DynamoDB Config Store Documentation

Release 0.1.0a1

Sebastian Dahlgren

Sep 27, 2017

Contents

CHAPTER 1

Using `pip`

DynamoDB Config Store is easiest installed via `pip`.

```
pip install dynamodb-config-store
```

CHAPTER 2

Copy the code from GitHub

You can also download the latest version of the project directly from the project GitHub page.

CHAPTER 3

Usage

Instanciate a Store

A store can be instanciated like this:

```
from dynamodb_config_store

store = DynamoDBConfigStore(
    connection,                 # boto.dynamodb2.layer1.DynamoDBConnection
    table_name,                 # Table name to use
    store_name,                 # Store name to use
    store_key='_store',         # (Optional) value to store Store name in
    option_key='_option')       # (Optional) value to store Option name in
```

When the Store is instanciated it will create the table if it does not exist. Currently a table with 1 read unit and 1 write unit will be created.

If the Store is instanciated towards an existing table, it will check that the table schema matches the expected Store schema. In other words it will check that `store_key` is the table hash key and that `option_key` is the table range key.

Writing configuration

You can insert new items in the configuration via the `set` method.

```
# 'option' is the name of the option where you want to store the values
store.set('option', {'key1': 'value', 'key2': 'value'})
```

`set` takes an `option` (`str`) and a `dict` with keys and values.

Reading configuration

DynamoDB Config Store supports a few different ways to retrieve configuration. Currently we support:

- Time based config store (`TimeBasedConfigStore`)
- Simple config store (`SimpleConfigStore`)

The **Time based config store** read data from DynamoDB on a schedule. That approach reduces the read unit consumption, but is not "strongly" consistent as all configuration updates are not reflect until the next config reload.

The **Simple config store** will always query DynamoDB for the latest configuration option. This behavior will consume more reads, but in return you'll always get the latest configuration back.

SimpleConfigStore

The implementations documented below here will always fetch the configuration options directly from DynamoDB. For each *get()* below an read towards DynamoDB will be executed.

Read a specific Option

Specific Options can be fetched like this:

```
store.config.get('option')
```

You will get a `dict` with all Keys related to the given Option:

```
{
    'key1': 'value1',
    'key2': 'value2',
    'key3': 'value3',
    'key4': 'value4'
}
```

Get specific Keys from an Option

If you don't need all Keys in the response, you can select which Keys you want to retrieve by adding the `keys` parameter:

```
store.config.get('option', keys=['key1', 'key4'])
```

Returns:

```
{
    'key1': 'value1',
    'key4': 'value4'
}
```

Fetching all Options in a Store

If you want to retrieve all Options within a Store, simply omit the `option` in the request:

```
store.config.get()
```

You will get an dictionary like this in return:

```
{
    'option1': {
        'key1': 'value1',
        'key2': 'value2'
    },
    'option2': {
        'key1': 'value1',
        'key4': 'value2'
    }
}
```

Load the SimpleConfigStore

The `SimpleConfigStore` is enabled by default, but you can explicitly load it like this:

```
from dynamodb_config_store

store = DynamoDBConfigStore(
    connection,
    table_name,
    store_name,
    config_store='SimpleConfigStore')
```

TimeBasedConfigStore

The recommended (but not the default) way to read configuration is to let DynamoDB Config Store store all your configuration in an object from which you can fetch the latest configuration. If you have an option called `db`, you would access that as

```
store.config.db
```

And you would get a `dict` in return:

```
{'host': '127.0.0.1', 'port': Decimal(8000)}
```

By default DynamoDB Config Store will re-read all configuration from DynamoDB every 5 minutes (300 seconds). Any changes in the configuration after an update will not be reflected in the configuration object until the next update has been executed.

The benefit with this over the *Reading configuration directly from DynamoDB* approach is that you will consume much less read capacity. The downside, however, is that the configuration is not always up to date.

Load the TimeBasedConfigStore

You can start using the `TimeBasedConfigStore` by calling `DynamoDBConfigStore` like this:

```
from dynamodb_config_store

store = DynamoDBConfigStore(
```

```
    connection,
    table_name,
    store_name,
    config_store='TimeBasedConfigStore')
```

Read an Option

You can fetch an Option like this:

```
store.config.option
```

Where `option` is the name of your Option.

Force config update

You can manually force a configuration update by issuing:

```
store.reload()
```

Set update interval

You can set the update interval when instanciating DynamoDB Config Store:

```
from dynamodb_config_store

store = DynamoDBConfigStore(
    connection,
    table_name,
    store_name,
    config_store='TimeBasedConfigStore',
    config_store_kwargs={'auto_reload': 60})
```

This will set the update interval to 60 seconds.

Table management

DynamoDB Config Store will automatically create a new DynamoDB table if the configured table does not exist. The new table will be provisioned with 1 read unit and 1 write unit. If you want another provisioning, please supply the `read_units` and `write_units` parameters when instanciating `DynamoDBConfigStore`, e.g:

```
store = DynamoDBConfigStore(
    'table_name',
    'store_name',
    read_units=10,
    write_units=5)
```

If the table already exists when `DynamoDBConfigStore` is instanciated, then the table will be left intact. DynamoDB Config Store will check that the table schema is compatible with the configuration. That is; it will check that the hash key is `store_key` and the `option_key` is the range key. An `MisconfiguredSchemaException` will be raised if the table schema is not correct.

CHAPTER 4

API documentation

This is the API documentation for DynamoDB Config Store.

DynamoDBConfigStore

class dynamodb_config_store.**DynamoDBConfigStore**(*connection*, *table_name*, *store_name*, *store_key='_store'*, *option_key='_option'*, *read_units=1*, *write_units=1*, *config_store='SimpleConfigStore'*, *config_store_args=[]*, *config_store_kwargs={}*)

DynamoDB Config Store instance

_create_table(*read_units=1*, *write_units=1*)

Create a new table

Parameters

- **read_units** (*int*) – Number of read capacity units to provision
- **write_units** (*int*) – Number of write capacity units to provision

Returns bool – Returns True if the table was created

_initialize_store()

Initialize the store to use

_initialize_table()

Initialize the table

Returns None

_wait_for_table(*target_state*, *sleep_time=5*, *retries=30*)

Wait for the table to get to a certain state

Parameters

- **target_state** (*str*) – The target state to wait for
- **sleep_time** (*int*) – Number of seconds to wait between the checks
- **retries** (*int*) – Number of retries before giving up

Returns bool – True if the target state was reached, else False

reload()
Reload the config store

Returns None

set(*option*, *data*)
Upsert a config item

A write towards DynamoDB will be executed when this method is called.

Parameters

- **option** (*str*) – Name of the configuration option
- **data** (*dict*) – Dictionary with all option data

Returns bool – True if the data was stored successfully

ConfigStores

ConfigStore

This is the `ConfigStore` base class that all other config stores inherit from.

class dynamodb_config_store.config_stores.**ConfigStore**
Base class for config stores

_delete_instance_attributes()
Delete all the instance attributes

Returns None

_set_instance_attributes(*options*)
Set instance attributes

Parameters **options** (*dict*) – Dict with {'option': {'key': 'value'}}

Returns None

DirectConfigStore

The `DirectConfigStore` is the default Config Store. All requests made will be send directly to DynamoDB so that the latest configuration is fetched at all times.

TimeBasedConfigStore

The `TimeBasedConfigStore` is a Config Store used for fetching configuration from DynamoDB on a preset interval. All configuration options will be exposed using instance attributes such as `store.config.option`, where `option` is the store option.

This class shall not be instanciated directly, it's called by `DynamoDBConfigStore`.

class dynamodb_config_store.config_stores.time_based.**TimeBasedConfigStore**(*table*, *store_name*, *store_key*, *option_key*, *update_interval=300*)

Fetches Stores on a periodic interval from DynamoDB

All internal variables and methods are private (with an leading _), as the configuration options from DynamoDB will be set as instance attributes.

_fetch_options()

Retrieve a dictionary with all options and values from DynamoDB

Returns dict – Dict with {'option': {'key': 'value'}}

_run()

Run periodic fetcher

Returns None

Exceptions

Exception definitions

exception dynamodb_config_store.exceptions.**MisconfiguredSchemaException**

Exception thrown if the table does not match the configuration

exception dynamodb_config_store.exceptions.**TableNotCreatedException**

Exception thrown if the table could not be created successfull

exception dynamodb_config_store.exceptions.**TableNotReadyException**

Exception thrown if the table is not in ACTIVE or UPDATING state

CHAPTER 5

Release notes

0.2.2 (2014-06-28)

- Address common DynamoDB exceptions (#17)

0.2.1 (2014-06-28)

- Throw exception if not implemented config store is used (#16)
- Renamed `store_type` parameter to `config_store`

0.2.0 (2014-06-27)

- Modular config store design implemented (#15)
- Implement TimeBasedConfigStore for automatic config reloading (#14)

0.1.1 (2014-06-26)

- Fixed broken reference to license in PyPI script (#13)

0.1.0 (2014-06-26)

- Initial release of DynamoDB Config Store
- Support for `get()`
- Support for `get_object()`

- Support for `set()`
- Table schema is validated upon instanciation
- Tables are created if the do not exist
- Full test suite implemented

CHAPTER 6

Bugs and feature requests

Please report any found bugs or file feature requests at our GitHub issues page.

CHAPTER 7

Contributing

Creating pull requests

If you want to open a pull request, please do it towards the `develop` branch. I'd also appreciate if the pull request contains tests for the added functionality.

Running tests

Requirements

You must have DynamoDB Local installed. It is a local version of DynamoDB that can be used for local development and test execution.

The test suite assumes that DynamoDB Local will be running at port `8000`.

You can either run DynamoDB Local your self or install it under `dynamodb-local` in the project root. If you do this, you can simply start the database with `make dynamodb_local`

Executing the test suite

You can run the test suite via `make tests` or `python test.py`.

CHAPTER 8

License

APACHE LICENSE 2.0 Copyright 2014 Sebastian Dahlgren

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER 9

DynamoDB Config Store documentation

Overview

Store your configuration in DynamoDB using DynamoDB Config Store.

Using this Python class you'll be able to easily manage application configuration directly in DynamoDB. It works almost like any configuration file, except that an option can have multiple values. For example your configuration could look like this:

```
------------+----------------+-----------------+----------------+-----------------
_store      | _option        | host            | port           | secret-key
------------+----------------+-----------------+----------------+-----------------
prod        | db             | db-cluster.com | 27017          |
prod        | external-port  |                 | 80             |
prod        | secret-key     |                 |                | abcd1234
test        | db             | localhost       | 27017          |
test        | external-port  |                 | 4000           |
prod        | secret-key     |                 |                | test1234
------------+----------------+-----------------+----------------+-----------------
```

You can then retrieve configuration like this:

```
store = DynamoDBConfigStore(
    connection,                 # dynamodb2 connection from boto
    'config',                   # DynamoDB table name
    'prod')                     # Store name

# Get the 'db' option and all it's values
store.config.get('db') # Returns {'host': 'db-cluster.com', 'port', Decimal(27017)}
```

In our lingo a **Store** is roughly equivivalent to a configuration file. And an **Option** is an key in the Store which holds zero or more **Keys**.

Python Module Index

d

Index

Symbols

C

D

M

R

S

T