consulate

Release 0.6.0

July 22, 2015

Contents

1 Installation **3**

2 Requirements **5**

3 API Documentation **7**
- 3.1 Consul . . . 7
- 3.2 ACL . . . 8
- 3.3 Agent . . . 10
- 3.4 Catalog . . . 12
- 3.5 Event . . . 14
- 3.6 Health . . . 15
- 3.7 KV . . . 15
- 3.8 Session . . . 19
- 3.9 Status . . . 21

4 Version History **23**

5 Issues **25**

6 Source **27**

7 License **29**

8 Indices and tables **31**

Consulate is a Python client library and set of application for the Consul service discovery and configuration system.

CHAPTER 1

Installation

consulate may be installed via the Python package index with the tool of your choice. I prefer pip:

```
pip install consulate
```

CHAPTER 2

Requirements

- requests

CHAPTER 3

API Documentation

3.1 Consul

The *consulate.Consul* class is core interface for interacting with all parts of the Consul API.

3.1.1 Usage Examples

Here is an example where the initial *consulate.Consul* is created, connecting to Consul at `localhost` on port `8500`. Once connected, a list of all service checks is returned.

```
import consulate

# Create a new instance of a consulate session
session = consulate.Consul()

# Get all of the service checks for the local agent
checks = session.agent.checks()
```

This next example creates a new *Consul* passing in an authorization token and then sets a key in the Consul KV service:

```
import consulate

session = consulate.Consul(token='5d24c96b4f6a4aefb99602ce9b60d16b')

# Set the key named release_flag to True
session.kv['release_flag'] = True
```

3.1.2 API

class `consulate.`**`Consul`**(*host='localhost'*, *port=8500*, *datacenter=None*, *token=None*, *scheme='http'*, *adapter=None*)

Access the Consul HTTP API via Python.

The default values connect to Consul via `localhost:8500` via http. If you want to connect to Consul via a local UNIX socket, you'll need to override both the `scheme`, `port` and the `adapter` like so:

```
consul = consulate.Consul('/path/to/socket', None, scheme='http+unix',
                          adapter=consulate.adapters.UnixSocketRequest)
services = consul.agent.services()
```

Parameters

- **host** (*str*) – The host name to connect to (Default: localhost)
- **port** (*int*) – The port to connect on (Default: 8500)
- **datacenter** (*str*) – Specify a specific data center
- **token** (*str*) – Specify a ACL token to use
- **scheme** (*str*) – Specify the scheme (Default: http)
- **adapter** (*class*) – Specify to override the request adapter (Default: `consulate.adapters.Request`)

acl
Access the Consul ACL API

Return type *consulate.api.acl.ACL*

agent
Access the Consul Agent API

Return type *consulate.api.agent.Agent*

catalog
Access the Consul Catalog API

Return type *consulate.api.catalog.Catalog*

event
Access the Consul Events API

Return type *consulate.api.event.Event*

health
Access the Consul Health API

Return type *consulate.api.health.Health*

kv
Access the Consul KV API

Return type *consulate.api.kv.KV*

session
Access the Consul Session API

Return type *consulate.api.session.Session*

status
Access the Consul Status API

Return type *consulate.api.status.Status*

3.2 ACL

class `consulate.api.acl.`**ACL**(*uri*, *adapter*, *datacenter=None*, *token=None*)
The ACL endpoints are used to create, update, destroy, and query ACL tokens.

clone(*acl_id*)
Clone an existing ACL returning the new ACL ID

Parameters **acl_id** (*str*) – The ACL id

Return type bool

Raises consulate.exceptions.Forbidden

Raises consulate.exceptions.NotFound

create(*name*, *acl_type='client'*, *rules=None*)

The create endpoint is used to make a new token. A token has a name, a type, and a set of ACL rules.

The `name` property is opaque to Consul. To aid human operators, it should be a meaningful indicator of the ACL's purpose.

`acl_type` is either client or management. A management token is comparable to a root user and has the ability to perform any action including creating, modifying, and deleting ACLs.

By contrast, a client token can only perform actions as permitted by the rules associated. Client tokens can never manage ACLs. Given this limitation, only a management token can be used to make requests to the create endpoint.

`rules` is a HCL string defining the rule policy. See **'https://consul.io/docs/internals/acl.html'_** for more information on defining rules.

The call to create will return the ID of the new ACL.

Parameters

- **name** (*str*) – The name of the ACL to create
- **acl_type** (*str*) – One of "client" or "management"
- **rules** (*str*) – The rules HCL string

Return type str

Raises consulate.exceptions.Forbidden

destroy(*acl_id*)

Delete the specified ACL

Parameters **acl_id** (*str*) – The ACL id

Return type bool

Raises consulate.exceptions.Forbidden

Raises consulate.exceptions.NotFound

info(*acl_id*)

Return a dict of information about the ACL

Parameters **acl_id** (*str*) – The ACL id

Return type dict

Raises consulate.exceptions.Forbidden

Raises consulate.exceptions.NotFound

list()

Return a list of all ACLs

Return type *list*

Raises consulate.exceptions.Forbidden

update(*acl_id*, *name*, *acl_type='client'*, *rules=None*)

Update an existing ACL, updating its values or add a new ACL if the ACL Id specified is not found.

Parameters **acl_id** (*str*) – The ACL id

Return type bool

Raises consulate.exceptions.Forbidden

3.3 Agent

class consulate.api.agent.**Agent**(*uri*, *adapter*, *datacenter=None*, *token=None*)

The Consul agent is the core process of Consul. The agent maintains membership information, registers services, runs checks, responds to queries and more. The agent must run on every node that is part of a Consul cluster.

class **Check**(*uri*, *adapter*, *datacenter=None*, *token=None*)

One of the primary roles of the agent is the management of system and application level health checks. A health check is considered to be application level if it associated with a service. A check is defined in a configuration file, or added at runtime over the HTTP interface.

There are two different kinds of checks:

- **Script + Interval: These checks depend on invoking an external** application which does the health check and exits with an appropriate exit code, potentially generating some output. A script is paired with an invocation interval (e.g. every 30 seconds). This is similar to the Nagios plugin system.
- **TTL: These checks retain their last known state for a given TTL.** The state of the check must be updated periodically over the HTTP interface. If an external system fails to update the status within a given TTL, the check is set to the failed state. This mechanism is used to allow an application to directly report it's health. For example, a web app can periodically curl the endpoint, and if the app fails, then the TTL will expire and the health check enters a critical state. This is conceptually similar to a dead man's switch.

deregister(*check_id*)

Remove a check from the local agent. The agent will take care of deregistering the check with the Catalog.

Parameters **check_id** (*str*) – The check id

register(*name*, *script=None*, *check_id=None*, *interval=None*, *ttl=None*, *notes=None*, *http=None*)

Add a new check to the local agent. Checks are either a script or TTL type. The agent is responsible for managing the status of the check and keeping the Catalog in sync.

The `name` field is mandatory, as is either `script` and `interval`, `http` and `interval` or `ttl`. Only one of `script` and `interval`, `http` and `interval` or `ttl` should be provided. If an `check_id` is not provided, it is set to `name`. You cannot have duplicate `check_id` entries per agent, so it may be necessary to provide a `check_id`. The `notes` field is not used by Consul, and is meant to be human readable.

If a `script` is provided, the check type is a script, and Consul will evaluate the script every `interval` to update the status. If a `http` URL is provided, Consul will poll the URL every `interval` to update the status - only 2xx results are considered healthy. If a `ttl` type is used, then the `ttl` update APIs must be used to periodically update the state of the check.

Parameters

- **name** (*str*) – The check name
- **http** (*str*) – The URL to poll for health checks
- **script** (*str*) – The path to the script to run
- **check_id** (*str*) – The optional check id
- **interval** (*int*) – The interval to run the check

- **ttl** (*int*) – The ttl to specify for the check
- **notes** (*str*) – Administrative notes.

Return type bool
Raises ValueError

ttl_fail(*check_id*)
This endpoint is used with a check that is of the TTL type. When this endpoint is accessed, the status of the check is set to "critical", and the TTL clock is reset.
Parameters **check_id** (*str*) – The check id

ttl_pass(*check_id*)
This endpoint is used with a check that is of the TTL type. When this endpoint is accessed, the status of the check is set to "passing", and the TTL clock is reset.
Parameters **check_id** (*str*) – The check id

ttl_warn(*check_id*)
This endpoint is used with a check that is of the TTL type. When this endpoint is accessed, the status of the check is set to "warning", and the TTL clock is reset.
Parameters **check_id** (*str*) – The check id

class Agent.**Service**(*uri*, *adapter*, *datacenter=None*, *token=None*)
One of the main goals of service discovery is to provide a catalog of available services. To that end, the agent provides a simple service definition format to declare the availability of a service, a nd to potentially associate it with a health check. A health check is considered to be application level if it associated with a service. A service is defined in a configuration file, or added at runtime over the HTTP interface.

deregister(*service_id*)
Deregister the service from the local agent. The agent will take care of deregistering the service with the Catalog. If there is an associated check, that is also deregistered.
Parameters **service_id** (*str*) – The service id to deregister
Return type bool

register(*name*, *service_id=None*, *address=None*, *port=None*, *tags=None*, *check=None*, *interval=None*, *ttl=None*, *httpcheck=None*)
Add a new service to the local agent.
Parameters

- **name** (*str*) – The name of the service
- **service_id** (*str*) – The id for the service (optional)
- **address** (*str*) – The service IP address
- **port** (*int*) – The service port
- **tags** (list) – A list of tags for the service
- **check** (*str*) – The path to the check script to run
- **interval** (*str*) – The check execution interval
- **ttl** (*str*) – The TTL for external script check pings
- **httpcheck** (*str*) – An URL to check every interval

Return type bool
Raises ValueError

Agent.**__init__**(*uri*, *adapter*, *datacenter=None*, *token=None*)
Create a new instance of the Agent class

Parameters

- **uri** (*str*) – Base URI
- **adapter** (*consul.adapters.Request*) – Request adapter
- **datacenter** (*str*) – datacenter
- **token** (*str*) – Access Token

Agent.**checks**()
return the all the checks that are registered with the local agent. These checks were either provided through configuration files, or added dynamically using the HTTP API. It is important to note that the checks known by the agent may be different than those reported by the Catalog. This is usually due to changes being made while there is no leader elected. The agent performs active anti-entropy, so in most situations everything will be in sync within a few seconds.

Return type *list*

Agent.**force_leave**(*node*)
Instructs the agent to force a node into the left state. If a node fails unexpectedly, then it will be in a "failed" state. Once in this state, Consul will attempt to reconnect, and additionally the services and checks belonging to that node will not be cleaned up. Forcing a node into the left state allows its old entries to be removed.

Agent.**join**(*address*, *wan=False*)
This endpoint is hit with a GET and is used to instruct the agent to attempt to connect to a given address. For agents running in server mode, setting wan=True causes the agent to attempt to join using the WAN pool.

Parameters

- **address** (*str*) – The address to join
- **wan** (*bool*) – Join a WAN pool as a server

Return type bool

Agent.**members**()
Returns the members the agent sees in the cluster gossip pool. Due to the nature of gossip, this is eventually consistent and the results may differ by agent. The strongly consistent view of nodes is instead provided by `Consulate.catalog.nodes`.

Return type *list*

Agent.**services**()
return the all the services that are registered with the local agent. These services were either provided through configuration files, or added dynamically using the HTTP API. It is important to note that the services known by the agent may be different than those]reported by the Catalog. This is usually due to changes being made while there is no leader elected. The agent performs active anti-entropy, so in most situations everything will be in sync within a few seconds.

Return type *list*

3.4 Catalog

class consulate.api.catalog.**Catalog**(*uri*, *adapter*, *dc=None*, *token=None*)
The Consul agent is the core process of Consul. The agent maintains membership information, registers services, runs checks, responds to queries and more. The agent must run on every node that is part of a Consul cluster.

datacenters()
Return all the datacenters that are known by the Consul server.

Return type *list*

deregister(*node*, *datacenter=None*, *check_id=None*, *service_id=None*)
Directly remove entries in the catalog. It is usually recommended to use the agent local endpoints, as they are simpler and perform anti-entropy.

The behavior of the endpoint depends on what keys are provided. The endpoint requires `node` to be provided, while `datacenter` will be defaulted to match that of the agent. If only `node` is provided, then the node, and all associated services and checks are deleted. If `check_id` is provided, only that check belonging to the node is removed. If `service_id` is provided, then the service along with it's associated health check (if any) is removed.

Parameters

- **node** (*str*) – The node for the action
- **datacenter** (*str*) – The optional datacenter for the node
- **check_id** (*str*) – The optional check_id to remove
- **service_id** (*str*) – The optional service_id to remove

Return type bool

node(*node_id*)

Return the node data for the specified node

Parameters **node_id** (*str*) – The node ID

Return type dict

nodes()

Return all of the nodes for the current datacenter.

Return type *list*

register(*node*, *address*, *datacenter=None*, *service=None*, *check=None*)

A a low level mechanism for directly registering or updating entries in the catalog. It is usually recommended to use the agent local endpoints, as they are simpler and perform anti-entropy.

The behavior of the endpoint depends on what keys are provided. The endpoint requires Node and Address to be provided, while Datacenter will be defaulted to match that of the agent. If only those are provided, the endpoint will register the node with the catalog.

If the Service key is provided, then the service will also be registered. If ID is not provided, it will be defaulted to Service. It is mandated that the ID be node-unique. Both Tags and Port can be omitted.

If the Check key is provided, then a health check will also be registered. It is important to remember that this register API is very low level. This manipulates the health check entry, but does not setup a script or TTL to actually update the status. For that behavior, an agent local check should be setup.

The CheckID can be omitted, and will default to the Name. Like before, the CheckID must be node-unique. The Notes is an opaque field that is meant to hold human readable text. If a ServiceID is provided that matches the ID of a service on that node, then the check is treated as a service level health check, instead of a node level health check. Lastly, the status must be one of "unknown", "passing", "warning", or "critical". The "unknown" status is used to indicate that the initial check has not been performed yet.

It is important to note that Check does not have to be provided with Service and visa-versa. They can be provided or omitted at will.

Example service dict:

```
'Service': {
    'ID': 'redis1',
    'Service': 'redis',
    'Tags': ['master', 'v1'],
    'Port': 8000,
}
```

Example check dict:

```
        'Check': {
            'Node': 'foobar',
            'CheckID': 'service:redis1',
            'Name': 'Redis health check',
            'Notes': 'Script based health check',
            'Status': 'passing',
            'ServiceID': 'redis1'
        }
```

Parameters

- **node** (*str*) – The node name
- **address** (*str*) – The node address
- **datacenter** (*str*) – The optional node datacenter
- **service** (*dict*) – An optional node service
- **check** (*dict*) – An optional node check

Return type bool

service(*service_id*)
Return the service details for the given service

Parameters **service_id** (*str*) – The service id

Return type *list*

services()
Return a list of all of the services for the current datacenter.

Return type *list*

3.5 Event

class consulate.api.event.**Event**(*uri*, *adapter*, *datacenter=None*, *token=None*)
The Event endpoints are used to fire a new event and list recent events.

fire(*name*, *payload=None*, *datacenter=None*, *node=None*, *service=None*, *tag=None*)
Trigger a new user Event

Parameters

- **name** (*str*) – The name of the event
- **payload** (*str*) – The opaque event payload
- **datacenter** (*str*) – Optional datacenter to fire the event in
- **node** (*str*) – Optional node to fire the event for
- **service** (*str*) – Optional service to fire the event for
- **tag** (*str*) – Option tag to fire the event for

Return str the new event ID

list(*name=None*)
Returns the most recent events known by the agent. As a consequence of how the event command works,

each agent may have a different view of the events. Events are broadcast using the gossip protocol, so they have no global ordering nor do they make a promise of delivery.

Returns list

3.6 Health

class consulate.api.health.**Health**(*uri*, *adapter*, *datacenter=None*, *token=None*)

Used to query health related information. It is provided separately from the Catalog, since users may prefer to not use the health checking mechanisms as they are totally optional. Additionally, some of the query results from the Health system are filtered, while the Catalog endpoints provide the raw entries.

checks(*service_id*)

Return checks for the given service.

Parameters **service_id** (*str*) – The service ID

Return type *list*

node(*node_id*)

Return the health info for a given node.

Parameters **node_id** (*str*) – The node ID

Return type *list*

service(*service_id*, *tag=None*, *passing=None*)

Returns the nodes and health info of a service

Parameters **service_id** (*str*) – The service ID

Return type *list*

state(*state*)

Returns the checks in a given state where state is one of "unknown", "passing", "warning", or "critical".

Parameters **state** (*str*) – The state to get checks for

Return type *list*

3.7 KV

The *KV* class provides both high and low level access to the Consul Key/Value service. To use the *KV* class, access the *consulate.Consul.kv()* attribute of the *Session* class.

For high-level operation, the *KV* class behaves like a standard Python dict. You can get, set, and delete items in the Key/Value service just as you would with a normal dictionary.

If you need to have access to the full record associated with an item, there are lower level methods such as *KV.set_record* and *KV.set_record*. These two methods provide access to the other fields associated with the item in Consul, including the flag and various index related fields.

3.7.1 Examples of Use

Here's a big blob of example code that uses most of the functionality in the KV class. Check the comments in the code to see what part of the class it is demonstrating.

```
import consulate

session = consulate.Session()

# Set the key named release_flag to True
session.kv['release_flag'] = True

# Get the value for the release_flag, if not set, raises AttributeError
try:
    should_release_feature = session.kv['release_flag']
except AttributeError:
    should_release_feature = False

# Delete the release_flag key
del session.kv['release_flag']

# Fetch how many rows are set in the KV store
print(len(self.session.kv))

# Iterate over all keys in the kv store
for key in session.kv:
    print('Key "{0}" set'.format(key))

# Iterate over all key/value pairs in the kv store
for key, value in session.kv.iteritems():
    print('{0}: {1}'.format(key, value))

# Iterate over all keys in the kv store
for value in session.kv.values:
    print(value)

# Find all keys that start with "fl"
for key in session.kv.find('fl'):
    print('Key "{0}" found'.format(key))

# Check to see if a key called "foo" is set
if "foo" in session.kv:
    print 'Already Set'

# Return all of the items in the key/value store
session.kv.items()
```

3.7.2 API

class `consulate.api.kv.`**KV**(*uri*, *adapter*, *datacenter=None*, *token=None*)

The `consul.api.KV` class implements a `dict` like interface for working with the Key/Value service. Simply use items on the `consulate.Session` like you would with a `dict` to `get`, `set`, or `delete` values in the key/value store.

Additionally, `KV` acts as an `iterator`, providing methods to iterate over `keys`, `values`, `keys and values`, etc.

Should you need access to get or set the flag value, the `get_record`, `set_record`, and `records` provide a way to access the additional fields exposed by the KV service.

__contains__(*item*)

Return True if there is a value set in the Key/Value service for the given key.

Parameters **item** (*str*) – The key to check for

Return type bool

__delitem__(*item*)
Delete an item from the Key/Value service

Parameters **item** (*str*) – The key name

__getitem__(*item*)
Get a value from the Key/Value service, returning it fully decoded if possible.

Parameters **item** (*str*) – The item name

Return type mixed

Raises KeyError

__iter__()
Iterate over all the keys in the Key/Value service

Return type iterator

__len__()
Return the number if items in the Key/Value service

Returns int

__setitem__(*item*, *value*)
Set a value in the Key/Value service, using the CAS mechanism to ensure that the set is atomic. If the value passed in is not a string, an attempt will be made to JSON encode the value prior to setting it.

Parameters

- **item** (*str*) – The key to set
- **value** (*mixed*) – The value to set

Raises KeyError

acquire_lock(*item*, *session*)
Use Consul for locking by specifying the item/key to lock with and a session value for removing the lock.

Parameters

- **item** (*str*) – The item in the Consul KV database
- **session** (*str*) – The session value for the lock

Returns bool

delete(*item*, *recurse=False*)
Delete an item from the Key/Value service

Parameters

- **item** (*str*) – The item key
- **recurse** (*bool*) – Remove keys prefixed with the item pattern

Raises KeyError

find(*prefix*, *separator=None*)
Find all keys with the specified prefix, returning a dict of matches.

Example:

```
>>> consul.kv.find('b')
{'baz': 'qux', 'bar': 'baz'}
```

Parameters **prefix** (*str*) – The prefix to search with

Return type dict

get(*item*, *default=None*, *raw=False*)
Get a value from the Key/Value service, returning it fully decoded if possible.

Parameters **item** (*str*) – The item key

Return type mixed

Raises KeyError

get_record(*item*)
Get the full record from the Key/Value service, returning all fields including the flag.

Parameters **item** (*str*) – The item key

Return type dict

Raises KeyError

items()
Return a dict of all of the key/value pairs in the Key/Value service

Example:

```
>>> consul.kv.items()
{'foo': 'bar', 'bar': 'baz', 'quz': True, 'corgie': 'dog'}
```

Return type dict

iteritems()
Iterate over the dict of key/value pairs in the Key/Value service

Example:

```
>>> for key, value in consul.kv.iteritems():
...     print(key, value)
...
(u'bar', 'baz')
(u'foo', 'bar')
(u'quz', True)
```

Return type iterator

keys()
Return a list of all of the keys in the Key/Value service

Example:

```
>>> consul.kv.keys()
[u'bar', u'foo', u'quz']
```

Return type *list*

records()
Return a list of tuples for all of the records in the Key/Value service

Example:

```
>>> consul.kv.records()
[(u'bar', 0, 'baz'),
 (u'corgie', 128, 'dog'),
 (u'foo', 0, 'bar'),
 (u'quz', 0, True)]
```

Return type list of (Key, Flags, Value)

release_lock(*item*, *session*)
Release an existing lock from the Consul KV database.

Parameters

- **item** (*str*) – The item in the Consul KV database
- **session** (*str*) – The session value for the lock

Returns bool

set(*item*, *value*)
Set a value in the Key/Value service, using the CAS mechanism to ensure that the set is atomic. If the value passed in is not a string, an attempt will be made to JSON encode the value prior to setting it.

Parameters

- **item** (*str*) – The key to set
- **value** (*mixed*) – The value to set

Raises KeyError

set_record(*item*, *flags=0*, *value=None*, *replace=True*)
Set a full record, including the item flag

Parameters

- **item** (*str*) – The key to set
- **value** (*mixed*) – The value to set
- **replace** – If True existing value will be overwritten:

values()
Return a list of all of the values in the Key/Value service

Example:

```
>>> consul.kv.values()
[True, 'bar', 'baz']
```

Return type *list*

3.8 Session

class consulate.api.session.**Session**(*uri*, *adapter*, *datacenter=None*, *token=None*)
Create, destroy, and query Consul sessions.

create(*name=None*, *behavior='release'*, *node=None*, *delay=None*, *ttl=None*, *checks=None*)
Initialize a new session.

None of the fields are mandatory, and in fact no body needs to be PUT if the defaults are to be used.

Name can be used to provide a human-readable name for the Session.

Behavior can be set to either `release` or `delete`. This controls the behavior when a session is invalidated. By default, this is release, causing any locks that are held to be released. Changing this to delete causes any locks that are held to be deleted. delete is useful for creating ephemeral key/value entries.

Node must refer to a node that is already registered, if specified. By default, the agent's own node name is used.

LockDelay (`delay`) can be specified as a duration string using a "s" suffix for seconds. The default is 15s.

The TTL field is a duration string, and like LockDelay it can use "s" as a suffix for seconds. If specified, it must be between 10s and 3600s currently. When provided, the session is invalidated if it is not renewed before the TTL expires. See the session internals page for more documentation of this feature.

Checks is used to provide a list of associated health checks. It is highly recommended that, if you override this list, you include the default "serfHealth".

Parameters

- **name** (*str*) – A human readable session name
- **behavior** (*str*) – One of `release` or `delete`
- **node** (*str*) – A node to create the session on
- **delay** (*str*) – A lock delay for the session
- **ttl** (*str*) – The time to live for the session
- **checks** (*lists*) – A list of associated health checks

Return str session ID

destroy(*session_id*)
Destroy an existing session

Parameters **session_id** (*str*) – The session to destroy

Returns bool

info(*session_id*)
Returns the requested session information within a given dc. By default, the dc of the agent is queried.

Parameters **session_id** (*str*) – The session to get info about

Returns dict

list()
Returns the active sessions for a given dc.

Returns list

node(*node*)
Returns the active sessions for a given node and dc. By default, the dc of the agent is queried.

Parameters **node** (*str*) – The node to get active sessions for

Returns list

renew(*session_id*)
Renew the given session. This is used with sessions that have a TTL, and it extends the expiration by the TTL. By default, the dc of the agent is queried.

Parameters **session_id** (*str*) – The session to renew

Returns dict

3.9 Status

class consulate.api.status.**Status**(*uri*, *adapter*, *datacenter=None*, *token=None*)
Get information about the status of the Consul cluster. This are generally very low level, and not really useful for clients.

leader()
Get the Raft leader for the datacenter the agent is running in.

Return type str

peers()
Get the Raft peers for the datacenter the agent is running in.

Return type *list*

CHAPTER 4

Version History

See `history`

CHAPTER 5

Issues

Please report any issues to the Github project at https://github.com/gmr/consulate/issues

CHAPTER 6

Source

consulate source is available on Github at https://github.com/gmr/consulate

CHAPTER 7

License

consulate is released under the 3-Clause BSD license.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Symbols

__contains__() (consulate.api.kv.KV method), 16
__delitem__() (consulate.api.kv.KV method), 17
__getitem__() (consulate.api.kv.KV method), 17
__init__() (consulate.api.agent.Agent method), 11
__iter__() (consulate.api.kv.KV method), 17
__len__() (consulate.api.kv.KV method), 17
__setitem__() (consulate.api.kv.KV method), 17

A

ACL (class in consulate.api.acl), 8
acl (consulate.Consul attribute), 8
acquire_lock() (consulate.api.kv.KV method), 17
Agent (class in consulate.api.agent), 10
agent (consulate.Consul attribute), 8
Agent.Check (class in consulate.api.agent), 10
Agent.Service (class in consulate.api.agent), 11

C

Catalog (class in consulate.api.catalog), 12
catalog (consulate.Consul attribute), 8
checks() (consulate.api.agent.Agent method), 11
checks() (consulate.api.health.Health method), 15
clone() (consulate.api.acl.ACL method), 8
Consul (class in consulate), 7
create() (consulate.api.acl.ACL method), 9
create() (consulate.api.session.Session method), 19

D

datacenters() (consulate.api.catalog.Catalog method), 12
delete() (consulate.api.kv.KV method), 17
deregister() (consulate.api.agent.Agent.Check method), 10
deregister() (consulate.api.agent.Agent.Service method), 11
deregister() (consulate.api.catalog.Catalog method), 12
destroy() (consulate.api.acl.ACL method), 9
destroy() (consulate.api.session.Session method), 20

E

Event (class in consulate.api.event), 14
event (consulate.Consul attribute), 8

F

find() (consulate.api.kv.KV method), 17
fire() (consulate.api.event.Event method), 14
force_leave() (consulate.api.agent.Agent method), 12

G

get() (consulate.api.kv.KV method), 18
get_record() (consulate.api.kv.KV method), 18

H

Health (class in consulate.api.health), 15
health (consulate.Consul attribute), 8

I

info() (consulate.api.acl.ACL method), 9
info() (consulate.api.session.Session method), 20
items() (consulate.api.kv.KV method), 18
iteritems() (consulate.api.kv.KV method), 18

J

join() (consulate.api.agent.Agent method), 12

K

keys() (consulate.api.kv.KV method), 18
KV (class in consulate.api.kv), 16
kv (consulate.Consul attribute), 8

L

leader() (consulate.api.status.Status method), 21
list() (consulate.api.acl.ACL method), 9
list() (consulate.api.event.Event method), 14
list() (consulate.api.session.Session method), 20

M

members() (consulate.api.agent.Agent method), 12

N

node() (consulate.api.catalog.Catalog method), 13

node() (consulate.api.health.Health method), 15
node() (consulate.api.session.Session method), 20
nodes() (consulate.api.catalog.Catalog method), 13

P

peers() (consulate.api.status.Status method), 21

R

records() (consulate.api.kv.KV method), 18
register() (consulate.api.agent.Agent.Check method), 10
register() (consulate.api.agent.Agent.Service method), 11
register() (consulate.api.catalog.Catalog method), 13
release_lock() (consulate.api.kv.KV method), 19
renew() (consulate.api.session.Session method), 20

S

service() (consulate.api.catalog.Catalog method), 14
service() (consulate.api.health.Health method), 15
services() (consulate.api.agent.Agent method), 12
services() (consulate.api.catalog.Catalog method), 14
Session (class in consulate.api.session), 19
session (consulate.Consul attribute), 8
set() (consulate.api.kv.KV method), 19
set_record() (consulate.api.kv.KV method), 19
state() (consulate.api.health.Health method), 15
Status (class in consulate.api.status), 21
status (consulate.Consul attribute), 8

T

ttl_fail() (consulate.api.agent.Agent.Check method), 11
ttl_pass() (consulate.api.agent.Agent.Check method), 11
ttl_warn() (consulate.api.agent.Agent.Check method), 11

U

update() (consulate.api.acl.ACL method), 9

V

values() (consulate.api.kv.KV method), 19