keys=None`*)
Sets up an exec instance in a running container.

Parameters:

- **container** (*str*) – Target container where exec instance will be created
- **cmd** (*str or list*) – Command to be executed
- **stdout** (*bool*) – Attach to stdout. Default: `True`
- **stderr** (*bool*) – Attach to stderr. Default: `True`
- **stdin** (*bool*) – Attach to stdin. Default: `False`
- **tty** (*bool*) – Allocate a pseudo-TTY. Default: False
- **privileged** (*bool*) – Run as privileged.
- **user** (*str*) – User to execute command as. Default: root
- **environment** (*dict or list*) – A dictionary or a list of strings in the following format `["PASSWORD=xxx"]` or `{"PASSWORD": "xxx"}`.
- **workdir** (*str*) – Path to working directory for this exec session
- **detach_keys** (*str*) – Override the key sequence for detaching a container. Format is a single character *[a-Z]* or *ctrl-<value>* where *<value>* is one of: *a-z*, *@*, *^*, *[*, *,* or *_*. ~/.docker/config.json is used by default.

Returns:
A dictionary with an exec `Id` key.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`exec_inspect(exec_id)`

Return low-level information about an exec command.

Parameters:

exec_id (*str*) – ID of the exec instance

Returns:

Dictionary of values returned by the endpoint.

Return type:

(dict)

Raises:

docker.errors.APIError – If the server returns an error.

`exec_resize(exec_id, height=None, width=None)`

Resize the tty session used by the specified exec command.

Parameters:

- **exec_id** (*str*) – ID of the exec instance
- **height** (*int*) – Height of tty session
- **width** (*int*) – Width of tty session

`exec_start(exec_id, detach=False, tty=False, stream=False, socket=False, demux=False)`

Start a previously set up exec instance.

Parameters:

- **exec_id** (*str*) – ID of the exec instance
- **detach** (*bool*) – If true, detach from the exec command. Default: False
- **tty** (*bool*) – Allocate a pseudo-TTY. Default: False
- **stream** (*bool*) – Return response data progressively as an iterator of strings, rather than a single string.
- **socket** (*bool*) – Return the connection socket to allow custom read/write operations. Must be closed by the caller when done.
- **demux** (*bool*) – Return stdout and stderr separately

Returns:

If `stream=True`, a generator yielding response chunks. If `socket=True`, a socket object for the connection. A string containing response data otherwise.
If `demux=True`, a tuple with two elements of type byte: stdout and stderr.

Return type:

(generator or str or tuple)

Raises:

docker.errors.APIError – If the server returns an error.

Swarms

`create_swarm_spec(*args, **kwargs)`

Create a `docker.types.SwarmSpec` instance that can be used as the `swarm_spec` argument in `init_swarm()`.

Parameters:

- **task_history_retention_limit** (*int*) – Maximum number of tasks history stored.
- **snapshot_interval** (*int*) – Number of logs entries between snapshot.
- **keep_old_snapshots** (*int*) – Number of snapshots to keep beyond the current snapshot.
- **log_entries_for_slow_followers** (*int*) – Number of log entries to keep around to sync up slow followers after a snapshot is created.
- **heartbeat_tick** (*int*) – Amount of ticks (in seconds) between each heartbeat.
- **election_tick** (*int*) – Amount of ticks (in seconds) needed without a leader to trigger a new election.
- **dispatcher_heartbeat_period** (*int*) – The delay for an agent to send a heartbeat to the dispatcher.
- **node_cert_expiry** (*int*) – Automatic expiry for nodes certificates.
- **external_cas** (`list`) – Configuration for forwarding signing requests to an external certificate authority. Use a list of `docker.types.SwarmExternalCA`.
- **name** (*string*) – Swarm's name
- **labels** (*dict*) – User-defined key/value metadata.
- **signing_ca_cert** (*str*) – The desired signing CA certificate for all swarm node TLS leaf certificates, in PEM format.
- **signing_ca_key** (*str*) – The desired signing CA key for all swarm node TLS leaf certificates, in PEM format.
- **ca_force_rotate** (*int*) – An integer whose purpose is to force swarm to generate a new signing CA certificate and key, if none have been specified.
- **autolock_managers** (*boolean*) – If set, generate a key and use it to lock data stored on the managers.
- **log_driver** (*DriverConfig*) – The default log driver to use for tasks created in the orchestrator.

Returns:

`docker.types.SwarmSpec`

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> spec = client.api.create_swarm_spec(
  snapshot_interval=5000, log_entries_for_slow_followers=1200
)
>>> client.api.init_swarm(
  advertise_addr='eth0', listen_addr='0.0.0.0:5000',
  force_new_cluster=False, swarm_spec=spec
)
```

`get_unlock_key()`

Get the unlock key for this Swarm manager.

Returns:
A `dict` containing an `UnlockKey` member

`init_swarm`(*`advertise_addr=None, listen_addr='0.0.0.0:2377', force_new_cluster=False, swarm_spec=None, default_addr_pool=None, subnet_size=None, data_path_addr=None, data_path_port=None`*)

Initialize a new Swarm using the current connected engine as the first node.

Parameters:

- **advertise_addr** (*string*) – Externally reachable address advertised to other nodes. This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the port number from the listen address is used. If `advertise_addr` is not specified, it will be automatically detected when possible. Default: None
- **listen_addr** (*string*) – Listen address used for inter-manager communication, as well as determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP). This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the default swarm listening port is used. Default: '0.0.0.0:2377'
- **force_new_cluster** (*bool*) – Force creating a new Swarm, even if already part of one. Default: False
- **swarm_spec** (*dict*) – Configuration settings of the new Swarm. Use `APIClient.create_swarm_spec` to generate a valid configuration. Default: None
- **default_addr_pool** (*list of strings*) – Default Address Pool specifies default subnet pools for global scope networks. Each pool should be specified as a CIDR block, like '10.0.0.0/8'. Default: None
- **subnet_size** (*int*) – SubnetSize specifies the subnet size of the networks created from the default subnet pool. Default: None
- **data_path_addr** (*string*) – Address or interface to use for data path traffic. For example, 192.168.1.1, or an interface, like eth0.

- **data_path_port** (*int*) – Port number to use for data path traffic. Acceptable port range is 1024 to 49151. If set to `None` or 0, the default port 4789 will be used. Default: None

Returns:
The ID of the created node.
Return type:
(str)
Raises:
docker.errors.APIError – If the server returns an error.

`inspect_node(`*`node_id`*`)`

Retrieve low-level information about a swarm node

Parameters:
node_id (*string*) – ID of the node to be inspected.
Returns:
A dictionary containing data about this node.
Raises:
docker.errors.APIError – If the server returns an error.

`inspect_swarm()`

Retrieve low-level information about the current swarm.

Returns:
A dictionary containing data about the swarm.
Raises:
docker.errors.APIError – If the server returns an error.

`join_swarm(`*`remote_addrs, join_token, listen_addr='0.0.0.0:2377', advertise_addr=None, data_path_addr=None`*`)`

Make this Engine join a swarm that has already been created.

Parameters:

- **remote_addrs** (`list`) – Addresses of one or more manager nodes already participating in the Swarm to join.
- **join_token** (*string*) – Secret token for joining this Swarm.
- **listen_addr** (*string*) – Listen address used for inter-manager communication if the node gets promoted to manager, as well as determining the networking interface used for the VXLAN Tunnel Endpoint (VTEP).
  Default: `'0.0.0.0:2377`
- **advertise_addr** (*string*) – Externally reachable address advertised to other nodes. This can either be an address/port combination in the form `192.168.1.1:4567`, or an interface followed by a port number, like `eth0:4567`. If the port number is omitted, the port number from the listen

address is used. If AdvertiseAddr is not specified, it will be automatically detected when possible. Default: `None`

- **data_path_addr** (*string*) – Address or interface to use for data path traffic. For example, 192.168.1.1, or an interface, like eth0.

Returns:

`True` if the request went through.

Raises:

docker.errors.APIError – If the server returns an error.

`leave_swarm`(*force=False*)

Leave a swarm.

Parameters:

force (*bool*) – Leave the swarm even if this node is a manager. Default: `False`

Returns:

`True` if the request went through.

Raises:

docker.errors.APIError – If the server returns an error.

`nodes`(*filters=None*)

List swarm nodes.

Parameters:

filters (*dict*) – Filters to process on the nodes list. Valid filters: `id`, `name`, `membership` and `role`. Default: `None`

Returns:

A list of dictionaries containing data about each swarm node.

Raises:

docker.errors.APIError – If the server returns an error.

`remove_node`(*node_id, force=False*)

Remove a node from the swarm.

Parameters:

- **node_id** (*string*) – ID of the node to be removed.
- **force** (*bool*) – Force remove an active node. Default: *False*

Raises:

- **docker.errors.NotFound** – If the node referenced doesn't exist in the swarm.
- **docker.errors.APIError** – If the server returns an error.

Returns:

True if the request was successful.

`unlock_swarm`(*key*)

Unlock a locked swarm.

Parameters:

key (*string*) – The unlock key as provided by `get_unlock_key()`

Raises:

- **docker.errors.InvalidArgument** – If the key argument is in an incompatible format
- **docker.errors.APIError** – If the server returns an error.

Returns:

True if the request was successful.

Example

```
>>> key = client.api.get_unlock_key()
>>> client.unlock_swarm(key)
```

`update_node`(*`node_id, version, node_spec=None`*)

Update the node's configuration

Parameters:

- **node_id** (*string*) – ID of the node to be updated.
- **version** (*int*) – The version number of the node object being updated. This is required to avoid conflicting writes.
- **node_spec** (*dict*) – Configuration settings to update. Any values not provided will be removed. Default: `None`

Returns:

True if the request went through.

Raises:

docker.errors.APIError – If the server returns an error.

Example

```
>>> node_spec = {'Availability': 'active',
             'Name': 'node-name',
             'Role': 'manager',
             'Labels': {'foo': 'bar'}
            }
>>> client.api.update_node(node_id='24ifsmvkjbyhk', version=8,
    node_spec=node_spec)
```

`update_swarm`(*`version, swarm_spec=None, rotate_worker_token=False, rotate_manager_token=False, rotate_manager_unlock_key=False`*)

Update the Swarm's configuration

Parameters:

- **version** (*int*) – The version number of the swarm object being updated. This is required to avoid conflicting writes.
- **swarm_spec** (*dict*) – Configuration settings to update. Use **`create_swarm_spec()`** to generate a valid configuration. Default: `None`.
- **rotate_worker_token** (*bool*) – Rotate the worker join token. Default: `False`.
- **rotate_manager_token** (*bool*) – Rotate the manager join token. Default: `False`.
- **rotate_manager_unlock_key** (*bool*) – Rotate the manager unlock key. Default: `False`.

Returns:
`True` if the request went through.
Raises:
docker.errors.APIError – If the server returns an error.

Services

`create_service`(*`task_template, name=None, labels=None, mode=None, update_config=None, networks=None, endpoint_config=None, endpoint_spec=None, rollback_config=None`*)
Create a service.

Parameters:
- **task_template** (*TaskTemplate*) – Specification of the task to start as part of the new service.
- **name** (*string*) – User-defined name for the service. Optional.
- **labels** (*dict*) – A map of labels to associate with the service. Optional.
- **mode** (*ServiceMode*) – Scheduling mode for the service (replicated or global). Defaults to replicated.
- **update_config** (*UpdateConfig*) – Specification for the update strategy of the service. Default: `None`
- **rollback_config** (*RollbackConfig*) – Specification for the rollback strategy of the service. Default: `None`
- **networks** (**`list`**) – List of network names or IDs or **`NetworkAttachmentConfig`** to attach the service to. Default: `None`.
- **endpoint_spec** (*EndpointSpec*) – Properties that can be configured to access and load balance a service. Default: `None`.

Returns:
A dictionary containing an `ID` key for the newly created service.
Raises:
docker.errors.APIError – If the server returns an error.

`inspect_service`(*`service, insert_defaults=None`*)

Return information about a service.

Parameters:

- **service** (*str*) – Service name or ID.
- **insert_defaults** (*boolean*) – If true, default values will be merged into the service inspect output.

Returns:
A dictionary of the server-side representation of the service, including all relevant properties.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`inspect_task(task)`

Retrieve information about a task.

Parameters:
task (*str*) – Task ID
Returns:
Information about the task.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`remove_service(service)`

Stop and remove a service.

Parameters:
service (*str*) – Service name or ID
Returns:
`True` if successful.
Raises:
docker.errors.APIError – If the server returns an error.

`service_logs(service, details=False, follow=False, stdout=False, stderr=False, since=0, timestamps=False, tail='all', is_tty=None)`

Get log stream for a service. Note: This endpoint works only for services with the `json-file` or `journald` logging drivers.

Parameters:

- **service** (*str*) – ID or name of the service
- **details** (*bool*) – Show extra details provided to logs. Default: `False`

- **follow** (*bool*) – Keep connection open to read logs as they are sent by the Engine. Default: `False`
- **stdout** (*bool*) – Return logs from `stdout`. Default: `False`
- **stderr** (*bool*) – Return logs from `stderr`. Default: `False`
- **since** (*int*) – UNIX timestamp for the logs staring point. Default: 0
- **timestamps** (*bool*) – Add timestamps to every log line.
- **tail** (*string or int*) – Number of log lines to be returned, counting from the current end of the logs. Specify an integer or `'all'` to output all log lines. Default: `all`
- **is_tty** (*bool*) – Whether the service's **`ContainerSpec`** enables the TTY option. If omitted, the method will query the Engine for the information, causing an additional roundtrip.

Returns (generator): Logs for the service.

`services`(*`filters=None, status=None`*)

List services.

Parameters:

- **filters** (*dict*) – Filters to process on the nodes list. Valid filters: `id`, `name` , `label` and `mode`. Default: `None`.
- **status** (*bool*) – Include the service task count of running and desired tasks. Default: `None`.

Returns:

A list of dictionaries containing data about each service.

Raises:

docker.errors.APIError – If the server returns an error.

`tasks`(*`filters=None`*)

Retrieve a list of tasks.

Parameters:

filters (*dict*) – A map of filters to process on the tasks list. Valid filters: `id`, `name`, `service`, `node`, `label` and `desired-state`.

Returns:

List of task dictionaries.

Return type:

(`list`)

Raises:

docker.errors.APIError – If the server returns an error.

`update_service`(*`service, version, task_template=None, name=None, labels=None, mode=None, update_config=None, networks=None, endpoint_config=None, endpoint_spec=None, fetch_current_spec=False, rollback_config=None`*)

Update a service.

Parameters:

- **service** (*string*) – A service identifier (either its name or service ID).
- **version** (*int*) – The version number of the service object being updated. This is required to avoid conflicting writes.
- **task_template** (*TaskTemplate*) – Specification of the updated task to start as part of the service.
- **name** (*string*) – New name for the service. Optional.
- **labels** (*dict*) – A map of labels to associate with the service. Optional.
- **mode** (*ServiceMode*) – Scheduling mode for the service (replicated or global). Defaults to replicated.
- **update_config** (*UpdateConfig*) – Specification for the update strategy of the service. Default: `None`.
- **rollback_config** (*RollbackConfig*) – Specification for the rollback strategy of the service. Default: `None`
- **networks** (`list`) – List of network names or IDs or `NetworkAttachmentConfig` to attach the service to. Default: `None`.
- **endpoint_spec** (*EndpointSpec*) – Properties that can be configured to access and load balance a service. Default: `None`.
- **fetch_current_spec** (*boolean*) – Use the undefined settings from the current specification of the service. Default: `False`

Returns:

A dictionary containing a `Warnings` key.

Raises:

docker.errors.APIError – If the server returns an error.

Plugins

`configure_plugin(`*`name, options`*`)`

Configure a plugin.

Parameters:

- **name** (*string*) – The name of the plugin. The `:latest` tag is optional, and is the default if omitted.
- **options** (*dict*) – A key-value mapping of options

Returns:

`True` if successful

`create_plugin(`*`name, plugin_data_dir, gzip=False`*`)`

Create a new plugin.

Parameters:

- **name** (*string*) – The name of the plugin. The `:latest` tag is optional, and is the default if omitted.
- **plugin_data_dir** (*string*) – Path to the plugin data directory. Plugin data directory must contain the `config.json` manifest file and the `rootfs` directory.
- **gzip** (*bool*) – Compress the context using gzip. Default: False

Returns:
`True` if successful

`disable_plugin``(`*`name, force=False`*`)`

Disable an installed plugin.

Parameters:

- **name** (*string*) – The name of the plugin. The `:latest` tag is optional, and is the default if omitted.
- **force** (*bool*) – To enable the force query parameter.

Returns:
`True` if successful

`enable_plugin``(`*`name, timeout=0`*`)`

Enable an installed plugin.

Parameters:

- **name** (*string*) – The name of the plugin. The `:latest` tag is optional, and is the default if omitted.
- **timeout** (*int*) – Operation timeout (in seconds). Default: 0

Returns:
`True` if successful

`inspect_plugin``(`*`name`*`)`

Retrieve plugin metadata.

Parameters:
name (*string*) – The name of the plugin. The `:latest` tag is optional, and is the default if omitted.

Returns:
A dict containing plugin info

`plugin_privileges``(`*`name`*`)`

Retrieve list of privileges to be granted to a plugin.

Parameters:
name (*string*) – Name of the remote plugin to examine. The `:latest` tag is optional, and is the default if omitted.

Returns:

A list of dictionaries representing the plugin's permissions

`plugins()`

Retrieve a list of installed plugins.

Returns:

A list of dicts, one per plugin

`pull_plugin`(*`remote, privileges, name=None`*)

Pull and install a plugin. After the plugin is installed, it can be enabled using **`enable_plugin()`**.

Parameters:

- **remote** (*string*) – Remote reference for the plugin to install. The `:latest` tag is optional, and is the default if omitted.
- **privileges** (**`list`**) – A list of privileges the user consents to grant to the plugin. Can be retrieved using **`plugin_privileges()`**.
- **name** (*string*) – Local name for the pulled plugin. The `:latest` tag is optional, and is the default if omitted.

Returns:

An iterable object streaming the decoded API logs

`push_plugin`(*`name`*)

Push a plugin to the registry.

Parameters:

name (*string*) – Name of the plugin to upload. The `:latest` tag is optional, and is the default if omitted.

Returns:

`True` if successful

`remove_plugin`(*`name, force=False`*)

Remove an installed plugin.

Parameters:

- **name** (*string*) – Name of the plugin to remove. The `:latest` tag is optional, and is the default if omitted.
- **force** (*bool*) – Disable the plugin before removing. This may result in issues if the plugin is in use by a container.

Returns:

`True` if successful

`upgrade_plugin`(*`name, remote, privileges`*)

Upgrade an installed plugin.

Parameters:

- **name** (*string*) – Name of the plugin to upgrade. The `:latest` tag is optional and is the default if omitted.
- **remote** (*string*) – Remote reference to upgrade to. The `:latest` tag is optional and is the default if omitted.
- **privileges** (`list`) – A list of privileges the user consents to grant to the plugin. Can be retrieved using **`plugin_privileges()`**.

Returns:
An iterable object streaming the decoded API logs

Secrets

`create_secret`(*name*, *data*, *labels=None*, *driver=None*)
Create a secret

Parameters:
- **name** (*string*) – Name of the secret
- **data** (*bytes*) – Secret data to be stored
- **labels** (*dict*) – A mapping of labels to assign to the secret
- **driver** (*DriverConfig*) – A custom driver configuration. If unspecified, the default `internal` driver will be used

Returns (dict): ID of the newly created secret

`inspect_secret`(*id*)
Retrieve secret metadata

Parameters:
id (*string*) – Full ID of the secret to inspect

Returns (dict): A dictionary of metadata

Raises:
docker.errors.NotFound – if no secret with that ID exists

`remove_secret`(*id*)
Remove a secret

Parameters:
id (*string*) – Full ID of the secret to remove

Returns (boolean): True if successful

Raises:
docker.errors.NotFound – if no secret with that ID exists

secrets(*filters=None*)

List secrets

Parameters:

- **filters** (*list. Available*) – A map of filters to process on the secrets
- **filters** – `names`

Returns (list): A list of secrets

The Docker daemon

df()

Get data usage information.

Returns:
A dictionary representing different resource categories and their respective data usage.
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

events(*since=None, until=None, filters=None, decode=None*)

Get real-time events from the server. Similar to the `docker events` command.

Parameters:

- **since** (*UTC datetime or int*) – Get events from this point
- **until** (*UTC datetime or int*) – Get events until this point
- **filters** (*dict*) – Filter the events by event time, container or image
- **decode** (*bool*) – If set to true, stream will be decoded into dicts on the fly. False by default.

Returns:
A **`docker.types.daemon.CancellableStream`** generator
Raises:
docker.errors.APIError – If the server returns an error.

Example

```
>>> for event in client.events(decode=True)
...   print(event)
{u'from': u'image/with:tag',
 u'id': u'container-id',
 u'status': u'start',
 u'time': 1423339459}
...
```

or

```
>>> events = client.events()
>>> for event in events:
...   print(event)
>>> # and cancel from another thread
>>> events.close()
```

`info()`

Display system-wide information. Identical to the `docker info` command.

Returns:
The info as a dict
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`login`(*`username, password=None, email=None, registry=None, reauth=False, dockercfg_path=None`*)

Authenticate with a registry. Similar to the `docker login` command.

Parameters:

- **username** (*str*) – The registry username
- **password** (*str*) – The plaintext password
- **email** (*str*) – The email for the registry account
- **registry** (*str*) – URL to the registry. E.g. `https://index.docker.io/v1/`
- **reauth** (*bool*) – Whether or not to refresh existing authentication on the Docker server.
- **dockercfg_path** (*str*) – Use a custom path for the Docker config file (default `$HOME/.docker/config.json` if present, otherwise `$HOME/.dockercfg`)

Returns:
The response from the login request
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

`ping()`

Checks the server is responsive. An exception will be raised if it isn’t responding.

Returns:
(bool) The response from the server.
Raises:
docker.errors.APIError – If the server returns an error.
`version`(*api_version=True*)
Returns version information from the server. Similar to the `docker version` command.

Returns:
The server version information
Return type:
(dict)
Raises:
docker.errors.APIError – If the server returns an error.

Configuration types

class **`ConfigReference`**(*config_id, config_name, filename=None, uid=None, gid=None, mode=292*)

Config reference to be used as part of a **`ContainerSpec`**. Describes how a config is made accessible inside the service's containers.

Parameters:

- **config_id** (*string*) – Config's ID
- **config_name** (*string*) – Config's name as defined at its creation.
- **filename** (*string*) – Name of the file containing the config. Defaults to the config's name if not specified.
- **uid** (*string*) – UID of the config file's owner. Default: 0
- **gid** (*string*) – GID of the config file's group. Default: 0
- **mode** (*int*) – File access mode inside the container. Default: 0o444

class **`ContainerSpec`**(*image, command=None, args=None, hostname=None, env=None, workdir=None, user=None, labels=None, mounts=None, stop_grace_period=None, secrets=None, tty=None, groups=None, open_stdin=None, read_only=None, stop_signal=None, healthcheck=None, hosts=None, dns_config=None, configs=None, privileges=None, isolation=None, init=None, cap_add=None, cap_drop=None, sysctls=None*)

Describes the behavior of containers that are part of a task, and is used when declaring a **`TaskTemplate`**.

Parameters:

- **image** (*string*) – The image name to use for the container.
- **command** (*string or list*) – The command to be run in the image.

- **args** (`list`) – Arguments to the command.
- **hostname** (*string*) – The hostname to set on the container.
- **env** (*dict*) – Environment variables.
- **workdir** (*string*) – The working directory for commands to run in.
- **user** (*string*) – The user inside the container.
- **labels** (*dict*) – A map of labels to associate with the service.
- **mounts** (`list`) – A list of specifications for mounts to be added to containers created as part of the service. See the `Mount` class for details.
- **stop_grace_period** (*int*) – Amount of time to wait for the container to terminate before forcefully killing it.
- **secrets** (`list`) – List of `SecretReference` to be made available inside the containers.
- **tty** (*boolean*) – Whether a pseudo-TTY should be allocated.
- **groups** (`list`) – A list of additional groups that the container process will run as.
- **open_stdin** (*boolean*) – Open `stdin`
- **read_only** (*boolean*) – Mount the container's root filesystem as read only.
- **stop_signal** (*string*) – Set signal to stop the service's containers
- **healthcheck** (*Healthcheck*) – Healthcheck configuration for this service.
- **hosts** (`dict`) – A set of host to IP mappings to add to the container's `hosts` file.
- **dns_config** (*DNSConfig*) – Specification for DNS related configurations in resolver configuration file.
- **configs** (`list`) – List of `ConfigReference` that will be exposed to the service.
- **privileges** (*Privileges*) – Security options for the service's containers.
- **isolation** (*string*) – Isolation technology used by the service's containers. Only used for Windows containers.
- **init** (*boolean*) – Run an init inside the container that forwards signals and reaps processes.
- **cap_add** (`list`) – A list of kernel capabilities to add to the default set for the container.
- **cap_drop** (`list`) – A list of kernel capabilities to drop from the default set for the container.
- **sysctls** (`dict`) – A dict of sysctl values to add to the container

class **DNSConfig**(*nameservers=None, search=None, options=None*)

Specification for DNS related configurations in resolver configuration file (`resolv.conf`). Part of a `ContainerSpec` definition.

Parameters:

- **nameservers** (`list`) – The IP addresses of the name servers.
- **search** (`list`) – A search list for host-name lookup.

- **options** (`list`) – A list of internal resolver variables to be modified (e.g., `debug`, `ndots:3`, etc.).

class `DriverConfig`(*name*, *options=None*)

Indicates which driver to use, as well as its configuration. Can be used as `log_driver` in a **`ContainerSpec`**, for the *driver_config* in a volume **`Mount`**, or as the driver object in **`create_secret()`**.

Parameters:

- **name** (*string*) – Name of the driver to use.
- **options** (*dict*) – Driver-specific options. Default: `None`.

class `EndpointSpec`(*mode=None*, *ports=None*)

Describes properties to access and load-balance a service.

Parameters:

- **mode** (*string*) – The mode of resolution to use for internal load balancing between tasks (`'vip'` or `'dnsrr'`). Defaults to `'vip'` if not provided.
- **ports** (*dict*) – Exposed ports that this service is accessible on from the outside, in the form of `{ published_port: target_port }` or `{ published_port: <port_config_tuple> }`. Port config tuple format is `(target_port [, protocol [, publish_mode]])`. Ports can only be provided if the `vip` resolution mode is used.

class `Healthcheck`(***kwargs*)

Defines a healthcheck configuration for a container or service.

Parameters:

- **test** (`list` or str) –
  Test to perform to determine container health. Possible values:
  - Empty list: Inherit healthcheck from parent image
  - `["NONE"]`: Disable healthcheck
  - `["CMD", args...]`: exec arguments directly.
  - `["CMD-SHELL", command]`: Run command in the system’s default shell.

  If a string is provided, it will be used as a `CMD-SHELL` command.
- **interval** (*int*) – The time to wait between checks in nanoseconds. It should be 0 or at least 1000000 (1 ms).
- **timeout** (*int*) – The time to wait before considering the check to have hung. It should be 0 or at least 1000000 (1 ms).
- **retries** (*int*) – The number of consecutive failures needed to consider a container as unhealthy.

- **start_period** (*int*) – Start period for the container to initialize before starting health-retries countdown in nanoseconds. It should be 0 or at least 1000000 (1 ms).

class **IPAMConfig**(*driver='default'*, *pool_configs=None*, *options=None*)

Create an IPAM (IP Address Management) config dictionary to be used with `create_network()`.

Parameters:

- **driver** (*str*) – The IPAM driver to use. Defaults to `default`.
- **pool_configs** (`list`) – A list of pool configurations (`IPAMPool`). Defaults to empty list.
- **options** (*dict*) – Driver options as a key-value dictionary. Defaults to *None*.

Example

```
>>> ipam_config = docker.types.IPAMConfig(driver='default')
>>> network = client.create_network('network1', ipam=ipam_config)
```

class **IPAMPool**(*subnet=None*, *iprange=None*, *gateway=None*, *aux_addresses=None*)

Create an IPAM pool config dictionary to be added to the `pool_configs` parameter of `IPAMConfig`.

Parameters:

- **subnet** (*str*) – Custom subnet for this IPAM pool using the CIDR notation. Defaults to `None`.
- **iprange** (*str*) – Custom IP range for endpoints in this IPAM pool using the CIDR notation. Defaults to `None`.
- **gateway** (*str*) – Custom IP address for the pool's gateway.
- **aux_addresses** (*dict*) – A dictionary of `key -> ip_address` relationships specifying auxiliary addresses that need to be allocated by the IPAM driver.

Example

```
>>> ipam_pool = docker.types.IPAMPool(
    subnet='124.42.0.0/16',
    iprange='124.42.0.0/24',
    gateway='124.42.0.254',
    aux_addresses={
        'reserved1': '124.42.1.1'
    }
)
>>> ipam_config = docker.types.IPAMConfig(
        pool_configs=[ipam_pool])
```

class **LogConfig**(***kwargs*)

Configure logging for a container, when provided as an argument to **`create_host_config()`**. You may refer to the official logging driver documentation for more information.

Parameters:

- **type** (*str*) – Indicate which log driver to use. A set of valid drivers is provided as part of the **`LogConfig.types`** enum. Other values may be accepted depending on the engine version and available logging plugins.
- **config** (*dict*) – A driver-dependent configuration dictionary. Please refer to the driver's documentation for a list of valid config keys.

Example

```
>>> from docker.types import LogConfig
>>> lc = LogConfig(type=LogConfig.types.JSON, config={
...   'max-size': '1g',
...   'labels': 'production_status,geo'
... })
>>> hc = client.create_host_config(log_config=lc)
>>> container = client.create_container('busybox', 'true',
...    host_config=hc)
>>> client.inspect_container(container)['HostConfig']['LogConfig']
{
    'Type': 'json-file',
    'Config': {'labels': 'production_status,geo', 'max-size': '1g'}
}
```

class **Mount**(*target, source, type='volume', read_only=False, consistency=None, propagation=None, no_copy=False, labels=None, driver_config=None, tmpfs_size=None, tmpfs_mode=None*)

Describes a mounted folder's configuration inside a container. A list of **`Mount`** would be used as part of a **`ContainerSpec`**.

Parameters:

- **target** (*string*) – Container path.
- **source** (*string*) – Mount source (e.g. a volume name or a host path).
- **type** (*string*) – The mount type (`bind` / `volume` / `tmpfs` / `npipe`). Default: `volume`.
- **read_only** (*bool*) – Whether the mount should be read-only.
- **consistency** (*string*) – The consistency requirement for the mount. One of
- **default`** –
- **consistent** –

- **cached** –
- **delegated.** –
- **propagation** (*string*) – A propagation mode with the value `[r]private`, `[r]shared`, or `[r]slave`. Only valid for the `bind` type.
- **no_copy** (*bool*) – False if the volume should be populated with the data from the target. Default: `False`. Only valid for the `volume` type.
- **labels** (*dict*) – User-defined name and labels for the volume. Only valid for the `volume` type.
- **driver_config** (*DriverConfig*) – Volume driver configuration. Only valid for the `volume` type.
- **tmpfs_size** (*int or string*) – The size for the tmpfs mount in bytes.
- **tmpfs_mode** (*int*) – The permission mode for the tmpfs mount.

class **NetworkAttachmentConfig**(*target*, *aliases=None*, *options=None*)

Network attachment options for a service.

Parameters:

- **target** (*str*) – The target network for attachment. Can be a network name or ID.
- **aliases** (`list`) – A list of discoverable alternate names for the service.
- **options** (`dict`) – Driver attachment options for the network target.

class **Placement**(*constraints=None*, *preferences=None*, *platforms=None*, *maxreplicas=None*)

Placement constraints to be used as part of a `TaskTemplate`

Parameters:

- **constraints** (`list` of str) – A list of constraints
- **preferences** (`list` of tuple) – Preferences provide a way to make the scheduler aware of factors such as topology. They are provided in order from highest to lowest precedence and are expressed as `(strategy, descriptor)` tuples. See `PlacementPreference` for details.
- **maxreplicas** (*int*) – Maximum number of replicas per node
- **platforms** (`list` of tuple) – A list of platforms expressed as `(arch, os)` tuples

class **PlacementPreference**(*strategy*, *descriptor*)

Placement preference to be used as an element in the list of preferences for `Placement` objects.

Parameters:

- **strategy** (*string*) – The placement strategy to implement. Currently, the only supported strategy is `spread`.
- **descriptor** (*string*) – A label descriptor. For the spread strategy, the scheduler will try to spread tasks evenly over groups of nodes identified by this label.

class **Privileges**(*credentialspec_file=None, credentialspec_registry=None, selinux_disable=None, selinux_user=None, selinux_role=None, selinux_type=None, selinux_level=None*)

Security options for a service's containers. Part of a **`ContainerSpec`** definition.

Parameters:

- **credentialspec_file** (*str*) – Load credential spec from this file. The file is read by the daemon, and must be present in the CredentialSpecs subdirectory in the docker data directory, which defaults to `C:\ProgramData\Docker\` on Windows. Can not be combined with credentialspec_registry.
- **credentialspec_registry** (*str*) – Load credential spec from this value in the Windows registry. The specified registry value must be located in: `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion \Virtualization\Containers\CredentialSpecs`. Can not be combined with credentialspec_file.
- **selinux_disable** (*boolean*) – Disable SELinux
- **selinux_user** (*string*) – SELinux user label
- **selinux_role** (*string*) – SELinux role label
- **selinux_type** (*string*) – SELinux type label
- **selinux_level** (*string*) – SELinux level label

class **Resources**(*cpu_limit=None, mem_limit=None, cpu_reservation=None, mem_reservation=None, generic_resources=None*)

Configures resource allocation for containers when made part of a **`ContainerSpec`**.

Parameters:

- **cpu_limit** (*int*) – CPU limit in units of 10^9 CPU shares.
- **mem_limit** (*int*) – Memory limit in Bytes.
- **cpu_reservation** (*int*) – CPU reservation in units of 10^9 CPU shares.
- **mem_reservation** (*int*) – Memory reservation in Bytes.
- **generic_resources** (dict or **`list`**) – Node level generic resources, for example a GPU, using the following format: `{ resource_name: resource_value }`. Alternatively, a list of of resource specifications as defined by the Engine API.

class **RestartPolicy**(*condition='none', delay=0, max_attempts=0, window=0*)

Used when creating a **`ContainerSpec`**, dictates whether a container should restart after stopping or failing.

Parameters:

- **condition** (*string*) – Condition for restart (`none`, `on-failure`, or `any`). Default: *none*.
- **delay** (*int*) – Delay between restart attempts. Default: 0

- **max_attempts** (*int*) – Maximum attempts to restart a given container before giving up. Default value is 0, which is ignored.
- **window** (*int*) – Time window used to evaluate the restart policy. Default value is 0, which is unbounded.

class **RollbackConfig**(*parallelism=0, delay=None, failure_action='continue', monitor=None, max_failure_ratio=None, order=None*)

Used to specify the way container rollbacks should be performed by a service

Parameters:

- **parallelism** (*int*) – Maximum number of tasks to be rolled back in one iteration (0 means unlimited parallelism). Default: 0
- **delay** (*int*) – Amount of time between rollbacks, in nanoseconds.
- **failure_action** (*string*) – Action to take if a rolled back task fails to run, or stops running during the rollback. Acceptable values are `continue`, `pause` or `rollback`. Default: `continue`
- **monitor** (*int*) – Amount of time to monitor each rolled back task for failures, in nanoseconds.
- **max_failure_ratio** (*float*) – The fraction of tasks that may fail during a rollback before the failure action is invoked, specified as a floating point number between 0 and 1. Default: 0
- **order** (*string*) – Specifies the order of operations when rolling out a rolled back task. Either `start-first` or `stop-first` are accepted.

class **SecretReference**(*secret_id, secret_name, filename=None, uid=None, gid=None, mode=292*)

Secret reference to be used as part of a **`ContainerSpec`**. Describes how a secret is made accessible inside the service's containers.

Parameters:

- **secret_id** (*string*) – Secret's ID
- **secret_name** (*string*) – Secret's name as defined at its creation.
- **filename** (*string*) – Name of the file containing the secret. Defaults to the secret's name if not specified.
- **uid** (*string*) – UID of the secret file's owner. Default: 0
- **gid** (*string*) – GID of the secret file's group. Default: 0
- **mode** (*int*) – File access mode inside the container. Default: 0o444

class **ServiceMode**(*mode, replicas=None, concurrency=None*)

Indicate whether a service or a job should be deployed as a replicated or global service, and associated parameters

Parameters:

- **mode** (*string*) – Can be either `replicated`, `global`, `replicated-job` or `global-job`

- **replicas** (*int*) – Number of replicas. For replicated services only.
- **concurrency** (*int*) – Number of concurrent jobs. For replicated job services only.

class `SwarmExternalCA`(*url*, *protocol=None*, *options=None*, *ca_cert=None*)

Configuration for forwarding signing requests to an external certificate authority.

Parameters:

- **url** (*string*) – URL where certificate signing requests should be sent.
- **protocol** (*string*) – Protocol for communication with the external CA.
- **options** (*dict*) – An object with key/value pairs that are interpreted as protocol-specific options for the external CA driver.
- **ca_cert** (*string*) – The root CA certificate (in PEM format) this external CA uses to issue TLS certificates (assumed to be to the current swarm root CA certificate if not provided).

class `SwarmSpec`(**args*, ***kwargs*)

Describe a Swarm's configuration and options. Use `create_swarm_spec()` to instantiate.

class `TaskTemplate`(*container_spec*, *resources=None*, *restart_policy=None*, *placement=None*, *log_driver=None*, *networks=None*, *force_update=None*)

Describe the task specification to be used when creating or updating a service.

Parameters:

- **container_spec** (*ContainerSpec*) – Container settings for containers started as part of this task.
- **log_driver** (*DriverConfig*) – Log configuration for containers created as part of the service.
- **resources** (*Resources*) – Resource requirements which apply to each individual container created as part of the service.
- **restart_policy** (*RestartPolicy*) – Specification for the restart policy which applies to containers created as part of this service.
- **placement** (*Placement*) – Placement instructions for the scheduler. If a list is passed instead, it is assumed to be a list of constraints as part of a `Placement` object.
- **networks** (`list`) – List of network names or IDs or `NetworkAttachmentConfig` to attach the service to.
- **force_update** (*int*) – A counter that triggers an update even if no relevant parameters have been changed.

class `Ulimit`(***kwargs*)

Create a ulimit declaration to be used with `create_host_config()`.

Parameters:

- **name** (*str*) – Which ulimit will this apply to. The valid names can be found in '/etc/security/limits.conf' on a gnu/linux system.
- **soft** (*int*) – The soft limit for this ulimit. Optional.
- **hard** (*int*) – The hard limit for this ulimit. Optional.

Example

```
>>> nproc_limit = docker.types.Ulimit(name='nproc', soft=1024)
>>> hc = client.create_host_config(ulimits=[nproc_limit])
>>> container = client.create_container(
        'busybox', 'true', host_config=hc
    )
>>> client.inspect_container(container)['HostConfig']['Ulimits']
[{'Name': 'nproc', 'Hard': 0, 'Soft': 1024}]
```

class **UpdateConfig**(*parallelism=0, delay=None, failure_action='continue', monitor=None, max_failure_ratio=None, order=None*)

Used to specify the way container updates should be performed by a service.

Parameters:

- **parallelism** (*int*) – Maximum number of tasks to be updated in one iteration (0 means unlimited parallelism). Default: 0.
- **delay** (*int*) – Amount of time between updates, in nanoseconds.
- **failure_action** (*string*) – Action to take if an updated task fails to run, or stops running during the update. Acceptable values are `continue`, `pause`, as well as `rollback` since API v1.28. Default: `continue`
- **monitor** (*int*) – Amount of time to monitor each updated task for failures, in nanoseconds.
- **max_failure_ratio** (*float*) – The fraction of tasks that may fail during an update before the failure action is invoked, specified as a floating point number between 0 and 1. Default: 0
- **order** (*string*) – Specifies the order of operations when rolling out an updated task. Either `start-first` or `stop-first` are accepted.

Using TLS

Both the main DockerClient and low-level APIClient can connect to the Docker daemon with TLS.

This is all configured automatically for you if you're using from_env(), but if you need some extra control it is possible to configure it manually by using a TLSConfig object.

Examples

For example, to check the server against a specific CA certificate:

tls_config = docker.tls.TLSConfig(ca_cert='/path/to/ca.pem', verify=**True**)

client = docker.DockerClient(base_url='<https_url>', tls=tls_config)

This is the equivalent of docker --tlsverify --tlscacert /path/to/ca.pem ....

To authenticate with client certs:

tls_config = docker.tls.TLSConfig(

  client_cert=('/path/to/client-cert.pem', '/path/to/client-key.pem')

)

client = docker.DockerClient(base_url='<https_url>', tls=tls_config)

This is the equivalent of docker --tls --tlscert /path/to/client-cert.pem --tlskey /path/to/client-key.pem ....

Reference

class **TLSConfig**

TLS configuration.

Parameters:

- **client_cert** (*tuple of str*) – Path to client cert, path to client key.
- **ca_cert** (*str*) – Path to CA cert file.
- **verify** (*bool or str*) – This can be a bool or a path to a CA cert file to verify against. If True, verify using ca_cert; if False or not specified, do not verify.

Changelog¶

7.1.0

Upgrade Notes

- Bumped minimum engine API version to 1.24
- Bumped default engine API version to 1.44 (Moby 25.0)

Bugfixes

- Fixed issue with tag parsing when the registry address includes ports that resulted in invalid tag format errors
- Fixed issue preventing creating new configs (ConfigCollection), which failed with a KeyError due to the name field
- Fixed an issue due to an update in the requests package breaking docker-py by applying the suggested fix

Miscellaneous

- Documentation improvements
- Updated Ruff (linter) and fixed minor linting issues
- Packaging/CI updates
    - Started using hatch for packaging (https://github.com/pypa/hatch)
    - Updated setup-python github action
- Updated tests
    - Stopped checking for deprecated container and image related fields (Container and ContainerConfig)
    - Updated tests that check NetworkSettings.Networks.<network>.Aliases due to engine changes

7.0.0

Upgrade Notes

- Removed SSL version (ssl_version) and explicit hostname check (assert_hostname) options
    - assert_hostname has not been used since Python 3.6 and was removed in 3.12
    - Python 3.7+ supports TLSv1.3 by default
- Websocket support is no longer included by default
    - Use pip install docker[websockets] to include websocket-client dependency
    - By default, docker-py hijacks the TCP connection and does not use Websockets
    - Websocket client is only required to use attach_socket(container, ws=True)
- Python 3.7 no longer officially supported (reached end-of-life June 2023)

Features

- Python 3.12 support
- Full networking_config support for containers.create()
    - Replaces network_driver_opt (added in 6.1.0)
- Add health() property to container that returns status (e.g. unhealthy)
- Add pause option to container.commit()
- Add support for bind mount propagation (e.g. rshared, private)
- Add filters, keep_storage, and all parameters to prune_builds() (requires API v1.39+)

Bugfixes

- Consistently return docker.errors.NotFound on 404 responses
- Validate tag format before image push

Miscellaneous

- Upgraded urllib3 version in requirements.txt (used for development/tests)
- Documentation typo fixes & formatting improvements
- Fixed integration test compatibility for newer Moby engine versions
- Switch to ruff for linting

6.1.3

Bugfixes

- Fix compatibility with **eventlet/eventlet**

6.1.2

Bugfixes

- Fix for socket timeouts on long docker exec calls

6.1.1

Bugfixes

- Fix containers.stats() hanging with stream=True
- Correct return type in docs for containers.diff() method

6.1.0

Upgrade Notes

- Errors are no longer returned during client initialization if the credential helper cannot be found. A warning will be emitted instead, and an error is returned if the credential helper is used.

Features

- Python 3.11 support
- Use poll() instead of select() on non-Windows platforms
- New API fields
    - network_driver_opt on container run / create
    - one-shot on container stats
    - status on services list

Bugfixes

- Support for requests 2.29.0+ and urllib3 2.x
- Do not strip characters from volume names
- Fix connection leak on container.exec_* operations
- Fix errors closing named pipes on Windows

6.0.1

Bugfixes

- Fix for The pipe has been ended errors on Windows
- Support floats for container log filtering by timestamp (since / until)

6.0.0

Upgrade Notes

- Minimum supported Python version is 3.7+
- When installing with pip, the docker[tls] extra is deprecated and a no-op, use docker for same functionality (TLS support is always available now)
- Native Python SSH client (used by default / use_ssh_client=False) will now reject unknown host keys with paramiko.ssh_exception.SSHException
- Short IDs are now 12 characters instead of 10 characters (same as Docker CLI)

Features

- Python 3.10 support
- Automatically negotiate most secure TLS version
- Add platform (e.g. linux/amd64, darwin/arm64) to container create & run
- Add support for GlobalJob and ReplicatedJobs for Swarm
- Add remove() method on Image
- Add force param to disable() on Plugin

Bugfixes

- Fix install issues on Windows related to pywin32
- Do not accept unknown SSH host keys in native Python SSH mode
- Use 12 character short IDs for consistency with Docker CLI
- Ignore trailing whitespace in .dockerignore files
- Fix IPv6 host parsing when explicit port specified
- Fix ProxyCommand option for SSH connections
- Do not spawn extra subshell when launching external SSH client
- Improve exception semantics to preserve context
- Documentation improvements (formatting, examples, typos, missing params)

Miscellaneous

- Upgrade dependencies in requirements.txt to latest versions
- Remove extraneous transitive dependencies
- Eliminate usages of deprecated functions/methods

- Test suite reliability improvements
- GitHub Actions workflows for linting, unit tests, integration tests, and publishing releases

5.0.3

List of PRs / issues for this release

Features

- Add cap_add and cap_drop parameters to service create and ContainerSpec
- Add templating parameter to config create

Bugfixes

- Fix getting a read timeout for logs/attach with a tty and slow output

Miscellaneous

- Fix documentation examples

5.0.2

List of PRs / issues for this release

Bugfixes

- Fix disable_buffering regression

5.0.1

List of PRs / issues for this release

Bugfixes

- Bring back support for ssh identity file
- Cleanup remaining python-2 dependencies
- Fix image save example in docs

Miscellaneous

- Bump urllib3 to 1.26.5
- Bump requests to 2.26.0

5.0.0

List of PRs / issues for this release

Breaking changes

- Remove support for Python 2.7
- Make Python 3.6 the minimum version supported

Features

- Add limit parameter to image search endpoint

Bugfixes

- Fix KeyError exception on secret create
- Verify TLS keys loaded from docker contexts
- Update PORT_SPEC regex to allow square brackets for IPv6 addresses
- Fix containers and images documentation examples

4.4.4

List of PRs / issues for this release

Bugfixes

- Remove LD_LIBRARY_PATH and SSL_CERT_FILE environment variables when shelling out to the ssh client

4.4.3

List of PRs / issues for this release

Features

- Add support for docker.types.Placement.MaxReplicas

Bugfixes

- Fix SSH port parsing when shelling out to the ssh client

4.4.2

List of PRs / issues for this release

Bugfixes

- Fix SSH connection bug where the hostname was incorrectly trimmed and the error was hidden
- Fix docs example

Miscellaneous

- Add Python3.8 and 3.9 in setup.py classifier list

4.4.1

List of PRs / issues for this release

Bugfixes

- Avoid setting unsuported parameter for subprocess.Popen on Windows
- Replace use of deprecated “filter” argument on “”docker/api/image”

4.4.0

List of PRs / issues for this release

Features

- Add an alternative SSH connection to the paramiko one, based on shelling out to the SSh client. Similar to the behaviour of Docker cli
- Default image tag to latest on pull

Bugfixes

- Fix plugin model upgrade
- Fix examples URL in ulimits

Miscellaneous

- Improve exception messages for server and client errors
- Bump cryptography from 2.3 to 3.2

4.3.1

List of PRs / issues for this release

Miscellaneous

- Set default API version to auto
- Fix conversion to bytes for float
- Support OpenSSH identityfile option

4.3.0

List of PRs / issues for this release

Features

- Add DeviceRequest type to expose host resources such as GPUs
- Add support for DriverOpts in EndpointConfig
- Disable compression by default when using container.get_archive method

Miscellaneous

- Update default API version to v1.39
- Update test engine version to 19.03.12

4.2.2

List of PRs / issues for this release

Bugfixes

- Fix context load for non-docker endpoints

4.2.1

List of PRs / issues for this release

Features

- Add option on when to use tls on Context constructor
- Make context orchestrator field optional

4.2.0

[List of PRs / issues for this release]()

Bugfixes

- Fix win32pipe.WaitNamedPipe throw exception in Windows containers
- Use Hostname, Username, Port and ProxyCommand settings from .ssh/config when on SSH
- Set host key policy for ssh transport to paramiko.WarningPolicy()
- Set logging level of paramiko to warn

Features

- Add support for docker contexts through docker.ContextAPI

4.1.0

[List of PRs / issues for this release]()

Bugfixes

- Correct INDEX_URL logic in build.py _set_auth_headers
- Fix for empty auth keys in config.json

Features

- Add NetworkAttachmentConfig for service create/update

Miscellaneous

- Bump pytest to 4.3.1
- Adjust --platform tests for changes in docker engine
- Update credentials-helpers to v0.6.3

4.0.2

[List of PRs / issues for this release]()

Bugfixes

- Unified the way HealthCheck is created/configured

Miscellaneous

- Bumped version of websocket-client

4.0.1

[List of PRs / issues for this release]()

Bugfixes

- Fixed an obsolete import in the credentials subpackage that caused import errors in Python 3.7

Miscellaneous

- Docs building has been repaired

4.0.0

List of PRs / issues for this release

Breaking changes

- Support for Python 3.3 and Python 3.4 has been dropped
- APIClient.update_service, APIClient.init_swarm, and DockerClient.swarm.init now return a dict from the API's response body
- In APIClient.build and DockerClient.images.build, the use_config_proxy parameter now defaults to True
- init_path is no longer a valid parameter for HostConfig

Features

- It is now possible to provide SCTP ports for port mappings
- ContainerSpecs now support the init parameter
- DockerClient.swarm.init and APIClient.init_swarm now support the data_path_addr parameter
- APIClient.update_swarm and DockerClient.swarm.update now support the rotate_manager_unlock_key parameter
- APIClient.update_service returns the API's response body as a dict
- APIClient.init_swarm, and DockerClient.swarm.init now return the API's response body as a dict

Bugfixes

- Fixed PlacementPreference instances to produce a valid API type
- Fixed a bug where not setting a value for buildargs in build could cause the library to attempt accessing attributes of a None value
- Fixed a bug where setting the volume_driver parameter in DockerClient.containers.create would result in an error
- APIClient.inspect_distribution now correctly sets the authentication headers on the request, allowing it to be used with private repositories This change also applies to DockerClient.get_registry_data

3.7.2

List of PRs / issues for this release

Bugfixes

- Fix base_url to keep TCP protocol on utils.py by letting the responsibility of changing the protocol to parse_host afterwards, letting base_url with the original value.
- XFAIL test_attach_stream_and_cancel on TLS

3.7.1

List of PRs / issues for this release

Bugfixes

- Set a different default number (which is now 9) for SSH pools
- Adds a BaseHTTPAdapter with a close method to ensure that the pools is clean on close()
- Makes SSHHTTPAdapter reopen a closed connection when needed like the others

3.7.0

List of PRs / issues for this release

Features

- Added support for multiplexed streams (for attach and exec_start). Learn more at https://docker-py.readthedocs.io/en/stable/user_guides/multiplex.html
- Added the use_config_proxy parameter to the following methods: APIClient.build, APIClient.create_container, DockerClient.images.build and DockerClient.containers.run (False by default). **This parameter will become True by default in the 4.0.0 release.**
- Placement preferences for Swarm services are better validated on the client and documentation has been updated accordingly

Bugfixes

- Fixed a bug where credential stores weren't queried for relevant registry credentials with certain variations of the config.json file.
- DockerClient.swarm.init now returns a boolean value as advertised.

3.6.0

List of PRs / issues for this release

Features

- Added support for connecting to the Docker Engine over SSH. Additional dependencies for this feature can be installed with pip install "docker[ssh]"
- Added support for the named parameter in Image.save, which may be used to ensure the resulting tarball retains the image's name on save.

Bugfixes

- Fixed a bug where builds on Windows with a context path using the \\?\ prefix would fail with some relative Dockerfile paths.
- Fixed an issue where pulls made with the DockerClient would fail when setting the stream parameter to True.

Miscellaneous

- The minimum requirement for the requests dependency has been bumped to 2.20.0

3.5.1

List of PRs / issues for this release

Miscellaneous

- Bumped version of pyOpenSSL in requirements.txt and setup.py to prevent installation of a vulnerable version
- Docs fixes

3.5.0

List of PRs / issues for this release

Deprecation warning

- Support for Python 3.3 will be dropped in the 4.0.0 release

Features

- Updated dependencies to ensure support for Python 3.7 environments
- Added support for the uts_mode parameter in HostConfig
- The UpdateConfig constructor now allows rollback as a valid value for failure_action
- Added support for rollback_config in APIClient.create_service, APIClient.update_service, DockerClient.services.create and Service.update.

Bugfixes

- Credential helpers are now properly leveraged by the build method
- Fixed a bug that caused placement preferences to be ignored when provided to DockerClient.services.create
- Fixed a bug that caused a user value of 0 to be ignored in APIClient.create_container and DockerClient.containers.create

3.4.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug that caused auth values in config files written using one of the legacy formats to be ignored

- Fixed issues with handling of double-wildcard ** patterns in .dockerignore files

3.4.0

List of PRs / issues for this release

Features

- The APIClient and DockerClient constructors now accept a credstore_env parameter. When set, values in this dictionary are added to the environment when executing the credential store process.

Bugfixes

- DockerClient.networks.prune now properly returns the operation's result
- Fixed a bug that caused custom Dockerfile paths in a subfolder of the build context to be invalidated, preventing these builds from working
- The plugin_privileges method can now be called for plugins requiring authentication to access
- Fixed a bug that caused attempts to read a data stream over an unsecured TCP socket to crash on Windows clients
- Fixed a bug where using the read_only parameter when creating a service using the DockerClient was being ignored
- Fixed an issue where Service.scale would not properly update the service's mode, causing the operation to fail silently

3.3.0

List of PRs / issues for this release

Features

- Added support for prune_builds in APIClient and DockerClient.images
- Added support for ignore_removed parameter in DockerClient.containers.list

Bugfixes

- Fixed an issue that caused builds to fail when an in-context Dockerfile would be specified using its absolute path
- Installation with pip 10.0.0 and above no longer fails
- Connection timeout for stop and restart now gets properly adjusted to allow for the operation to finish in the specified time
- Improved docker credential store support on Windows

3.2.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug with builds not properly identifying Dockerfile paths relative to the build context
- Fixed an issue where builds would raise a ValueError when attempting to build with a Dockerfile on a different Windows drive.

3.2.0

List of PRs / issues for this release

Features

- Generators returned by attach(), logs() and events() now have a cancel() method to let consumers stop the iteration client-side.
- build() methods can now handle Dockerfiles supplied outside of the build context.
- Added sparse argument to DockerClient.containers.list()
- Added isolation parameter to build() methods.
- Added close() method to DockerClient
- Added APIClient.inspect_distribution() method and DockerClient.images.get_registry_data()
    - The latter returns an instance of the new RegistryData class

3.1.4

List of PRs / issues for this release

Bugfixes

- Fixed a bug where build contexts containing directory symlinks would produce invalid tar archives

3.1.3

Bugfixes

- Regenerated invalid wheel package

3.1.2

List of PRs / issues for this release

Bugfixes

- Fixed a bug that led to a Dockerfile not being included in the build context in some situations when the Dockerfile's path was prefixed with ./

3.1.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug that caused costly DNS lookups on Mac OSX when connecting to the engine through UNIX socket
- Fixed a bug that caused .dockerignore comments to be read as exclusion patterns

3.1.0

List of PRs / issues for this release

Features

- Added support for device_cgroup_rules in host config
- Added support for generic_resources when creating a Resources object.
- Added support for a configurable chunk_size parameter in export, get_archive and get_image (Image.save)
- Added a force_update method to the Service class.
- In Service.update, when the force_update parameter is set to True, the current force_update counter is incremented by one in the update request.

Bugfixes

- Fixed a bug where authentication through login() was being ignored if the SDK was configured to use a credential store.
- Fixed a bug where download methods would use an absurdly small chunk size, leading to slow data retrieval
- Fixed a bug where using DockerClient.images.pull to pull an image by digest would lead to an exception being raised.
- .dockerignore rules should now be respected as defined by the spec, including respect for last-line precedence and proper handling of absolute paths
- The pass credential store is now properly supported.

3.0.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where APIClient.login didn't populate the _auth_configs dictionary properly, causing subsequent pull and push operations to fail
- Fixed a bug where some build context files were incorrectly recognized as being inaccessible.
- Fixed a bug where files with a negative mtime value would cause errors when included in a build context

3.0.0

List of PRs / issues for this release

Breaking changes

- Support for API version < 1.21 has been removed.
- The following methods have been removed:
    - APIClient.copy has been removed. Users should use APIClient.get_archive instead.

    - APIClient.insert has been removed. Users may use APIClient.put_archive combined with APIClient.commit to replicate the method’s behavior.
    - utils.ping_registry and utils.ping have been removed.
- The following parameters have been removed:
    - stream in APIClient.build
    - cpu_shares, cpuset, dns, mem_limit, memswap_limit, volume_driver, volumes_from in APIClient.create_container. These are all replaced by their equivalent in create_host_config
    - insecure_registry in APIClient.login, APIClient.pull, APIClient.push, DockerClient.images.push and DockerClient.images.pull
    - viz in APIClient.images
- The following parameters have been renamed:
    - endpoint_config in APIClient.create_service and APIClient.update_service is now endpoint_spec
    - name in DockerClient.images.pull is now repository
- The return value for the following methods has changed:
    - APIClient.wait and Container.wait now return a dict representing the API’s response instead of returning the status code directly.
    - DockerClient.images.load now returns a list of Image objects that have for the images that were loaded, instead of a log stream.
    - Container.exec_run now returns a tuple of (exit_code, output) instead of just the output.
    - DockerClient.images.build now returns a tuple of (image, build_logs) instead of just the image object.
    - APIClient.export, APIClient.get_archive and APIClient.get_image now return generators streaming the raw binary data from the server’s response.
    - When no tag is provided, DockerClient.images.pull now returns a list of Images associated to the pulled repository instead of just the latest image.

Features

- The Docker Python SDK is now officially supported on Python 3.6
- Added scale method to the Service model ; this method is a shorthand that calls update_service with the required number of replicas
- Added support for the platform parameter in APIClient.build, DockerClient.images.build, APIClient.pull and DockerClient.images.pull
- Added support for the until parameter in APIClient.logs and Container.logs
- Added support for the workdir argument in APIClient.exec_create and Container.exec_run

- Added support for the condition argument in APIClient.wait and Container.wait
- Users can now specify a publish mode for ports in EndpointSpec using the {published_port: (target_port, protocol, publish_mode)} syntax.
- Added support for the isolation parameter in ContainerSpec, DockerClient.services.create and Service.update
- APIClient.attach_socket, APIClient.exec_create now allow specifying a detach_keys combination. If unspecified, the value from the config.json file will be used
- TLS connections now default to using the TLSv1.2 protocol when available

Bugfixes

- Fixed a bug where whitespace-only lines in .dockerignore would break builds on Windows
- Fixed a bug where broken symlinks inside a build context would cause the build to fail
- Fixed a bug where specifying volumes with Windows drives would cause incorrect parsing in DockerClient.containers.run
- Fixed a bug where the networks data provided to create_service and update_service would be sent incorrectly to the Engine with API < 1.25
- Pulling all tags from a repository with no latest tag using the DockerClient will no longer raise a NotFound exception

2.7.0

List of PRs / issues for this release

Features

- Added unlock_swarm and get_unlock_key methods to the APIClient.
    - Added unlock and get_unlock_key to DockerClient.swarm.
- Added a greedy parameter to DockerClient.networks.list, yielding additional details about the listed networks.
- Added cpu_rt_runtime and cpu_rt_period as parameters to APIClient.create_host_config and DockerClient.containers.run.
- Added the order argument to UpdateConfig.
- Added fetch_current_spec to APIClient.update_service and Service.update that will retrieve the current configuration of the service and merge it with the provided parameters to determine the new configuration.

Bugfixes

- Fixed a bug where the build method tried to include inaccessible files in the context, leading to obscure errors during the build phase (inaccessible files inside the context now raise an IOError instead).
- Fixed a bug where the build method would try to read from FIFOs present inside the build context, causing it to hang.

- APIClient.stop will no longer override the stop_timeout value present in the container's configuration.
- Fixed a bug preventing removal of networks with names containing a space.
- Fixed a bug where DockerClient.containers.run would crash if the auto_remove parameter was set to True.
- Changed the default value of listen_addr in join_swarm to match the one in init_swarm.
- Fixed a bug where handling HTTP errors with no body would cause an unexpected exception to be thrown while generating an APIError object.

2.6.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug on Python 3 installations preventing the use of the attach and exec_run methods.

2.6.0

List of PRs / issues for this release

Features

- Added support for mounts in APIClient.create_host_config and DockerClient.containers.run
- Added support for consistency, tmpfs_size and tmpfs_mode when creating mount objects.
- Mount objects now support the tmpfs and npipe types.
- Added support for extra_hosts in the build methods.
- Added support for the configs API:
    - In APIClient: create_config, inspect_config, remove_config, configs
    - In DockerClient: configs.create, configs.get, configs.list and the Config model.
    - Added configs parameter to ContainerSpec. Each item in the configs list must be a docker.types.ConfigReference instance.
- Added support for the following parameters when creating a ContainerSpec object: groups, open_stdin, read_only, stop_signal, helathcheck, hosts, ns_config, configs, privileges.
- Added the following configuration classes to docker.types: ConfigReference, DNSConfig, Privileges, SwarmExternalCA.
- Added support for driver in APIClient.create_secret and DockerClient.secrets.create.
- Added support for scope in APIClient.inspect_network and APIClient.create_network, and their DockerClient equivalent.

- Added support for the following parameters to create_swarm_spec: external_cas, labels, signing_ca_cert, signing_ca_key, ca_force_rotate, autolock_managers, log_driver. These additions also apply to DockerClient.swarm.init.
- Added support for insert_defaults in APIClient.inspect_service and DockerClient.services.get.

Bugfixes

- Fixed a bug where reading a 0-length frame in log streams would incorrectly interrupt streaming.
- Fixed a bug where the id member on Swarm objects wasn't being populated.
- Fixed a bug that would cause some data at the beginning of an upgraded connection stream (attach, exec_run) to disappear.

2.5.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where patterns ending with ** in .dockerignore would raise an exception
- Fixed a bug where using attach with the stream argument set to False would raise an exception

2.5.0

List of PRs / issues for this release

Features

- Added support for the squash parameter in APIClient.build and DockerClient.images.build.
- When using API version 1.23 or above, load_image will now return a generator of progress as JSON dicts.
- remove_image now returns the content of the API's response.

Bugfixes

- Fixed an issue where the auto_remove parameter in DockerClient.containers.run was not taken into account.
- Fixed a bug where .dockerignore patterns starting with a slash were ignored.
- Fixed an issue with the handling of ** patterns in .dockerignore
- Fixed a bug where building FROM a private Docker Hub image when not using a cred store would fail.
- Fixed a bug where calling create_service or update_service with task_template as a dict would raise an exception.
- Fixed the handling of TTY-enabled containers in attach and exec_run.

- DockerClient.containers.run will no longer attempt to stream logs if the log driver doesn't support the operation.

Miscellaneous

- Added extra requirements for better TLS support on some platforms. These can be installed or required through the docker[tls] notation.

2.4.2

[List of PRs / issues for this release]()

Bugfixes

- Fixed a bug where the split_port utility would raise an exception when passed a non-string argument.

2.4.0

[List of PRs / issues for this release]()

Features

- Added support for the target and network_mode parameters in APIClient.build and DockerClient.images.build.
- Added support for the runtime parameter in APIClient.create_container and DockerClient.containers.run.
- Added support for the ingress parameter in APIClient.create_network and DockerClient.networks.create.
- Added support for placement configuration in docker.types.TaskTemplate.
- Added support for tty configuration in docker.types.ContainerSpec.
- Added support for start_period configuration in docker.types.Healthcheck.
- The credHelpers section in Docker's configuration file is now recognized.
- Port specifications including IPv6 endpoints are now supported.

Bugfixes

- Fixed a bug where instantiating a DockerClient using docker.from_env wouldn't correctly set the default timeout value.
- Fixed a bug where DockerClient.secrets was not accessible as a property.
- Fixed a bug where DockerClient.build would sometimes return the wrong image.
- Fixed a bug where values for HostConfig.nano_cpus exceeding 2^32 would raise a type error.
- Image.tag now properly returns True when the operation is successful.
- APIClient.logs and Container.logs now raise an exception if the since argument uses an unsupported type instead of ignoring the value.

- Fixed a bug where some methods would raise a NullResource exception when the resource ID was provided using a keyword argument.

Miscellaneous

- APIClient instances can now be pickled.

2.3.0

[List of PRs / issues for this release]

Features

- Added support for the following HostConfig parameters: volume_driver, cpu_count, cpu_percent, nano_cpus, cpuset_mems.
- Added support for verbose parameter in APIClient.inspect_network and DockerClient.networks.get.
- Added support for the environment parameter in APIClient.exec_create and Container.exec_run
- Added reload_config method to APIClient, that lets the user reload the config.json data from disk.
- Added labels property to the Image and Container classes.
- Added image property to the Container class.

Bugfixes

- Fixed a bug where setting replicas to zero in ServiceMode would not register as a valid entry.
- Fixed a bug where DockerClient.images.build would report a failure after a successful build if a tag was set.
- Fixed an issue where DockerClient.images.pull would fail to return the corresponding image object if a tag was set.
- Fixed a bug where a list of mounts provided to APIClient.create_service would sometimes be parsed incorrectly.
- Fixed a bug where calling Network.containers would crash when no containers were associated with the network.
- Fixed an issue where Network.connect and Network.disconnect would not accept some of the documented parameters.
- Fixed a bug where the cpuset_cpus parameter would not be properly set in APIClient.create_host_config.

Miscellaneous

- The invalid networks argument in DockerClient.containers.run has been replaced with a (working) singular network argument.

2.2.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where the status_code attribute of APIError exceptions would not reflect the expected value.
- Fixed an issue where the events method would time out unexpectedly if no data was sent by the engine for a given amount of time.

2.2.0

List of PRs / issues for this release

Features

- Default API version has been bumped to 1.26 (Engine 1.13.1+)
- Upgrade plugin:
    - Added the upgrade_plugin method to the APIClient class
    - Added the upgrade method to the Plugin class
- Service logs:
    - Added the service_logs method to the APIClient class
    - Added the logs method to the Service class
- Added the df method to APIClient and DockerClient
- Added support for init and init_path parameters in HostConfig and DockerClient.containers.run
- Added support for hostname parameter in ContainerSpec and DockerClient.service.create
- Added support for port range to single port in port mappings (e.g. 8000-8010:80)

Bugfixes

- Fixed a bug where a missing container port in a port mapping would raise an unexpected TypeError
- Fixed a bug where the events method in APIClient and DockerClient would not respect custom headers set in config.json

2.1.0

List of PRs / issues for this release

Features

- Added the following pruning methods:
    - In APIClient: prune_containers, prune_images, prune_networks, prune_volumes
    - In DockerClient: containers.prune, images.prune, networks.prune, volumes.prune
- Added support for the plugins API:

    - In APIClient: configure_plugin, create_plugin, disable_plugin, enable_plugin, inspect_plugin, pull_plugin, plugins, plugin_privileges, push_plugin, remove_plugin
    - In DockerClient: plugins.create, plugins.get, plugins.install, plugins.list, and the Plugin model.
- Added support for the secrets API:
    - In APIClient: create_secret, inspect_secret, remove_secret, secrets
    - In DockerClient: secret.create, secret.get, secret.list and the Secret model.
    - Added secrets parameter to ContainerSpec. Each item in the secrets list must be a docker.types.SecretReference instance.
- Added support for cache_from in APIClient.build and DockerClient.images.build.
- Added support for auto_remove and storage_opt in APIClient.create_host_config and DockerClient.containers.run
- Added support for stop_timeout in APIClient.create_container and DockerClient.containers.run
- Added support for the force parameter in APIClient.remove_volume and Volume.remove
- Added support for max_failure_ratio and monitor in UpdateConfig
- Added support for force_update in TaskTemplate
- Made name parameter optional in APIClient.create_volume and DockerClient.volumes.create

Bugfixes

- Fixed a bug where building from a directory containing socket-type files would raise an unexpected AttributeError.
- Fixed an issue that was preventing the DockerClient.swarm.init method to take into account arguments passed to it.
- Image.tag now correctly returns a boolean value upon completion.
- Fixed several issues related to passing volumes in DockerClient.containers.run
- Fixed an issue where DockerClient.image.build wouldn’t return an Image object even when the build was successful

2.0.2

List of PRs / issues for this release

Bugfixes

- Installation of the package now fails if the docker-py package is installed in order to prevent obscure naming conflicts when both packages co-exist.
- Added missing filters parameter to APIClient.networks.

- Resource objects generated by the DockerClient are now hashable.
- Fixed a bug where retrieving untagged images using DockerClient would raise a TypeError exception.
- mode parameter in create_service is now properly converted to a valid data type for the Engine API. Use ServiceMode for advanced configurations.
- Fixed a bug where the decoded APIClient.events stream would sometimes raise an exception when a container is stopped or restarted.

2.0.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where forward slashes in some .dockerignore patterns weren't being parsed correctly on Windows
- Fixed a bug where Mount.parse_mount_string would never set the read_only parameter on the resulting Mount.
- Fixed a bug where Mount.parse_mount_string would incorrectly mark host binds as being of volume type.

2.0.0

List of PRs / issues for this release

Breaking changes

- Dropped support for Python 2.6
- docker.Client has been renamed to docker.APIClient
- docker.from_env now creates a DockerClient instance instead of an APIClient instance.
- Removed HostConfig parameters from APIClient.start
- The minimum supported API version is now 1.21 (Engine version 1.9.0+)
- The name of the pip package is now docker (was: docker-py). New versions of this library will only be published as docker from now on.
- docker.ssladapter is now docker.transport.ssladapter
- The package structure has been flattened in certain cases, which may affect import for docker.auth and docker.utils.ports
- docker.utils.types has been moved to docker.types
- create_host_config, create_ipam_pool and create_ipam_config have been removed from docker.utils. They have been replaced by the following classes in docker.types: HostConfig, IPAMPool and IPAMCOnfig.

Features

- Added a high-level, user-focused API as docker.DockerClient. See the README and documentation for more information.
- Implemented update_node method in APIClient.
- Implemented remove_node method in APIClient.
- Added support for restart_policy in update_container.
- Added support for labels and shmsize in build.
- Added support for attachable in create_network
- Added support for healthcheck in create_container.
- Added support for isolation in HostConfig.
- Expanded support for pid_mode in HostConfig (now supports arbitrary values for API version >= 1.24).
- Added support for options in IPAMConfig
- Added a HealthCheck class to docker.types to be used in create_container.
- Added an EndpointSpec class to docker.types to be used in create_service and update_service.

Bugfixes

- Fixed a bug where auth information would not be properly passed to the engine during a build if the client used a credentials store.
- Fixed an issue with some exclusion patterns in build.
- Fixed an issue where context files were bundled with the wrong permissions when calling build on Windows.
- Fixed an issue where auth info would not be retrieved from its default location on Windows.
- Fixed an issue where lists of networks in create_service and update_service wouldn't be properly converted for the engine.
- Fixed an issue where endpoint_config in create_service and update_service would be ignored.
- endpoint_config in create_service and update_service has been deprecated in favor of endpoint_spec
- Fixed a bug where constraints in a TaskTemplate object wouldn't be properly converted for the engine.
- Fixed an issue where providing a dictionary for env in ContainerSpec would provoke an APIError when sent to the engine.
- Fixed a bug where providing an env_file containing empty lines in create_containerwould raise an exception.
- Fixed a bug where detach was being ignored by exec_start.

Documentation

- Documentation for classes and methods is now included alongside the code as docstrings.

1.10.6

List of PRs / issues for this release

Bugfixes

- Fixed an issue where setting a NpipeSocket instance to blocking mode would put it in non-blocking mode and vice-versa.

1.10.5

List of PRs / issues for this release

Bugfixes

- Fixed an issue where concurrent attempts to access to a named pipe by the client would sometimes cause recoverable exceptions to be raised.

1.10.4

List of PRs / issues for this release

Bugfixes

- Fixed an issue where RestartPolicy.condition_types.ON_FAILURE would yield an invalid value.
- Fixed an issue where the SSL connection adapter would receive an invalid argument.
- Fixed an issue that caused the Client to fail to reach API endpoints when the provided base_url had a trailing slash.
- Fixed a bug where some environment values in create_container containing unicode characters would raise an encoding error.
- Fixed a number of issues tied with named pipe transport on Windows.
- Fixed a bug where inclusion patterns in .dockerignore would cause some excluded files to appear in the build context on Windows.

Miscellaneous

- Adjusted version requirements for the requests library.
- It is now possible to run the docker-py test suite on Windows.

1.10.3

List of PRs / issues for this release

Bugfixes

- Fixed an issue where identity tokens in configuration files weren't handled by the library.

Miscellaneous

- Increased the default number of connection pools from 10 to 25. This number can now be configured using the num_pools parameter in the Client constructor.

1.10.2

List of PRs / issues for this release

Bugfixes

- Updated the docker-pycreds dependency as it was causing issues for some users with dependency resolution in applications using docker-py.

1.10.1

List of PRs / issues for this release

Bugfixes

- The docker.utils.types module was removed in favor of docker.types, but some applications imported it explicitly. It has been re-added with an import warning advising to use the new module path.

1.10.0

List of PRs / issues for this release

Features

- Added swarm mode and service management methods. See the documentation for details.
- Added support for IPv6 Docker host addresses in the Client constructor.
- Added (read-only) support for the Docker credentials store.
- Added support for custom auth_config in Client.push.
- Added support for labels in Client.create_volume.
- Added support for labels and enable_ipv6 in Client.create_network.
- Added support for force param in Client.disconnect_container_from_network.
- Added support for pids_limit, sysctls, userns_mode, cpuset_cpus, cpu_shares, mem_reservation and kernel_memory parameters in Client.create_host_config.
- Added support for link_local_ips in create_endpoint_config.
- Added support for a changes parameter in Client.import_image.
- Added support for a version parameter in Client.from_env.

Bugfixes

- Fixed a bug where Client.build would crash if the config.json file contained a HttpHeaders entry.
- Fixed a bug where passing decode=True in some streaming methods would crash when the daemon's response had an unexpected format.

- Fixed a bug where environment values with unicode characters weren't handled properly in create_container.
- Fixed a bug where using the npipe protocol would sometimes break with ValueError: buffer size must be strictly positive.

Miscellaneous

- Fixed an issue where URL-quoting in docker-py was inconsistent with the quoting done by the Docker CLI client.
- The client now sends TCP upgrade headers to hint potential proxies about connection hijacking.
- The client now defaults to using the npipe protocol on Windows.

1.9.0

List of PRs / issues for this release

Features

- Added **experimental** support for Windows named pipes (npipe:// protocol).
- Added support for Block IO constraints in Client.create_host_config. This includes parameters blkio_weight, blkio_weight_device, device_read_bps, device_write_bps, device_read_iops and device_write_iops.
- Added support for the internal param in Client.create_network.
- Added support for ipv4_address and ipv6_address in utils function create_endpoint_config.
- Added support for custom user agent setting in the Client constructor. By default, docker-py now also declares itself in the User-Agent header.

Bugfixes

- Fixed an issue where the HTTP timeout on streaming responses would sometimes be set incorrectly.
- Fixed an issue where explicit relative paths in .dockerignore files were not being recognized.

1.8.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where calling login() against the default registry would fail with the 1.10.x engine
- Fixed a bug where values in environment files would be parsed incorrectly if they contained an equal sign.
- Switched to a better supported backport of the match_hostname function, fixing dependency issues in some environments.

1.8.0

[List of PRs / issues for this release]

Features

- Added Client.update_container method (Update resource configs of a container)
- Added support for gzipped context in Client.build
- Added ability to specify IP address when connecting a container to a network
- Added tmpfs support to Client.create_host_config
- Added support for the changes param in Client.commit
- Added support for the follow param in Client.logs
- Added support for the check_duplicate param in Client.create_network
- Added support for the decode param in Client.push and Client.pull
- Added docker.from_env shortcut function. Instantiates a client with kwargs_from_env
- kwargs_from_env now supports an optional environment parameter. If present, values will be fetched from this dictionary instead of os.environ

Bugfixes

- Fixed a bug where TLS verification would fail when using IP addresses in the certificate's subjectAltName fields
- Fixed an issue where the default TLS version in TLSConfig would break in some environments. docker-py now uses TLSv1 by default This setting can be overridden using the ssl_version param in kwargs_from_env or the TLSConfig constructor
- Fixed a bug where tcp hosts would fail to connect to TLS-enabled endpoints
- Fixed a bug where loading a valid docker configuration file would fail
- Fixed a bug where some environment variables specified through create_container would be improperly formatted
- Fixed a bug where using the unix socket connection would raise an error in some edge-case situations

Miscellaneous

- Default API version is now 1.22 (introduced in Docker 1.10.0)

1.7.2

[List of PRs / issues for this release]

Bugfixes

- Fixed a bug where TLS verification was improperly executed when providing a custom CA certificate.

1.7.1

List of PRs / issues for this release

Features

- Added support for shm_size in Client.create_host_config

Bugfixes

- Fixed a bug where Dockerfile would sometimes be excluded from the build context.
- Fixed a bug where a docker config file containing unknown keys would raise an exception.
- Fixed an issue with SSL connections behaving improperly when pyOpenSSL was installed in the same environment.
- Several TLS configuration improvements

1.7.0

List of PRs / issues for this release

Features

- Added support for cusom IPAM configuration in Client.create_network
- Added input support to Client.exec_create
- Added support for stop_signal in Client.create_host_config
- Added support for custom HTTP headers in Docker config file.
- Added support for unspecified transfer protocol in base_url when TLS is enabled.

Bugfixes

- Fixed a bug where the filters parameter in Client.volumes would not be applied properly.
- Fixed a bug where memory limits would parse to incorrect values.
- Fixed a bug where the devices parameter in Client.create_host_config would sometimes be misinterpreted.
- Fixed a bug where instantiating a Client object would sometimes crash if base_url was unspecified.
- Fixed a bug where an error message related to TLS configuration would link to a non-existent (outdated) docs page.

Miscellaneous

- Processing of .dockerignore has been made significantly faster.
- Dropped explicit support for Python 3.2

1.6.0

List of PRs / issues for this release

Features

- Added support for the since param in Client.logs (introduced in API version 1.19)
- Added support for the DOCKER_CONFIG environment variable when looking up auth config
- Added support for the stream param in Client.stats (when set to False, allows user to retrieve a single snapshot instead of a constant data stream)
- Added support for the mem_swappiness, oom_kill_disable params in Client.create_host_config
- Added support for build arguments in Client.build through the buildargs param.

Bugfixes

- Fixed a bug where streaming data over HTTPS would sometimes behave incorrectly with Python 3.x
- Fixed a bug where commands containing unicode characters would be incorrectly handled by Client.create_container.
- Fixed a bug where auth config credentials containing unicode characters would cause failures when pushing / pulling images.
- Setting tail=0 in Client.logs no longer shows past logs.
- Fixed a bug where Client.pull and Client.push couldn't handle image names containing a dot.

Miscellaneous

- Default API version is now 1.21 (introduced in Docker 1.9.0)
- Several test improvements and cleanup that should make the suite easier to expand and maintain moving forward.

1.5.0

List of PRs / issues for this release

Features

- Added support for the networking API introduced in Docker 1.9.0 (Client.networks, Client.create_network, Client.remove_network, Client.inspect_network, Client.connect_container_to_network, Client.disconnect_container_from_network).
- Added support for the volumes API introduced in Docker 1.9.0 (Client.volumes, Client.create_volume, Client.inspect_volume, Client.remove_volume).
- Added support for the group_add parameter in create_host_config.
- Added support for the CPU CFS (cpu_quota and cpu_period) parameters in create_host_config.
- Added support for the archive API endpoint (Client.get_archive, Client.put_archive).
- Added support for ps_args parameter in Client.top.

Bugfixes

- Fixed a bug where specifying volume binds with unicode characters would fail.
- Fixed a bug where providing an explicit protocol in Client.port would fail to yield the expected result.
- Fixed a bug where the priority protocol returned by Client.port would be UDP instead of the expected TCP.

Miscellaneous

- Broke up Client code into several files to facilitate maintenance and contribution.
- Added contributing guidelines to the repository.

1.4.0

List of PRs / issues for this release

Deprecation warning

- docker.utils.create_host_config is deprecated in favor of Client.create_host_config.

Features

- Added utils.parse_env_file to support env-files. See docs for usage.
- Added support for arbitrary log drivers
- Added support for URL paths in the docker host URL (base_url)
- Drastically improved support for .dockerignore syntax

Bugfixes

- Fixed a bug where exec_inspect would allow invocation when the API version was too low.
- Fixed a bug where docker.utils.ports.split_port would break if an open range was provided.
- Fixed a bug where invalid image IDs / container IDs could be provided to bypass or reroute request URLs
- Default base_url now adapts depending on the OS (better Windows support)
- Fixed a bug where using an integer as the user param in Client.create_container would result in a failure.

Miscellaneous

- Docs fixes
- Integration tests are now run as part of our continuous integration.
- Updated dependency on six library

1.3.1

List of PRs / issues for this release

Bugfixes

- Fixed a bug where empty chunks in streams was misinterpreted as EOF.
- datetime arguments passed to Client.events parameters since and until are now always considered to be UTC.
- Fixed a bug with Docker 1.7.x where the wrong auth headers were being passed in Client.build, failing builds that depended on private images.
- Client.exec_create can now retrieve the Id key from a dictionary for its container param.

Miscellaneous

- 404 API status now raises docker.errors.NotFound. This exception inherits APIError which was used previously.
- Docs fixes
- Test fixes

1.3.0

List of PRs / issues for this release

Deprecation warning

- As announced in the 1.2.0 release, Client.execute has been removed in favor of Client.exec_create and Client.exec_start.

Features

- extra_hosts parameter in host config can now also be provided as a list.
- Added support for memory_limit and memswap_limit in host config to comply with recent deprecations.
- Added support for volume_driver in Client.create_container
- Added support for advanced modes in volume binds (using the mode key)
- Added support for decode in Client.build (decodes JSON stream on the fly)
- docker-py will now look for login configuration under the new config path, and fall back to the old ~/.dockercfg path if not present.

Bugfixes

- Configuration file lookup now also work on platforms that don't define a $HOME environment variable.
- Fixed an issue where pinging a v2 private registry wasn't working properly, preventing users from pushing and pulling.
- pull parameter in Client.build now defaults to False. Fixes a bug where the default options would try to force a pull of non-remote images.
- Fixed a bug where getting logs from tty-enabled containers wasn't working properly with more recent versions of Docker

- Client.push and Client.pull will now raise exceptions if the HTTP status indicates an error.
- Fixed a bug with adapter lookup when using the Unix socket adapter (this affected some weird edge cases, see issue #647 for details)
- Fixed a bug where providing timeout=None to Client.stop would result in an exception despite the usecase being valid.
- Added git@ to the list of valid prefixes for remote build paths.

Dependencies

- The websocket-client dependency has been updated to a more recent version. This new version also supports Python 3.x, making attach_socket available on those versions as well.

Documentation

- Various fixes

1.2.3

List of PRs / issues for this release

Deprecation warning

- Passing host config in the Client.start method is now deprecated. Please use the host_config in Client.create_container instead.

Features

- Added support for privileged param in Client.exec_create (only available in API >= 1.19)
- Volume binds can now also be specified as a list of strings.

Bugfixes

- Fixed a bug where the read_only param in host_config wasn't handled properly.
- Fixed a bug in Client.execute (this method is still deprecated).
- The cpuset param in Client.create_container is also passed as the CpusetCpus param (Cpuset deprecated in recent versions of the API)
- Fixed an issue with integration tests being run inside a container (make integration-test)
- Fixed a bug where an empty string would be considered a valid container ID or image ID.
- Fixed a bug in Client.insert

Documentation

- Various fixes

1.2.2

Bugfixes

- Fixed a bug where parameters passed to Client.exec_resize would be ignored (#576)
- Fixed a bug where auth config wouldn't be resolved properly in Client.pull (#577)

1.2.1

Bugfixes

- Fixed a bug where the check_resource decorator would break with some argument-passing methods. (#573)

1.2.0

List of PRs / issues for this release

Deprecation warning

- Client.execute is being deprecated in favor of the more dev-friendly Client.exec_start and Client.exec_create. **It will be removed in 1.3.0**

Features

- Added exec_create, exec_start, exec_inspect and exec_resize to client, accurately mirroring the Exec API
- Added auth_config param to Client.pull (allows to use one-off credentials for this pull request)
- Added support for ipc_mode in host config.
- Added support for the log_config param in host config.
- Added support for the ulimit param in host config.
- Added support for container resource limits in Client.build.
- When a resource identifier (image or container ID) is passed to a Client method, we now check for None values to avoid crashing (now raises docker.errors.NullResource)
- Added tools to parse port ranges inside the new docker.utils.ports package.
- Added a version_info attribute to the docker package.

Bugfixes

- Fixed a bug in Client.port where absence of a certain key in the container's JSON would raise an error (now just returns None)
- Fixed a bug with the trunc parameter in Client.containers having no effect (moved functionality to the client)
- Several improvements have been made to the Client.import_image method.
- Fixed pushing / pulling to v2 registries
- Fixed a bug where passing a container dictionary to Client.commit would fail

Miscellaneous

- Default API version has been bumped to 1.18 (Docker Engine 1.6.0)
- Several testing coverage improvements

- Docs fixes and improvements

1.1.0

Features

- Added dockerfile param support to Client.build (mirrors docker build -f behavior)
- Added the ability to specify 'auto' as version in Client.__init__, allowing the constructor to autodetect the daemon’s API version.

Bugfixes

- Fixed a bug where decoding a result stream using the decode parameter would break when using Python 3.x
- Fixed a bug where some files in .dockerignore weren’t being handled properly
- Fixed resolve_authconfig issues by bringing it closer to Docker Engine’s behavior. This should fix all issues encountered with private registry auth
- Fixed an issue where passwords containing a colon weren’t being handled properly.
- Bumped requests version requirement, which should fix most of the SSL issues encountered recently.

Miscellaneous

- Several integration test improvements.
- Fixed some unclosed resources in unit tests.
- Several docs improvements.

1.0.0

Features

- Added new Client.rename method (docker rename)
- Added now Client.stats method (docker stats)
- Added read_only param support to utils.create_host_config and Client.start (docker run --read-only)
- Added pid_mode param support to utils.create_host_config and Client.start (docker run --pid='host')
- Added since, until and filters params to Client.events.
- Added decode parameter to Client.stats and Client.events to decode JSON objects on the fly (False by default).

Bugfixes

- Fixed a bug that caused Client.build to crash when the provided source was a remote source.

Miscellaneous

- Default API version has been bumped to 1.17 (Docker Engine 1.5.0)
- Client.timeout is now a public attribute, and users are encouraged to use it when request timeouts need to be changed at runtime.
- Added Client.api_version as a read-only property.
- The memswap_limit argument in Client.create_container now accepts string type values similar to mem_limit (‘6g’, ‘120000k’, etc.)
- Improved documentation

0.7.2

Features

- Added support for mac_address in Client.create_container

Bugfixes

- Fixed a bug where streaming responses (pull, push, logs, etc.) were unreliable (#300)
- Fixed a bug where resolve_authconfig wouldn’t properly resolve configuration for private repositories (#468)
- Fixed a bug where some errors wouldn’t be properly constructed in client.py, leading to unhelpful exceptions bubbling up (#466)
- Fixed a bug where Client.build would try to close context when externally provided (custom_context == True) (#458)
- Fixed an issue in create_host_config where empty sequences wouldn’t be interpreted properly (#462)

Miscellaneous

- Added resolve_authconfig tests.

0.7.1

Bugfixes

- setup.py now indicates a maximum version of requests to work around the boot2docker / assert_hostname bug.
- Removed invalid exception when using the Registry Hub’s FQDN when pulling.
- Fixed an issue where early HTTP errors weren’t handled properly in streaming responses.
- Fixed a bug where sockets would close unexpectedly using Python 3.x
- Various fixes for integration tests.

Miscellaneous

- Small doc fixes

0.7.0

Breaking changes

- Passing dns or volumes_from in Client.start with API version < 1.10 will now raise an exception (previously only triggered a warning)

Features

- Added support for host_config in Client.create_container
- Added utility method docker.utils.create_host_config to help build a proper HostConfig dictionary.
- Added support for the pull parameter in Client.build
- Added support for the forcerm parameter in Client.build
- Added support for extra_hosts in Client.start
- Added support for a custom timeout in Client.wait
- Added support for custom .dockercfg loading in Client.login (dockercfg_path argument)

Bugfixes

- Fixed a bug where some output wouldn’t be streamed properly in streaming chunked responses
- Fixed a bug where the devices param didn’t recognize the proper delimiter
- Client.login now properly expands the registry URL if provided.
- Fixed a bug where unicode characters in passed for environment in create_container would break.

Miscellaneous

- Several unit tests and integration tests improvements.
- Client constructor now enforces passing the version parameter as a string.
- Build context files are now ordered by filename when creating the archive (for consistency with docker mainline behavior)

0.6.0

- **This version introduces breaking changes!**

Breaking changes

- The default SSL protocol is now the highest TLS v1.x (was SSL v2.3 before) (Poodle fix)
- The history command now returns a dict instead of a raw JSON string.

Features

- Added the execute command.
- Added pause and unpause commands.
- Added support fo the cpuset param in create_container

- Added support for host devices (devices param in start)
- Added support for the tail param in logs.
- Added support for the filters param in images and containers
- The kwargs_from_env method is now available in the docker.utils module. This should make it easier for boot2docker user to connect to their daemon.

Bugfixes

- Fixed a bug where empty directories weren't correctly included when providing a context to Client.build.
- Fixed a bug where UNIX socket connections weren't properly cleaned up, causing ResourceWarnings to appear in some cases.
- Fixed a bug where docker-py would crash if the docker daemon was stopped while reading a streaming response
- Fixed a bug with streaming responses in Python 3
- remove_image now supports a dict containing an Id key as its id parameter (similar to other methods requiring a resource ID)

Documentation

- Added new MkDocs documentation. Currently hosted on ReadTheDocs

Miscellaneous

- Added tests to sdist
- Added a Makefile for running tests in Docker
- Updated Dockerfile

0.5.3

- Fixed attaching when connecting to the daemon over a UNIX socket.

0.5.2

- Fixed a bug where sockets were closed immediately when attaching over TLS.

0.5.1

- Added a assert_hostname option to TLSConfig which can be used to disable verification of hostnames.
- Fixed SSL not working due to an incorrect version comparison
- Fixed streams not working on Windows

0.5.0

- **This version introduces breaking changes!**

- Added insecure_registry parameter in Client.push and Client.pull. *It defaults to False and code pushing to non-HTTPS private registries might break as a result.*
- Added support for adding and dropping capabilities
- Added support for restart policy
- Added support for string values in Client.create_container's mem_limit
- Added support for .dockerignore file in Client.build

Bugfixes

- Fixed timeout behavior in Client.stop

Miscellaneous

- Client.create_container provides better validation of the volumes parameter
- Improved integration tests

0.4.0

- **This version introduces breaking changes!**
- The base_url parameter in the Client constructor should now allow most of the DOCKER_HOST environment values (except for the fd:// protocol)
  - As a result, URLs that don't specify a port are now invalid (similar to the official client's behavior)
- Added TLS support (see documentation)

Bugfixes

- Fixed an issue with Client.build streamed logs in Python 3

Miscellaneous

- Added unit tests coverage
- Various integration tests fixes

0.3.2

- Default API version is now 1.12 (support for docker 1.0)
- Added new methods Client.get_image and Client.load_image (docker save and docker load)
- Added new method Client.ping
- Added new method Client.resize
- Client.build can now be provided with a custom context using the custom_context parameter.
- Added support for memswap_limit parameter in create_container
- Added support for force parameter in remove_container

- Added support for force and noprune parameters in remove_image
- Added support for timestamps parameter in logs
- Added support for dns_search parameter in start
- Added support for network_mode parameter in start
- Added support for size parameter in containers
- Added support for volumes_from and dns parameters in start. As of API version >= 1.10, these parameters no longer belong to create_container
- Client.logs now uses the logs endpoint when API version is sufficient

Bugfixes

- Fixed a bug in pull where the repo:tag notation wasn't interpreted properly
- Fixed a bug in streaming methods with python 3 (unicode, bytes/str related)
- Fixed a bug in Client.start where legacy notation for volumes wasn't supported anymore.

Miscellaneous

- The client now raises DockerExceptions when appropriate. You can import DockerException (and its subclasses) from the docker.errors module to catch them if needed.
- docker.APIError has been moved to the new docker.errors module as well.
- Client.insert is deprecated in API version > 1.11
- Improved integration tests should now run much faster.
- There is now a single source of truth for the docker-py version number.

0.3.1

- Default API version is now 1.9
- Streaming responses no longer yield blank lines.
- Client.create_container now supports the domainname parameter.
- volumes_from parameter in Client.create_container now supports iterables.
- Auth credentials are provided to the docker daemon when using Client.build (new feature in API version 1.9)

Bugfixes

- Various fixes for response streams (logs, pull, etc.).
- Fixed a bug with Client.push when using API version < 1.5
- Fixed a bug with API version checks.

Miscellaneous

- mock has been removed from the runtime requirements.
- Added installation instructions in the README.

0.3.0

- **This version introduces breaking changes!**
- Support for API version 1.7 through 1.9 (Docker 0.8.0+)
- Default API version is now 1.8
- The client has been updated to support Requests 2.x. requests==2.2.1 is now the recommended version.
- Links can now be specified as tuples in Client.start (see docs for more information)
- Added support for various options in Client.create_container (network_disabled, cpu_shares, working_dir and entrypoint)
- Client.attach has been reworked to work similarly to Client.logs minus the historical data.
- Logs can now be streamed using the stream parameter.
- Added support for tcp:// URLs as client base_url.
- Various auth improvements.
- Added support for custom Client.build timeout.

Bugfixes

- Fixed a bug where determining the protocol of a private registry would sometimes yield the wrong result.
- Fixed a bug where Client.copy wouldn’t accept a dict as argument.
- Fixed several streaming bugs.
- Removed unused parameter in Client.import_image.
- The client’s base_url now tolerates trailing slashes.

Miscellaneous

- Updated integration tests
- Small doc fixes

0.2.3

- Support for API version 1.6
- Added support for links
- Added support for global request timeout
- Added signal parameter in Client.kill
- Added support for publish_all_ports in Client.start

- Client.pull, Client.push and Client.build can be streamed now
- Added support for websockets in Client.attach
- Fixed ports for Docker 0.6.5+
- Added Client.events method (access to the /events endpoint)
- Changed the way the ports and volumes are provided
  in Client.start and Client.create_container` to make them simpler and more intuitive.

Bugfixes

- Fixed a bug where private registries on HTTPS weren't handled properly
- Fixed a bug where auth would break with Python 3

Miscellaneous

- Test improvements
- Slight doc improvements

0.2.2

- Added support for the rm parameter in Client.build
- Added support for tarball imports in Client.import_image through data parameter.
- The command parameter in Client.create_container is now optional (for containers that include a default run command)

Bugfixes

- Fixed Python 3 support
- Fixed a bug where anonymous push/pull would break when no authconfig is present
- Fixed a bug where the quiet parameter wouldn't be taken into account in Client.containers
- Fixed a bug where Client.push would break when pushing to private registries.
- Removed unused registry parameter in Client.pull.
- Removed obsolete custom error message in Client.create_container.

Miscellaneous

- docker-py is now unit-tested, and Travis-CI has been enabled on the source repository.

0.2.1

- Improvements to the tox.ini file

Bugfixes

- Fixed a bug where the package would fail with an ImportError if requests was installed using apt-get
- Fixed a bug where Client.build would fail if given a path parameter.

- Fixed several bugs in Client.login. It should now work with API versions 1.4, 1.5.
- Please note that Client.login currently doesn't write auth to the .dockercfg file, thus **auth is not persistent when using this method.**

0.2.0

- **This version introduces breaking changes!**
- Client.kill, Client.remove_container, Client.remove_image, Client.restart, Client.start, Client.s top and Client.wait don't support varargs anymore.
- Added commands Client.top and Client.copy
- Added lxc_conf parameter to Client.start
- Added support for authentication in Client.pull (API version >=1.5)
- Added support for privileged containers.
- Error management overhaul. The new version should be more consistent and
- All methods that expected a container ID as argument now also support a dict containing an Id key.
- Added license header to python files.
- Several README.md updates.

Bugfixes

- Fixed several bugs with auth config parsing.
- Fixed a bug in Client.push where it would raise an exception if the auth config wasn't loaded.
- Fixed a bug in Client.pull where private registry images wouldn't be parsed properly if it contained port information.

0.1.5

- Client.build now uses tempfiles to store build context instead of storing it in memory
- Added nocache option to Client.build
- Client.remove_container now raises an exception when trying to remove a running container
- Client.create_container now accepts dicts for the environment parameter

Bugfixes

- Fixed a bug in Client.create_container on Python 2.6 where unicode commands would fail to be parsed
- Fixed a bug in Client.build where the tag parameter would not be taken into account

0.1.4

- Added support for API connection through UNIX socket (default for docker 0.5.2+)

0.1.3

- The client now tries to load the auth config from ~/.dockercfg. This is necessary to use the push command if API version is >1.0

0.1.2

- Added a quiet parameter to Client.build (mirrors the q parameter in the API)

0.1.1

- Fixed a bug where the build command would list tar contents before sending the request
- Fixed a bug in Client.port

0.1.0

- **This version introduces breaking changes!**
- Switched to server side build system
- Removed the BuilderClient
- Added support for contextual builds
- Added support for remote URL builds
- Added python 3 support
- Added bind mounts support
- Added API version support
- Fixed a bug where Client.port would fail if provided with a port of type number
- Fixed a bug where Client._post_json wouldn't set the Content-Type header to application/json

0.0.6

- Added support for custom loggers in Client.build
- Added Client.attach command
- Added support for ADD command in builder
- Fixed a bug in Client.logs
- Improved unit tests

0.0.5

- Added tag support for the builder
- Use shlex to parse plain string commands when creating a container
- Fixed several bugs in the builder
- Fixed the quiet option in Client.images
- Unit tests

0.0.4

- Improved error reporting

0.0.3

- Fixed a bug in Client.tag
- Fixed a bug where generated images would be removed after a successful build

0.0.2

- Implemented first version of the builder client

Handling multiplexed streams

Note

The following instruction assume you’re interested in getting output from an `exec` command. These instruction are similarly applicable to the output of `attach`.

First create a container that runs in the background:

```
>>> client = docker.from_env()
>>> container = client.containers.run(
...     'bfirsh/reticulate-splines', detach=True)
```

Prepare the command we are going to use. It prints “hello stdout” in *stdout*, followed by “hello stderr” in *stderr*:

```
>>> cmd = '/bin/sh -c "echo hello stdout ; echo hello stderr
>&2"'
```

We’ll run this command with all four the combinations of `stream` and `demux`.

With `stream=False` and `demux=False`, the output is a string that contains both the *stdout* and the *stderr* output:

```
>>> res = container.exec_run(cmd, stream=False, demux=False)
>>> res.output
b'hello stderr\nhello stdout\n'
```

With `stream=True`, and `demux=False`, the output is a generator that yields strings containing the output of both *stdout* and *stderr*:

```
>>> res = container.exec_run(cmd, stream=True, demux=False)
>>> next(res.output)
b'hello stdout\n'
>>> next(res.output)
b'hello stderr\n'
>>> next(res.output)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

With `stream=True` and `demux=True`, the generator now separates the streams, and yield tuples `(stdout, stderr)`:

```
>>> res = container.exec_run(cmd, stream=True, demux=True)
>>> next(res.output)
(b'hello stdout\n', None)
>>> next(res.output)
(None, b'hello stderr\n')
>>> next(res.output)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

- Finally, with `stream=False` and `demux=True`, the output is a tuple `(stdout, stderr)`:

```
>>> res = container.exec_run(cmd, stream=False, demux=True)
>>> res.output
(b'hello stdout\n', b'hello stderr\n')
```

Swarm services

- Warning: This is a stale document and may contain outdated information. Refer to the API docs for updated classes and method signatures.
- Starting with Engine version 1.12 (API 1.24), it is possible to manage services using the Docker Engine API. Note that the engine needs to be part of a Swarm cluster before you can use the service-related methods.

Creating a service

- The `APIClient.create_service` method lets you create a new service inside the cluster. The method takes several arguments, `task_template` being mandatory. This dictionary of values is most easily produced by instantiating a `TaskTemplate` object.

```
container_spec = docker.types.ContainerSpec(
    image='busybox', command=['echo', 'hello']
)
task_tmpl = docker.types.TaskTemplate(container_spec)
service_id = client.create_service(task_tmpl, name=name)
```

Listing services

- List all existing services using the `APIClient.services` method.

```
client.services(filters={'name': 'mysql'})
```

Retrieving service configuration

- To retrieve detailed information and configuration for a specific service, you may use the `APIClient.inspect_service` method using the service's ID or name.

```
client.inspect_service(service='my_service_name')
```

Updating service configuration

- The `APIClient.update_service` method lets you update a service's configuration. The mandatory `version` argument (used to prevent concurrent writes) can be retrieved using `APIClient.inspect_service`.

```
container_spec = docker.types.ContainerSpec(
    image='busybox', command=['echo', 'hello world']
)
task_tmpl = docker.types.TaskTemplate(container_spec)

svc_version = client.inspect_service(svc_id)['Version']['Index']

client.update_service(
    svc_id, svc_version, name='new_name', task_template=task_tmpl
)
```

Removing a service

- A service may be removed simply using the `APIClient.remove_service` method. Either the service name or service ID can be used as argument.

```
client.remove_service('my_service_name')
```