aiodocker Documentation

Release 0.21.0-

Paul Tagliamonte, Joongi Kim

Jan 19, 2022

CONTENTS

A simple Docker HTTP API wrapper written with asyncio and aiohttp.

CHAPTER

ONE

INSTALLATION

```
pip install aiodocker
```

CHAPTER

TWO

EXAMPLES

```python
import asyncio
import aiodocker

async def list_things():
    docker = aiodocker.Docker()
    print('== Images ==')
    for image in (await docker.images.list()):
        tags = image['RepoTags'][0] if image['RepoTags'] else ''
        print(image['Id'], tags)
    print('== Containers ==')
    for container in (await docker.containers.list()):
        print(f" {container._id}")
    await docker.close()

async def run_container():
    docker = aiodocker.Docker()
    print('== Running a hello-world container ==')
    container = await docker.containers.create_or_replace(
        config={
            'Cmd': ['/bin/ash', '-c', 'echo "hello world"'],
            'Image': 'alpine:latest',
        },
        name='testing',
    )
    await container.start()
    logs = await container.log(stdout=True)
    print(''.join(logs))
    await container.delete(force=True)
    await docker.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(list_things())
    loop.run_until_complete(run_container())
    loop.close()
```

2.1 Source code

The project is hosted on GitHub: https://github.com/aio-libs/aiodocker

Please feel free to file an issue on the bug tracker if you have found a bug or have some suggestion in order to improve the library.

2.2 Communication channels

aio-libs google group: https://groups.google.com/forum/#!forum/aio-libs

Feel free to post your questions and ideas here.

Gitter Chat https://gitter.im/aio-libs/Lobby

We support Stack Overflow. Please add *python-asyncio* tag to your question there.

2.3 Contribution

Please follow the Contribution Guide.

2.4 Author and License

The `aiodocker` package is written by Andrew Svetlov.

It's *Apache 2* licensed and freely available.

2.4.1 Client

Reference

Docker

class aiodocker.docker.**Docker**(*url=None*, *connector=None*, *session=None*, *ssl_context=None*, *api_version='auto'*)

coroutine auth(*self*, ***credentials*)

Return type Dict[str, Any]

coroutine close(*self*)

Return type None

coroutine version(*self*)

Return type Dict[str, Any]

2.4.2 Configs

Create a config

```
import asyncio
import aiodocker

docker = aiodocker.Docker()

async def create_config():
    config = await docker.configs.create(
        name="my_config",
        data="This is my config data"
    )
    await docker.close()
    return config

async def create_service(TaskTemplate):
    service = await docker.services.create(
        task_template=TaskTemplate,
        name="my_service"
    )
    await docker.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    my_config = loop.run_until_complete(create_config())
    TaskTemplate = {
        "ContainerSpec": {
            "Image": "redis",
            "Configs": [
            {
                "File": {
                    "Name": my_config["Spec"]["Name"],
                    "UID": "0",
                    "GID": "0",
                    "Mode": 292
                },
                "ConfigID": my_config["ID"],
                "ConfigName": my_config["Spec"]["Name"],
            }
            ],
        },
    }
    loop.run_until_complete(create_service(TaskTemplate))
    loop.close()
```

Reference

DockerConfigs

class aiodocker.configs.**DockerConfigs**(*docker*)

coroutine create(*self*, *name*, *data*, *, *b64=False*, *labels=None*, *templating=None*)
Create a config

Parameters

- **name** (str) – name of the config
- **labels** (Optional[Mapping[str, str]]) – user-defined key/value metadata
- **data** (str) – config data
- **b64** (bool) – True if data is already Base64-url-safe-encoded
- **templating** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).

Return type Mapping[str, Any]

Returns a dict with info of the created config

coroutine delete(*self*, *config_id*)
Remove a config

Parameters **config_id** (str) – ID or name of the config

Return type bool

Returns True if successful

coroutine inspect(*self*, *config_id*)
Inspect a config

Parameters **config_id** (str) – ID of the config

Return type Mapping[str, Any]

Returns a dict with info about a config

coroutine list(*self*, *, *filters=None*)
Return a list of configs

Parameters **filters** (Optional[Mapping]) – a dict with a list of filters

Available filters: id=<config id> label=<key> or label=<key>=value name=<config name> names=<config name>

Return type List[Mapping]

coroutine update(*self*, *config_id*, *version*, *, *name=None*, *data=None*, *b64=False*, *labels=None*, *templating=None*)
Update a config.

Parameters

- **config_id** (str) – ID of the config.
- **name** (Optional[str]) – name of the config

- **labels** (Optional[Mapping[str, str]]) – user-defined key/value metadata
- **data** (Optional[str]) – config data
- **b64** (bool) – True if data is already Base64-url-safe-encoded
- **templating** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).

Return type bool

Returns True if successful.

2.4.3 Containers

Create a container

```
import asyncio
import aiodocker

docker = aiodocker.Docker()

config = {
     "Cmd": ["/bin/ls"],
     "Image": "alpine:latest",
     "AttachStdin": False,
     "AttachStdout": False,
     "AttachStderr": False,
     "Tty": False,
     "OpenStdin": False,
 }

async def create_container():
    container = await docker.containers.create(config=config)
    print(container)
    await docker.close()


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(create_container())
    loop.close()
```

Reference

DockerContainers

class aiodocker.docker.**DockerContainers**(*docker*)

container(*container_id*, ***kwargs*)

coroutine create(*config*, *, *name=None*)

coroutine create_or_replace(*name*, *config*)

exec(*exec_id*)
Return Exec instance for already created exec object.

Return type Exec

coroutine get(*container*, ***kwargs*)

coroutine list(***kwargs*)

coroutine run(*self*, *config*, ***, *auth=None*, *name=None*)
Create and start a container.

If container.start() will raise an error the exception will contain a *container_id* attribute with the id of the container.

Use *auth* for specifying credentials for pulling absent image from a private registry.

DockerContainer

class aiodocker.docker.**DockerContainer**(*docker*, ***kwargs*)

attach(***, *stdout=False*, *stderr=False*, *stdin=False*, *detach_keys=None*, *logs=False*)

Return type Stream

coroutine commit(*self*, ***, *repository=None*, *tag=None*, *message=None*, *author=None*, *changes=None*, *config=None*, *pause=True*)
Commit a container to an image. Similar to the `docker commit` command.

Return type Dict[str, Any]

coroutine delete(***kwargs*)

coroutine exec(*self*, *cmd*, *stdout=True*, *stderr=True*, *stdin=False*, *tty=False*, *privileged=False*, *user=''*, *environment=None*, *workdir=None*, *detach_keys=None*)

coroutine get_archive(*self*, *path*)

Return type TarFile

property id: str

Return type str

coroutine kill(***kwargs*)

log(***, *stdout=False*, *stderr=False*, *follow=False*, ***kwargs*)

coroutine pause(*self*)

Return type None

coroutine port(*private_port*)

coroutine put_archive(*path*, *data*)

coroutine rename(*newname*)

coroutine resize(*self*, ***, *h*, *w*)

Return type None

coroutine restart(*timeout=None*)

coroutine show(***kwargs*)

coroutine start(***kwargs*)

stats(*, stream=True*)

coroutine stop(***kwargs*)

coroutine unpause(*self*)

Return type None

coroutine wait(*, timeout=None, **kwargs*)

coroutine websocket(***params*)

2.4.4 Images

Reference

DockerImages

class aiodocker.images.**DockerImages**(*docker*)

build(*, remote: str = 'None', fileobj: BinaryIO = 'None', path_dockerfile: str = 'None', tag: str = 'None', quiet: bool = 'False', nocache: bool = 'False', buildargs: Mapping = 'None', pull: bool = 'False', rm: bool = 'True', forcerm: bool = 'False', labels: Mapping = 'None', stream: typing_extensions.Literal[False] = 'False', encoding: str = 'None'*) → Dict[str, Any]

build(*, remote: str = 'None', fileobj: BinaryIO = 'None', path_dockerfile: str = 'None', tag: str = 'None', quiet: bool = 'False', nocache: bool = 'False', buildargs: Mapping = 'None', pull: bool = 'False', rm: bool = 'True', forcerm: bool = 'False', labels: Mapping = 'None', stream: typing_extensions.Literal[True], encoding: str = 'None'*) → AsyncIterator[Dict[str, Any]]

Build an image given a remote Dockerfile or a file object with a Dockerfile inside

Parameters

- **path_dockerfile** (Optional[str]) – path within the build context to the Dockerfile
- **remote** (Optional[str]) – a Git repository URI or HTTP/HTTPS context URI
- **quiet** (bool) – suppress verbose build output
- **nocache** (bool) – do not use the cache when building the image
- **rm** (bool) – remove intermediate containers after a successful build
- **pull** (bool) – downloads any updates to the FROM image in Dockerfiles
- **encoding** (Optional[str]) – set *Content-Encoding* for the file object your send
- **forcerm** (bool) – always remove intermediate containers, even upon failure
- **labels** (Optional[Mapping]) – arbitrary key/value labels to set on the image
- **fileobj** (Optional[BinaryIO]) – a tar archive compressed or not

Return type Any

coroutine delete(*self, name, *, force=False, noprune=False*)

Remove an image along with any untagged parent images that were referenced by that image

Parameters

- **name** (str) – name/id of the image to delete
- **force** (bool) – remove the image even if it is being used by stopped containers or has other tags
- **noprune** (bool) – don't delete untagged parent images

Return type List

Returns List of deleted images

export_image(*name*)
Get a tarball of an image by name or id.

Parameters **name** (str) – name/id of the image to be exported

Returns Streamreader of tarball image

coroutine get(*self*, *name*)

Return type Mapping

coroutine history(*self*, *name*)

Return type Mapping

import_image(*data*, *stream=False*)
Import tarball of image to docker.

Parameters **data** – tarball data of image to be imported

Returns Tarball of the image

coroutine inspect(*self*, *name*)
Return low-level information about an image

Parameters **name** (str) – name of the image

Return type Mapping

coroutine list(*self*, ***params*)
List of images

Return type Mapping

pull(*from_image: str*, ***, *auth: Optional[Union[MutableMapping, str, bytes]] = 'None'*, *tag: str = 'None'*, *repo: str = 'None'*, *stream: typing_extensions.Literal[False] = 'False'*) → Dict[str, Any]
pull(*from_image: str*, ***, *auth: Optional[Union[MutableMapping, str, bytes]] = 'None'*, *tag: str = 'None'*, *repo: str = 'None'*, *stream: typing_extensions.Literal[True]*) → AsyncIterator[Dict[str, Any]]
Similar to *docker pull*, pull an image locally

Parameters

- **fromImage** – name of the image to pull
- **repo** (Optional[str]) – repository name given to an image when it is imported
- **tag** (Optional[str]) – if empty when pulling an image all tags for the given image to be pulled
- **auth** (Union[MutableMapping, str, bytes, None]) – special {'auth': base64} pull private repo

Return type Any

push(*name: str*, ***, *auth: Union[MutableMapping, str, bytes] = 'None'*, *tag: str = 'None'*, *stream: typing_extensions.Literal[False] = 'False'*) → Dict[str, Any]
push(*name: str*, ***, *auth: Union[MutableMapping, str, bytes] = 'None'*, *tag: str = 'None'*, *stream: typing_extensions.Literal[True]*) → AsyncIterator[Dict[str, Any]]

Return type Any

coroutine tag(*self*, *name*, *repo*, ***, *tag=None*)
Tag the given image so that it becomes part of a repository.

Parameters

- **name** (str) – name/id of the image to be tagged
- **repo** (str) – the repository to tag in
- **tag** (Optional[str]) – the name for the new tag

Return type bool

2.4.5 Secrets

Create a secret

```
import asyncio
import aiodocker

docker = aiodocker.Docker()

async def create_secret():
    secret = await docker.secrets.create(
        name="my_secret",
        data="you can not read that terrible secret"
    )
    await docker.close()
    return secret

async def create_service(TaskTemplate):
    service = await docker.services.create(
        task_template=TaskTemplate,
        name="my_service"
    )
    await docker.close()

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    my_secret = loop.run_until_complete(create_secret())
    TaskTemplate = {
        "ContainerSpec": {
            "Image": "redis",
            "Secrets": [
            {
                "File": {
```

(continues on next page)

(continued from previous page)

```
                    "Name": my_secret["Spec"]["Name"],
                    "UID": "0",
                    "GID": "0",
                    "Mode": 292
                },
                "SecretID": my_secret["ID"],
                "SecretName": my_secret["Spec"]["Name"]
            }
            ],
        },
    }
    loop.run_until_complete(create_service(TaskTemplate))
    loop.close()
```

Reference

DockerSecrets

class aiodocker.secrets.**DockerSecrets**(*docker*)

coroutine create(*self*, *name*, *data*, *, *b64=False*, *labels=None*, *driver=None*, *templating=None*)

Create a secret

Parameters

- **name** (str) – name of the secret
- **labels** (Optional[Mapping[str, str]]) – user-defined key/value metadata
- **data** (str) – data to store as secret
- **b64** (bool) – True if data is already Base64-url-safe-encoded
- **driver** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).
- **templating** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).

Return type Mapping[str, Any]

Returns a dict with info of the created secret

coroutine delete(*self*, *secret_id*)

Remove a secret

Parameters **secret_id** (str) – ID of the secret

Return type bool

Returns True if successful

coroutine inspect(*self*, *secret_id*)

Inspect a secret

Parameters **secret_id** (str) – ID of the secret

Return type Mapping[str, Any]

Returns a dict with info about a secret

coroutine list(*self*, ***, *filters=None*)
Return a list of secrets

Parameters **filters** (Optional[Mapping]) – a dict with a list of filters

Available filters: id=<secret id> label=<key> or label=<key>=value name=<secret name> names=<secret name>

Return type List[Mapping]

coroutine update(*self*, *secret_id*, *version*, ***, *name=None*, *data=None*, *b64=False*, *labels=None*, *driver=None*, *templating=None*)
Update a secret.

Parameters

- **secret_id** (str) – ID of the secret.
- **name** (Optional[str]) – name of the secret
- **labels** (Optional[Mapping[str, str]]) – user-defined key/value metadata
- **data** (Optional[str]) – data to store as secret
- **b64** (bool) – True if data is already Base64-url-safe-encoded
- **driver** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).
- **templating** (Optional[Mapping]) – Driver represents a driver (network, logging, secrets).

Return type bool

Returns True if successful.

2.4.6 Services

Create a service

```
import asyncio
import aiodocker

docker = aiodocker.Docker()

TaskTemplate = {
    "ContainerSpec": {
        "Image": "redis",
        },
}

async def create_service():
    service = await docker.services.create(
        task_template=TaskTemplate,
        name="my_service"
    )
    await docker.close()
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(create_service())
    loop.close()
```

Reference

DockerServices

class aiodocker.services.**DockerServices**(*docker*)

coroutine create(*self*, *task_template*, *, *name=None*, *labels=None*, *mode=None*, *update_config=None*, *rollback_config=None*, *networks=None*, *endpoint_spec=None*, *auth=None*, *registry=None*)

Create a service

Parameters

- **task_template** (Mapping[str, Any]) – user modifiable task configuration
- **name** (Optional[str]) – name of the service
- **labels** (Optional[Mapping[str, str]]) – user-defined key/value metadata
- **mode** (Optional[Mapping]) – scheduling mode for the service
- **update_config** (Optional[Mapping]) – update strategy of the service
- **rollback_config** (Optional[Mapping]) – rollback strategy of the service
- **networks** (Optional[List]) – array of network names/IDs to attach the service to
- **endpoint_spec** (Optional[Mapping]) – ports to expose
- **auth** (Union[MutableMapping, str, bytes, None]) – authentication information, can be a string, dict or bytes
- **registry** (Optional[str]) – used when auth is specified, it provides domain/IP of the registry without a protocol

Return type Mapping[str, Any]

Returns a dict with info of the created service

coroutine delete(*self*, *service_id*)

Remove a service

Parameters **service_id** (str) – ID or name of the service

Return type bool

Returns True if successful

coroutine inspect(*self*, *service_id*)

Inspect a service

Parameters **service_id** (str) – ID or name of the service

Return type Mapping[str, Any]

Returns a dict with info about a service

coroutine list(*self*, ***, *filters=None*)
Return a list of services

Parameters **filters** (Optional[Mapping]) – a dict with a list of filters

Available filters: id=<service id> label=<service label> mode=["replicated"|"global"] name=<service name>

Return type List[Mapping]

logs(*service_id*, ***, *details=False*, *follow=False*, *stdout=False*, *stderr=False*, *since=0*, *timestamps=False*, *is_tty=False*, *tail='all'*)
Retrieve logs of the given service

Parameters

- **details** (bool) – show service context and extra details provided to logs
- **follow** (bool) – return the logs as a stream.
- **stdout** (bool) – return logs from stdout
- **stderr** (bool) – return logs from stderr
- **since** (int) – return logs since this time, as a UNIX timestamp
- **timestamps** (bool) – add timestamps to every log line
- **is_tty** (bool) – the service has a pseudo-TTY allocated
- **tail** (str) – only return this number of log lines from the end of the logs, specify as an integer or *all* to output all log lines.

Return type Union[str, AsyncIterator[str]]

coroutine update(*self*, *service_id*, *version*, ***, *image=None*, *rollback=False*)
Update a service. If rollback is True image will be ignored.

Parameters

- **service_id** (str) – ID or name of the service.
- **version** (str) – Version of the service that you want to update.
- **rollback** (bool) – Rollback the service to the previous service spec.

Return type bool

Returns True if successful.

2.4.7 Swarm

Reference

DockerSwarm

class aiodocker.swarm.**DockerSwarm**(*docker*)

coroutine **init**(*self*, ***, *advertise_addr=None*, *listen_addr='0.0.0.0:2377'*, *force_new_cluster=False*, *swarm_spec=None*)

Initialize a new swarm.

Parameters

- **ListenAddr** – listen address used for inter-manager communication
- **AdvertiseAddr** – address advertised to other nodes.
- **ForceNewCluster** – Force creation of a new swarm.
- **SwarmSpec** – User modifiable swarm configuration.

Return type str

Returns id of the swarm node

coroutine **inspect**(*self*)

Inspect a swarm.

Return type Mapping

Returns Info about the swarm

coroutine **join**(*self*, ***, *remote_addrs*, *listen_addr='0.0.0.0:2377'*, *join_token*, *advertise_addr=None*, *data_path_addr=None*)

Join a swarm.

Parameters

- **listen_addr** (str) – Used for inter-manager communication
- **advertise_addr** (Optional[str]) – Externally reachable address advertised to other nodes.
- **data_path_addr** (Optional[str]) – Address or interface to use for data path traffic.
- **remote_addrs** (Iterable[str]) – Addresses of manager nodes already participating in the swarm.
- **join_token** (str) – Secret token for joining this swarm.

Return type bool

coroutine **leave**(*self*, ***, *force=False*)

Leave a swarm.

Parameters **force** (bool) – force to leave the swarm even if the node is a master

Return type bool

2.4.8 Volumes

Reference

DockerVolumes

class aiodocker.docker.**DockerVolumes**(*docker*)

coroutine **create**(*config*)

coroutine **get**(*id*)

coroutine **list**(*, *filters=None*)
Return a list of volumes

Parameters **filters** – a dict with a list of filters

Available filters: dangling=<boolean> driver=<volume-driver-name> label=<key> or label=<key>:<value> name=<volume-name>

DockerVolume

class aiodocker.docker.**DockerVolume**(*docker*, *name*)

coroutine **delete**()

coroutine **show**()

2.4.9 Tasks

Reference

DockerTasks

class aiodocker.tasks.**DockerTasks**(*docker*)

coroutine **inspect**(*self*, *task_id*)
Return info about a task

Parameters **task_id** (str) – is ID of the task

Return type Mapping[str, Any]

coroutine **list**(*self*, *, *filters=None*)
Return a list of tasks

Parameters **filters** (Optional[Mapping]) – a collection of filters

Available filters: desired-state=(running | shutdown | accepted) id=<task id> label=key or label="key=value" name=<task name> node=<node id or name> service=<service name>

Return type List[Mapping]

2.4.10 Log

Reference

DockerLog

class aiodocker.docker.**DockerLog**(*docker*, *container*)

listen()

Return type ChannelSubscriber

coroutine **run**(*self*, ***params*)

Return type None

coroutine **stop**(*self*)

Return type None

subscribe()

Return type ChannelSubscriber

2.4.11 Events

Reference

DockerEvents

class aiodocker.docker.**DockerEvents**(*docker*)

listen()

coroutine **run**(***params*)
Query the events endpoint of the Docker daemon.

Publish messages inside the asyncio queue.

coroutine **stop**()

subscribe(*, *create_task=True*, ***params*)
Subscribes to the Docker events channel. Use the keyword argument create_task=False to prevent automatically spawning the background tasks that listen to the events.

This function returns a ChannelSubscriber object.

2.4.12 Exceptions

Reference

DockerError

class aiodocker.exceptions.**DockerError**(*status*, *data*, **args*)

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

K

L

P

R

S

T

U

V

W