AIO Consul Documentation

Release 0.2

Xavier Barbosa

March 24, 2015

Contents

1 Installation **3**

2 Tutorial **5**

3 In the pit **7**
3.1 Client 7
3.2 Objects 8

4 Endpoints **11**
4.1 ACL 11
4.2 Agent 13
4.3 Catalog 19
4.4 Event 21
4.5 Health 22
4.6 KV 23
4.7 Session 25

5 Indices and tables **29**

Python Module Index **31**

AIOConsul is a Python >= 3.3 library for requesting consul API, build on top of asyncio and aiohttp.

Currently, this library aims a full compatibility with consul 0.5.

CHAPTER 1

Installation

```
pip install aioconsul
```

CHAPTER 2

Tutorial

In this example I will show you how to join my cluster with another:

```
from aioconsul import Consul
client = Consul('my.node.ip')

# do I have a members?
members = yield from client.agent.members()
assert len(members) == 1, "I am alone in my cluster"

# let's join another cluster
joined = yield from client.agent.join('other.node.ip')
if joined:
    members = yield from client.agent.members()
    assert len(members) > 1, "I'm not alone anymore"
```

And display the catalog:

```
for dc in (yield from client.catalog.datacenters()):
    print(dc)

for service, tags in (yield from client.catalog.services()).items():
    print(service, tags)

for node in (yield from client.catalog.nodes()):
    print(node.name, node.address)
```

CHAPTER 3

In the pit

3.1 Client

```
from aioconsul import Consul
client = Consul('my.node.ip', token='my.token', consistency='stale')
info = yield from client.agent.info()
```

3.1.1 Internals

class **Consul** (*host=None*, *, *token=None*, *consistency=None*)

Most of the read query endpoints support multiple levels of consistency. Since no policy will suit all clients' needs, these consistency modes allow the user to have the ultimate say in how to balance the trade-offs inherent in a distributed system.

Variables

- **host** (*str*) – host api
- **version** (*str*) – api version
- **token** (*str*) – Token ID
- **consistency** (*str*) – default, consistent or stale

delete (*path*, ***kwargs*)

Short-cut towards *request()*

get (*path*, ***kwargs*)

Short-cut towards *request()*

post (*path*, ***kwargs*)

Short-cut towards *request()*

put (*path*, ***kwargs*)

Short-cut towards *request()*

request (*method*, *path*, ***kwargs*)

Makes single http request.

Requested url will be in the form of `{host}/{version}/{path}`

Parameters

- **method** (*str*) – http method
- **path** (*str*) – path after version

Keyword Arguments

- **params** (*dict*) – get params

- **data** (*str*) – body of the request
- **headers** (*dict*) – custom headers

3.2 Objects

3.2.1 Consul objects

class **Member**(*name*, *address*, *port*, ***opts*)
Node as exposed by *AgentEndpoint*.

Variables

- **name** (*str*) – name
- **address** (*str*) – address
- **port** (*int*) – port
- **status** (*int*) – status
- **tags** (*dict*) – tags
- **delegate_cur** (*int*) – delegate current
- **delegate_max** (*int*) – delegate maximum
- **delegate_min** (*int*) – delegate mininum
- **protocol_cur** (*int*) – protocol current
- **protocol_max** (*int*) – protocol maximum
- **protocol_min** (*int*) – protocol mininum

class **Node**(*name*, *address*)
Node as exposed by *CatalogEndpoint*.

Variables

- **name** (*str*) – name
- **address** (*str*) – address

class **Service**(*id*, *, *name*)

Variables

- **id** (*str*) – id
- **name** (*str*) – name

class **NodeService**(*id*, *, *name*, *address=None*, *port=None*, *tags=None*)
A service that belongs to a *Node*.

Variables

- **id** (*str*) – id
- **name** (*str*) – name
- **address** (*str*) – address
- **port** (*int*) – port
- **tags** (*list*) – tags

class **Token** (*id*, ***, *name*, *type*, *rules*, *create_index=None*, *modify_index=None*)

A token has an ID, a name, a type and a `Rule` set. The ID is randomly generated by the API, making it unfeasible to guess. The name is opaque and human readable. The type is either "client" meaning it cannot modify ACL rules, and is restricted by the provided rules, or is "management" and is allowed to perform all actions.

The token ID is passed along with each RPC request to the servers.

Variables

- **id** (*str*) – token id
- **name** (*str*) – token name
- **type** (*str*) – token type
- **rules** (*list*) – list of token `Rule`
- **create_index** (*int*) – create index when fetched
- **modify_index** (*int*) – modify index when fetched

class **Rule**

Describe the policy that must be enforced.

Key policies provide both a prefix and a policy. The rules are enforced using a longest-prefix match policy. This means we pick the most specific policy possible. The policy is either "read", "write" or "deny".

Services policies provide both a service name and a policy. The rules are enforced using an exact match policy. The default rule is provided using the empty string. The policy is either "read", "write", or "deny".

Variables

- **type** (*str*) – key or service
- **value** (*str*) – value of rule
- **policy** (*str*) – read, write or deny

class **Check** (*id*, ***, *name*, *status=None*, *notes=None*, *output=None*, *service_id=None*, *service_name=None*, *node=None*)

Variables

- **id** (*str*) – id
- **name** (*str*) – name
- **status** (*str*) – status
- **notes** (*str*) – notes
- **output** (*str*) – output
- **service_id** (*str*) – service_id
- **service_name** (*str*) – service_name
- **node** (*str*) – node

class **Event** (*name*, ***, *id=None*, *payload=None*, *service_filter=None*, *node_filter=None*, *tag_filter=None*, *version=None*, *l_time=None*)

Variables

- **id** (*str*) – id
- **name** (*str*) – name
- **payload** (*str*) – payload
- **node_filter** (*str*) – node_filter
- **service_filter** (*str*) – service_filter

- **tag_filter** (*str*) – tag_filter
- **version** (*str*) – version
- **l_time** (*str*) – l_time

class Session(*id*, *, *node=None*, *checks=None*, *create_index=None*, *behavior=None*)

Variables

- **id** (*str*) – session id
- **behavior** (*str*) – session behavior (delete, release)
- **checks** (*str*) – session checks
- **create_index** (*str*) – used for locks
- **node** (*str*) – attached node

class Key(*name*, *, *session=None*, *create_index=None*, *modify_index=None*, *lock_index=None*)

Variables

- **name** (*str*) – key
- **session** (*str*) – session that acquired this key
- **create_index** (*int*) – create_index
- **lock_index** (*int*) – lock_index
- **modify_index** (*int*) – modify_index

3.2.2 Data collections

class DataSet(*values*, **params*)
Just a *set* that holds response headers.

Variables

- **modify_index** (*int*) – modify index
- **last_contact** (*str*) – last contact
- **known_leader** (*bool*) – leader was known while requesting data

class DataMapping(*values*, **params*)
Just a *dict* that holds response headers.

Variables

- **modify_index** (*int*) – modify index
- **last_contact** (*str*) – last contact
- **known_leader** (*bool*) – leader was known while requesting data

CHAPTER 4

Endpoints

4.1 ACL

The ACL endpoints are used to create, update, destroy, and query ACL tokens.

How to create a new token:

```
from aioconsul import Consul, ACLPermissionDenied
import pytest

master = Consul(token='master.token')

# create a token that disable almost everything
token = (yield from master.acl.create('my-acl', rules=[
    ('key', '', 'read'),
    ('key', 'foo/', 'deny'),
]))

# open a new master with the fresh token
node = Consul(token=token)
yield from node.kv.get('foo')

# writes must be disabled
with pytest.raises(ACLPermissionDenied):
    yield from node.kv.set('foo', 'baz')

# everything under `foo/` must be hidden
with pytest.raises(node.kv.NotFound):
    yield from node.kv.get('foo/bar')
```

How to list tokens:

```
from aioconsul import Consul
master = Consul(token='master.token')

# create a token that disable almost everything
for token in (yield from master.acl()):
    print(token)
```

See *Token* and *Rule*.

4.1.1 Internals

class ACLEndpoint (*client*, *supported=None*)
ACL Endpoint

Variables **supported** (*bool*) – Used as a barrier, it will be defined at the first request. Set it to `None` for resetting.

exception NotFound
Raises when a token was not found.

`ACLEndpoint.__call__()`
Returns a set of all Token.

Returns *DataSet* – set of *Token* instances

`ACLEndpoint.clone(token, *, obj=False)`
Clone a token.

The result can be used as a token into *Consul* instances.

Parameters

- **token** (Token) – token or id to clone
- **obj** (*bool*) – must returns a *Token* instance at the cost of additional http queries.

Returns *str* | *Token* – id or *Token*, depending of *obj* parameter.

`ACLEndpoint.create(name, *, type=None, rules=None, obj=False)`
Create a new token.

A *Token* has a name, a type, and a set of ACL rules.

The result can be used as a token into *Consul* instances.

Parameters

- **name** (*str*) – human name
- **type** (*str*) – `client` or `management`
- **rules** (*list*) – a set of rules to implement, which can be a list of *Rule* instances or 3 length tuples.
- **obj** (*bool*) – must returns a *Token* instance at the cost of additional http queries.

Returns *str* | *Token* – id or *Token*, depending of *obj* parameter.

`ACLEndpoint.delete(token)`
Destroy a token.

Parameters **token** (Token) – token or id to delete

Returns *bool* – `True`, it was destroyed

`ACLEndpoint.destroy(token)`
Destroy a token.

Parameters **token** (Token) – token or id to delete

Returns *bool* – `True`, it was destroyed

`ACLEndpoint.get(token)`
Get a token.

The result can be used as a token into *Consul* instances.

Parameters **token** (Token) – token or id

Returns *Token* – token instance

Raises *NotFound* – token was not found

`ACLEndpoint.is_supported()`
Tells if ACL is supported or not.

Returns *bool* – yes or no

ACLEndpoint.**items**()
Returns a set of all Token.

Returns *DataSet* – set of *Token* instances

ACLEndpoint.**update**(*token*, *, *name=None*, *type=None*, *rules=None*, *obj=False*)
Update a token.

The result can be used as a token into *Consul* instances.

Parameters

- **token** (Token) – token or id to update
- **name** (Token) – human name
- **type** (*str*) – `client` or `management`
- **rules** (*list*) – a set of rules to implement, which can be a list of *Rule* instances or 3 length tuples.
- **obj** (*bool*) – must returns a *Token* instance at the cost of additional http queries.

Returns *str* | *Token* – id or *Token*, depending of *obj* parameter.

4.2 Agent

The Agent endpoints are used to interact with the local Consul agent. Usually, services and checks are registered with an agent which then takes on the burden of keeping that data synchronized with the cluster.

The following endpoints are supported:

Returns the checks the local agent is managing:

```
>>> yield from client.agent.checks()
```

Returns the services the local agent is managing:

```
>>> yield from client.agent.services()
```

Returns the members as seen by the local serf agent:

```
>>> members = yield from client.agent.members()
```

Returns the local node configuration:

```
>>> yield from client.agent.config()
```

Manages node maintenance mode:

```
>>> yield from client.agent.disable()
>>> yield from client.agent.enable()
```

Triggers the local agent to join a node:

```
>>> yield from client.agent.join('other.node.ip')
```

Forces removal of a node:

```
>>> yield from client.agent.force_leave('other.node.ip')
```

Registers a new local check:

```
>>> check = yield from client.agent.checks.register_ttl('my-ttl-check')
```

Registers a new local service:

```
>>> service = yield from client.agent.services.register('my-service')
```

Disable a local service:

```
>>> yield from client.agent.services.disable(service)
```

Etc...

4.2.1 Internals

class **AgentEndpoint** (*client*)

config ()
Returns configuration of agent.

Returns *Config* – instance

disable (*reason=None*)
Disable agent.

Parameters **reason** (*str*) – human readable reason

Returns *bool* – `True` it has been disabled

enable (*reason=None*)
Enable agent.

Parameters **reason** (*str*) – human readable reason

Returns *bool* – `True` it has been enabled

force_leave (*member*)
Asks a member to leave the cluster.

Parameters **member** (Member) – member or name

Returns *bool* – action status

join (*address*, *, *wan=None*)
Asks the agent to join a cluster.

Parameters

- **address** (*str*) – address to join
- **wan** (*str*) – use wan?

Returns *bool* – agent status

me ()
Returns the member object of agent.

Returns *Member* – instance

members ()
Returns a set of members.

Returns *set* – set of *Member* instances

class **AgentCheckEndpoint** (*client*)

exception **NotFound**
Raised when check was not found

AgentCheckEndpoint.**__call__** ()
Returns the checks the local agent is managing.

Returns *set* – set of *Check* instances

AgentCheckEndpoint.**create**(*name*, ***params*)
Registers a new local check.

Parameters

- **name** (*str*) – check name
- **http** (*str*) – url to ping
- **script** (*str*) – path to script
- **ttl** (*str*) – period status update
- **interval** (*str*) – evaluate script every *ìnterval*
- **id** (*str*) – check id
- **notes** (*str*) – human readable notes

Returns *Check* – instance

AgentCheckEndpoint.**critical**(*check*, *note=None*)
Marks a local test as critical.

Parameters

- **check** (Check) – check or id
- **note** (*str*) – human readable reason

Returns *bool* – `True` check has been deregistered

AgentCheckEndpoint.**delete**(*check*)
Deregisters a local check.

Parameters **check** (Check) – check or id

Returns *bool* – `True` check has been deregistered

AgentCheckEndpoint.**deregister**(*check*)
Deregisters a local check.

Parameters **check** (Check) – check or id

Returns *bool* – `True` check has been deregistered

AgentCheckEndpoint.**failing**(*check*, *note=None*)
Marks a local test as critical.

Parameters

- **check** (Check) – check or id
- **note** (*str*) – human readable reason

Returns *bool* – `True` check has been deregistered

AgentCheckEndpoint.**get**(*check*)
Get a local test.

Parameters **check** (Check) – check or id

Returns *Check* – instance

Raises *NotFound* – check was not found

AgentCheckEndpoint.**items**()
Returns the checks the local agent is managing.

Returns *set* – set of *Check* instances

AgentCheckEndpoint.**mark**(*check*, *state*, *, *note=None*)
Set state of a local test.

Parameters

- **check** (Check) – check or id
- **state** (*str*) – `passing`, `warning` or `failing`
- **note** (*str*) – human readable reason

Returns *bool* – `True` check has been deregistered

`AgentCheckEndpoint.`**`passing`**(*check*, *note=None*)
Marks a local test as passing.

Parameters

- **check** (Check) – check or id
- **note** (*str*) – human readable reason

Returns *bool* – `True` check has been deregistered

`AgentCheckEndpoint.`**`register`**(*name*, ***params*)
Registers a new local check.

Parameters

- **name** (*str*) – check name
- **http** (*str*) – url to ping
- **script** (*str*) – path to script
- **ttl** (*str*) – period status update
- **interval** (*str*) – evaluate script every *ìnterval*
- **id** (*str*) – check id
- **notes** (*str*) – human readable notes

Returns *Check* – instance

`AgentCheckEndpoint.`**`register_http`**(*name*, *http*, *, *interval*, *id=None*, *notes=None*)
Registers a new local check by http.

Parameters

- **name** (*str*) – check name
- **http** (*str*) – url to ping
- **interval** (*str*) – evaluate script every *ìnterval*
- **id** (*str*) – check id
- **notes** (*str*) – human readable notes

Returns *Check* – instance

`AgentCheckEndpoint.`**`register_script`**(*name*, *script*, *, *interval*, *id=None*, *notes=None*)
Registers a new local check by script.

Parameters

- **name** (*str*) – check name
- **script** (*str*) – path to script
- **interval** (*str*) – evaluate script every *ìnterval*
- **id** (*str*) – check id
- **notes** (*str*) – human readable notes

Returns *Check* – instance

`AgentCheckEndpoint.`**`register_ttl`**(*name*, *ttl*, *, *id=None*, *notes=None*)
Registers a new local check by ttl.

Parameters

- **name** (*str*) – check name
- **ttl** (*str*) – period status update
- **id** (*str*) – check id
- **notes** (*str*) – human readable notes

Returns *Check* – instance

AgentCheckEndpoint.**warning**(*check*, *note=None*)
Marks a local test as warning.

Parameters

- **check** (Check) – check or id
- **note** (*str*) – human readable reason

Returns *bool* – `True` check has been deregistered

class AgentServiceEndpoint(*client*)

exception NotFound
Raised when service was not found

AgentServiceEndpoint.**__call__**()
Returns the services the local agent is managing.

Returns *set* – set of *Check* instances

AgentServiceEndpoint.**create**(*name*, *, *id=None*, *tags=None*, *address=None*, *port=None*, *check=None*)
Registers a new local service.

Parameters

- **name** (*str*) – service name
- **id** (*str*) – service id
- **tags** (*list*) – service tags
- **address** (*str*) – service address
- **port** (*str*) – service port

Returns *NodeService* – instance

AgentServiceEndpoint.**delete**(*service*)
Deregister a local service.

Parameters **service** (NodeService) – service or id

Returns *bool* – `True` it has been deregistered

AgentServiceEndpoint.**deregister**(*service*)
Deregister a local service.

Parameters **service** (NodeService) – service or id

Returns *bool* – `True` it has been deregistered

AgentServiceEndpoint.**disable**(*service*, *reason=None*)
Disable service.

Parameters

- **service** (NodeService) – service or id
- **reason** (*str*) – human readable reason

Returns *bool* – `True` it has been disabled

`AgentServiceEndpoint.`**`enable`**(*service*, *reason=None*)
Enable service.

Parameters

- **`service`** (NodeService) – service or id
- **`reason`** (*str*) – human readable reason

Returns *bool* – `True` it has been enabled

`AgentServiceEndpoint.`**`get`**(*service*)
Fetch local service.

Parameters **`service`** (NodeService) – service or id

Returns *Service* – instance

Raises *`NotFound`* – service was not found

`AgentServiceEndpoint.`**`items`**()
Returns the services the local agent is managing.

Returns *set* – set of *`Check`* instances

`AgentServiceEndpoint.`**`maintenance`**(*service*, *enable*, *reason=None*)
Manages service maintenance mode.

Parameters

- **`service`** (NodeService) – service or id
- **`enable`** (*bool*) – in maintenance or not
- **`reason`** (*str*) – human readable reason

Returns *bool* – `True` all is OK

`AgentServiceEndpoint.`**`register`**(*name*, ***, *id=None*, *tags=None*, *address=None*, *port=None*, *check=None*)
Registers a new local service.

Parameters

- **`name`** (*str*) – service name
- **`id`** (*str*) – service id
- **`tags`** (*list*) – service tags
- **`address`** (*str*) – service address
- **`port`** (*str*) – service port

Returns *NodeService* – instance

`AgentServiceEndpoint.`**`register_http`**(*name*, *http*, ***, *id=None*, *tags=None*, *address=None*, *port=None*, *interval=None*)
Registers a new local service with a check by http.

Parameters

- **`name`** (*str*) – service name
- **`http`** (*str*) – url to ping
- **`interval`** (*str*) – evaluate script every *ìnterval*
- **`id`** (*str*) – service id
- **`tags`** (*list*) – service tags
- **`address`** (*str*) – service address

- **port** (*str*) – service port

Returns *NodeService* – instance

AgentServiceEndpoint.**register_script**(*name*, *script*, *, *id=None*, *tags=None*, *address=None*, *port=None*, *interval=None*)

Registers a new local service with a check by script.

Parameters

- **name** (*str*) – service name
- **script** (*str*) – path to script
- **interval** (*str*) – evaluate script every *ìnterval*
- **id** (*str*) – service id
- **tags** (*list*) – service tags
- **address** (*str*) – service address
- **port** (*str*) – service port

Returns *NodeService* – instance

AgentServiceEndpoint.**register_ttl**(*name*, *ttl*, *, *id=None*, *tags=None*, *address=None*, *port=None*)

Registers a new local service with a check by ttl.

Parameters

- **name** (*str*) – service name
- **ttl** (*str*) – period status update
- **id** (*str*) – service id
- **tags** (*list*) – service tags
- **address** (*str*) – service address
- **port** (*str*) – service port

Returns *NodeService* – instance

4.3 Catalog

The Catalog is the endpoint used to register and deregister nodes, services, and checks. It also provides query endpoints

You can also wrap next requests to the specified datacenter. The following example will fetch all nodes of *dc2*:

```
>>> sessions = yield from client.catalog.dc('dc2').nodes()
```

4.3.1 Internals

class CatalogEndpoint(*client*, *dc=None*)

Variables ***dc*** (*str*) – the datacenter

exception NotFound

Raised when a node was not found.

CatalogEndpoint.**datacenters**()

Lists datacenters

Returns *set* – a set of datacenters

CatalogEndpoint.**dc**(*name*)
Wraps requests to the specified datacenter.

Parameters **name** (*str*) – datacenter name

Returns *CatalogEndpoint* – a new endpoint

CatalogEndpoint.**deregister**(*node*, *, *check=None*, *service=None*)
Deregisters from catalog

Parameters

- **node** (Node) – node or id
- **check** (Check) – check or id
- **service** (NodeService) – service or id

Returns *bool* – `True` it is deregistered

Raises `ValidationError` – an error occured

CatalogEndpoint.**deregister_check**(*node*, *, *check*)
Deregisters a check

Parameters

- **node** (Node) – node or id
- **check** (Check) – check or id

Returns *bool* – `True` it is deregistered

Raises `ValidationError` – an error occured

CatalogEndpoint.**deregister_node**(*node*)
Deregisters a node

Parameters **node** (Node) – node or id

Returns *bool* – `True` it is deregistered

Raises `ValidationError` – an error occured

CatalogEndpoint.**deregister_service**(*node*, *, *service*)
Deregisters a service

Parameters

- **node** (Node) – node or id
- **service** (NodeService) – service or id

Returns *bool* – `True` it is deregistered

Raises `ValidationError` – an error occured

CatalogEndpoint.**get**(*node*)
Get a node. Raises a NotFound if it's not found.

The returned *Node* instance has a special attribute named *services* which holds a list of *NodeService*.

The returned objects has a special attribute named *services* which holds the *Key* informations.

Parameters **node** (*str*) – node or name

Returns *Node* – instance

Raises *NotFound* – node was not found

CatalogEndpoint.**nodes**(*, *service=None*, *tag=None*)
Lists nodes.

If service is given, *Node* instances will have a special attribute named *service*, which holds a *NodeService* instance.

Parameters

- **service** (Service) – service or id
- **tag** (*str*) – tag of service

Returns *DataSet* – set of *Node* instances

Raises ValidationError – an error occured

CatalogEndpoint.**register**(*node*, *, *check=None*, *service=None*)
Registers to catalog

CatalogEndpoint.**register_check**(*node*, *, *check*)
Registers a check

CatalogEndpoint.**register_node**(*node*)
Registers a node

CatalogEndpoint.**register_service**(*node*, *, *service*)
Registers a service

CatalogEndpoint.**services**()
Lists services.

Returns *dict* – a mapping of services - known tags

4.4 Event

The Event endpoint is used to fire new events and to query the available events.

Fires a new user event:

```
>>> event = yield from client.events.fire('my-event-b', 'my-payload')
```

Lists the most recent events an agent has seen:

```
>>> events = yield from client.events('my-event')
```

4.4.1 Internals

class EventEndpoint(*client*)

__call__(*, *event=None*)
Lists latest events.

Parameters **event** (*str*) – filter by event

Returns *set* – set of *Event*

fire(*event*, *payload*, *, *dc=None*, *node_filter=None*, *service_filter=None*, *tag_filter=None*)
Fires a new event.

Parameters

- **event** (*str*) – name of the event
- **payload** (*str*) – content to send
- **dc** (*str*) – Select a datacenter

- **node_filter** (*str*) – Filter to these nodes
- **service_filter** (*str*) – Filter to these services
- **tag_filter** (*str*) – Filter to these tags

Returns *Event* – instance

Raises `ValidationError` – an error occured

items (*, *event=None*)
Lists latest events.

Parameters **event** (*str*) – filter by event

Returns *set* – set of *Event*

4.5 Health

The Health endpoint is used to query health-related information.

Returns the health info of a node:

```
>>> checks = yield from client.health(node='my.node')
```

Returns the checks of a service:

```
>>> checks = yield from client.health(service='my.service')
```

Returns the nodes and health info of a service:

```
>>> nodes = yield from client.health.nodes(service='my.service',
>>>                                        tag='master')
```

Returns the checks in a given state:

```
>>> checks = yield from client.health(state='passing')
```

4.5.1 Internals

class HealthEndpoint (*client*)

__call__ (*, *node=None*, *service=None*, *state=None*, *dc=None*)
Returns checks filtered by node, service and state.

Parameters

- **node** (Node) – node or id
- **service** (Service) – service or id
- **state** (*str*) – check state
- **dc** (*str*) – datacenter name

Returns *DataSet* – set of *Check* instances

Raises `ValidationError` – an error occured

items (*, *node=None*, *service=None*, *state=None*, *dc=None*)
Returns checks filtered by node, service and state.

Parameters

- **node** (Node) – node or id

- **service** (Service) – service or id
- **state** (*str*) – check state
- **dc** (*str*) – datacenter name

Returns *DataSet* – set of *Check* instances

Raises ValidationError – an error occured

nodes (*service*, *, *dc=None*, *tag=None*, *state=None*)

Returns nodes by service, tag and state.

The returned *Node* instance has two special attributes:

•*service* which holds an instance of *NodeService*

•*checks* which holds instances of *Check*

Parameters

- **service** (Service) – service or id
- **dc** (*str*) – datacenter name
- **tag** (*str*) – service tag
- **state** (*str*) – passing or any

Returns *DataSet* – set of *Node* instances

4.6 KV

The KV endpoint is used to access Consul's simple key/value store, useful for storing service configuration or other metadata.

You can also wrap next requests to the specified datacenter. The following example will fetch all values of *dc2*:

```
>>> sessions = yield from client.kv.dc('dc2').items('foo/bar')
```

4.6.1 Internals

class KVEndpoint (*client*, *dc=None*)

Variables ***dc*** (*str*) – the datacenter

exception NotFound

Raised when a key was not found.

KVEndpoint.**__call__** (*path*)

Fetch values by prefix

The returned objects has a special attribute named *consul* which holds the *Key* informations.

Parameters **path** (*str*) – prefix to check

Returns *DataMapping* – mapping of key names - values

KVEndpoint.**acquire** (*path*, *, *session*)

Acquire a key

Parameters

- **path** (*str*) – the key
- **session** (Session) – session or id

Returns *bool* – key has been acquired

KVEndpoint.**dc**(*name*)
Wraps requests to the specified datacenter.

Parameters **name** (*str*) – datacenter name

Returns *KVEndpoint* – instance

KVEndpoint.**delete**(*path*, *, *recurse=None*, *cas=None*)
Deletes one or many keys.

Parameters

- **path** (*str*) – the key to delete
- **recurse** (*bool*) – delete all keys which have the specified prefix
- **cas** (*str*) – turn the delete into a Check-And-Set operation.

Returns *bool* – succeed

KVEndpoint.**get**(*path*)
Fetch one value

The returned object has a special attribute named *consul* which holds the *Key* informations.

Parameters **path** (*str*) – exact match

Returns *object* – The value corresponding to key.

Raises *NotFound* – key was not found

KVEndpoint.**items**(*path*)
Fetch values by prefix

The returned objects has a special attribute named *consul* which holds the *Key* informations.

Parameters **path** (*str*) – prefix to check

Returns *DataMapping* – mapping of key names - values

KVEndpoint.**keys**(*path*, *, *separator=None*)
Returns all keys that starts with path

Parameters

- **path** (*str*) – the key to fetch
- **separator** (*str*) – everything until

Returns *DataSet* – a set of *Key*

KVEndpoint.**put**(*path*, *value*, *, *flags=None*, *cas=None*)
Sets a key - value (lowlevel)

If the cas parameter is set, Consul will only put the key if it does not already exist. If the index is non-zero, the key is only set if the index matches the ModifyIndex of that key.

Parameters

- **path** (*str*) – the key
- **value** (*str*) – value to put
- **flags** (*int*) – flags
- **cas** (*int*) – modify_index of key
- **acquire** (*str*) – session id
- **release** (*str*) – session id

Returns *bool* – succeed

KVEndpoint.**release**(*path*, *, *session*)
Release a key

Parameters

- **path** (*str*) – the key
- **session** (Session) – session or id

Returns *bool* – key has been released

KVEndpoint.**set**(*path*, *obj*, *, *cas=None*)

Sets a key - obj

If CAS is providen, then it will acts as a Check and Set. CAS must be the ModifyIndex of that key

Parameters

- **path** (*str*) – the key
- **obj** (*object*) – any object type (will be compressed by codec)
- **cas** (*str*) – modify_index of key

Returns *bool* – value has been setted

4.7 Session

The Session endpoint is used to create, destroy, and query sessions. The following endpoints are supported.

Create a session:

```
>>> created = yield from client.sessions.create(name='foo',
>>>                                             node='my.node.name',
>>>                                             ttl='60s')
```

Fetch this session:

```
>>> session = yield from client.sessions.get(created)
>>> assert created == session  # they are the same
```

List all attached sessions of datacenter:

```
>>> sessions = yield from client.sessions()
>>> assert session in sessions  # my session is in the list
```

List all attached sessions of datacenter, but filtered by my node:

```
>>> sessions = yield from client.sessions(node='my.node.name')
```

I'm done with it, delete my session:

```
>>> deleted = yield from client.sessions.delete(session)
>>> assert deleted  # my session does not exists anymore
```

You can also wrap next requests to the specified datacenter. The following example will fetch all sessions of *dc2*:

```
>>> sessions = yield from client.sessions.dc('dc2').items()
```

4.7.1 Internals

class SessionEndpoint(*client*, *dc=None*)

exception NotFound

Raised when session was not found

SessionEndpoint.**__call__**(*, *node=None*)
List active sessions.

It will returns the active sessions for current datacenter. If node is specified, it will returns the active sessions for given node and current datacenter.

Parameters **node** (Node) – filter this node

Returns *DataSet* – a set of *Session*

SessionEndpoint.**create**(*, *name=None*, *node=None*, *checks=None*, *behavior=None*, *lock_delay=None*, *ttl=None*)
Initialize a new session.

A session can be invalidated if ttl is provided.

Parameters

- **name** (*str*) – human-readable name for the session
- **node** (*str*) – attach to this node, default to current agent
- **checks** (*list*) – associate health checks
- **behavior** (*str*) – controls the behavior when a session is invalidated
- **lock_delay** (*int*) – duration of key lock.
- **ttl** (*int*) – invalidated session until renew.

Returns *Session* – the fresh session

SessionEndpoint.**dc**(*name*)
Wraps next requests to the specified datacenter.

For example:

```
>>> sessions = yield from client.sessions.dc('dc2').items()
```

will fetch all sessions of `dc2`.

Parameters **name** (*str*) – datacenter name

Returns *SessionEndpoint* – a clone of this instance

SessionEndpoint.**delete**(*session*)
Delete session

Parameters **session** (Session) – id of the session

Returns *bool* – True

SessionEndpoint.**destroy**(*session*)
Delete session

Parameters **session** (Session) – id of the session

Returns *bool* – True

SessionEndpoint.**get**(*session*)
Returns the requested session information within datacenter.

Parameters **session** (Session) – session id

Returns *Session* – queried session

Raises *NotFound* – session was not found

SessionEndpoint.**items**(*, *node=None*)
List active sessions.

It will returns the active sessions for current datacenter. If node is specified, it will returns the active sessions for given node and current datacenter.

Parameters **node** (Node) – filter this node

Returns *DataSet* – a set of *Session*

SessionEndpoint.**renew**(*session*)

If session was created with a TTL set, it will renew this session.

Parameters **session** (Session) – the session

Returns *bool* – True

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

a

aioconsul, 8

Symbols

__call__() (ACLEndpoint method), 12
__call__() (AgentCheckEndpoint method), 14
__call__() (AgentServiceEndpoint method), 17
__call__() (EventEndpoint method), 21
__call__() (HealthEndpoint method), 22
__call__() (KVEndpoint method), 23
__call__() (SessionEndpoint method), 25

A

ACLEndpoint (class in aioconsul), 11
ACLEndpoint.NotFound, 12
acquire() (KVEndpoint method), 23
AgentCheckEndpoint (class in aioconsul), 14
AgentCheckEndpoint.NotFound, 14
AgentEndpoint (class in aioconsul), 14
AgentServiceEndpoint (class in aioconsul), 17
AgentServiceEndpoint.NotFound, 17
aioconsul (module), 7, 8, 11, 13, 19, 21–23, 25

C

CatalogEndpoint (class in aioconsul), 19
CatalogEndpoint.NotFound, 19
Check (class in aioconsul), 9
clone() (ACLEndpoint method), 12
config() (AgentEndpoint method), 14
Consul (class in aioconsul), 7
create() (ACLEndpoint method), 12
create() (AgentCheckEndpoint method), 14
create() (AgentServiceEndpoint method), 17
create() (SessionEndpoint method), 26
critical() (AgentCheckEndpoint method), 15

D

datacenters() (CatalogEndpoint method), 19
DataMapping (class in aioconsul), 10
DataSet (class in aioconsul), 10
dc() (CatalogEndpoint method), 19
dc() (KVEndpoint method), 24
dc() (SessionEndpoint method), 26
delete() (ACLEndpoint method), 12
delete() (AgentCheckEndpoint method), 15
delete() (AgentServiceEndpoint method), 17
delete() (Consul method), 7
delete() (KVEndpoint method), 24
delete() (SessionEndpoint method), 26
deregister() (AgentCheckEndpoint method), 15
deregister() (AgentServiceEndpoint method), 17
deregister() (CatalogEndpoint method), 20
deregister_check() (CatalogEndpoint method), 20
deregister_node() (CatalogEndpoint method), 20
deregister_service() (CatalogEndpoint method), 20
destroy() (ACLEndpoint method), 12
destroy() (SessionEndpoint method), 26
disable() (AgentEndpoint method), 14
disable() (AgentServiceEndpoint method), 17

E

enable() (AgentEndpoint method), 14
enable() (AgentServiceEndpoint method), 18
Event (class in aioconsul), 9
EventEndpoint (class in aioconsul), 21

F

failing() (AgentCheckEndpoint method), 15
fire() (EventEndpoint method), 21
force_leave() (AgentEndpoint method), 14

G

get() (ACLEndpoint method), 12
get() (AgentCheckEndpoint method), 15
get() (AgentServiceEndpoint method), 18
get() (CatalogEndpoint method), 20
get() (Consul method), 7
get() (KVEndpoint method), 24
get() (SessionEndpoint method), 26

H

HealthEndpoint (class in aioconsul), 22

I

is_supported() (ACLEndpoint method), 12
items() (ACLEndpoint method), 12
items() (AgentCheckEndpoint method), 15
items() (AgentServiceEndpoint method), 18
items() (EventEndpoint method), 22
items() (HealthEndpoint method), 22
items() (KVEndpoint method), 24

items() (SessionEndpoint method), 26

J

join() (AgentEndpoint method), 14

K

Key (class in aioconsul), 10
keys() (KVEndpoint method), 24
KVEndpoint (class in aioconsul), 23
KVEndpoint.NotFound, 23

M

maintenance() (AgentServiceEndpoint method), 18
mark() (AgentCheckEndpoint method), 15
me() (AgentEndpoint method), 14
Member (class in aioconsul), 8
members() (AgentEndpoint method), 14

N

Node (class in aioconsul), 8
nodes() (CatalogEndpoint method), 20
nodes() (HealthEndpoint method), 23
NodeService (class in aioconsul), 8

P

passing() (AgentCheckEndpoint method), 16
post() (Consul method), 7
put() (Consul method), 7
put() (KVEndpoint method), 24

R

register() (AgentCheckEndpoint method), 16
register() (AgentServiceEndpoint method), 18
register() (CatalogEndpoint method), 21
register_check() (CatalogEndpoint method), 21
register_http() (AgentCheckEndpoint method), 16
register_http() (AgentServiceEndpoint method), 18
register_node() (CatalogEndpoint method), 21
register_script() (AgentCheckEndpoint method), 16
register_script() (AgentServiceEndpoint method), 19
register_service() (CatalogEndpoint method), 21
register_ttl() (AgentCheckEndpoint method), 16
register_ttl() (AgentServiceEndpoint method), 19
release() (KVEndpoint method), 24
renew() (SessionEndpoint method), 27
request() (Consul method), 7
Rule (class in aioconsul), 9

S

Service (class in aioconsul), 8
services() (CatalogEndpoint method), 21
Session (class in aioconsul), 10
SessionEndpoint (class in aioconsul), 25
SessionEndpoint.NotFound, 25
set() (KVEndpoint method), 25

T

Token (class in aioconsul), 8

U

update() (ACLEndpoint method), 13

W

warning() (AgentCheckEndpoint method), 17