cockatiel Documentation

Release 0.0.2

Raphael Michel

April 10, 2016

Contents

Cockatiel is a replicating file server for small-scale setups. It is intended to be used e.g. for handling user-uploaded files to a web application in a redundant way.

CHAPTER 1

Features

- Uploading and deleting files
- Asynchronous replication across multiple nodes
- Automatic failure recovery

CHAPTER 2

Documentation content

2.1 Using cockatiel

2.1.1 Requirements

- cockatiel requires Python 3.4 or newer
- cockatiel has only been tested in Linux so far

2.1.2 Installation

Installing cockatiel is really straightforward. We recommend that you set it up inside a virtual environment in order to isolate its dependencies from other python projects that you might use. Inside the Python 3 virtual environment you can then just run:

```
$ pip install cockatiel
```

to obtain the latest release.

Warning: Before using cockatiel for your project, please make sure that you read and understood the *Assumptions* that cockatiel makes about your requirements.

2.1.3 Command-line options

Cockatiel is currently configured via command-line parameters. Logging is performed via stdout.

Simple replicating file storage.

```
usage: python3 -m cockatiel [-h] [--port PORT] [--host HOST] --storage PATH
                            --queue PATH [--url URL] [--node URL] [-v]
```

Options:

--port=8080, -p=8080 The port that this cockatiel server should bind to.

--host=0.0.0.0, -H=0.0.0.0 The IP address of the interface that this cockatiel server should listen on.

--storage The path to the directory to store the actual files in. The cockatiel process needs permission to read and write files and create new subdirectories at this location.

--queue Path to a directory to store the retry queue. The cockatiel process needs permission to read, write and delete file at this location.

--url= The URL this service is publicly reachable at, e.g. http://10.1.1.1:8123/foo or https://mydomain.com/media, depending on your reverse proxy setup.

--node Specify this option once for every other node on your cluster. Every value should be a valid URL prefix like http://10.1.1.2:8012

-v=False, --verbose=False Enable debugging output. Without this flag, only errors and warnings are logged.

2.1.4 Running cockatiel as a service

To automatically run cockatiel at system startup, you can register it as a system service.

TBD systemd example

2.1.5 Adding new nodes to the cluster

If you're system is growing and you'd like to add a new node to the cluster, you'll need to go through the following steps in the given order:

1. Set up and start cockatiel on the new server, including all existing servers in its cluster configuration.
2. Add the URL of the new server to the cluster configuration on all other nodes, then restart those nodes.
3. Manually copy over the complete storage directory from one of the existing nodes to your new node, e.g. using `rsync`.

2.1.6 Using cockatiel for a Django application

We have a Django storage backend for cockatiel available at django-cockatiel.

2.2 Design decisions

Cockatiel doesn't try to be a CDN, but to implement the simplest solution that fulfills our needs. Currently, cockatiel makes a number of assumptions that are outlined below. If those assumptions do not apply to your needs, you should probably be looking for a CDN-like solution or for a proper distributed file system or block device.

2.2.1 Assumptions

All files are on all servers. Cockatiel currently does not implement any kind of sharding and we do not plan to do so, so Cockatiel is designed for file collections that can easily fit on a modern hard drive.

File names will be (partly) auto-generated. In order to avoid collisions, cockatiel will insert a file's SHA1 checksum into the filename. Therefore, the file will not be stored exactly at the location the client specified.

Files get replicated asynchronously. If your network connection is slow or flaky, this can lead to a delay between a file being on one server and a file being distributed across all servers.

Files don't change. It is not possible to change a file through cockatiel. If you want to replace a file, just delete the old one and upload a new one that will get a new name (due to the checksum that will be inserted into the filename).

Adding or removing nodes may require manual intervention. There currently is neither automatic service discovery nor cluster configuration management during runtime.

Files are being served by a different webserver. Cockatiel does not intend to be a high-performance web server. If your files get accessed a lot, please use a proper web server like nginx and point it to the cockatiel's storage directory.

2.2.2 Implementation

- cockatiel is a stand-alone service implemented in Python using `asyncio`.
- The service exposes a very simple *HTTP API* that is used both for the communication between a client and the service as well as for the replication between the cockatiel nodes.
- Every operation gets inserted into a queue. This queue is persisted to a directory on the file system. An operation stays inside the queue as long as it has not been accepted by all neighbor servers.

2.2.3 Failure modes

cockatiel is currently designed to automatically cope with the following events:

Server downtime: If one node of the cluster goes offline, the other servers will queue up all operations and retry them periodically. The retrial interval is currently configured to increase from twice a second two once every 30 seconds if the server is down for a longer period. Therefore, once the server returns, the other servers will start pushing all changes within 30 seconds.

Network corruption: If a file arrives corrupted after a replication, e.g. the calculated SHA1 sums of the sender and the receiver mismatch, the operation will be aborted and retried.

Network partition: If you have three nodes A, B, and C, and the network between A and C gets interrupted, an operation performed on A will still be propagated to server C.

Connection interruption: Any operation stays queued for replication as long as the receiving server did not acknowledge it. Therefore, if an operation is interrupted, it will be retried.

2.3 HTTP API

Cockatiel exposes a HTTP API that you can use to store, retrieve and delete files from its storage. The same API is being used by cockatiel for the communication between different nodes.

2.3.1 API methods

`GET /` (*filename*)
Returns the file with the given filename.

Request Headers

- If-None-Match – A value that you obtained from the `ETag` header of a response that you still have in your cache.

Response Headers

- Content-Type – The content type of a file, determined by its extension
- ETag – A hash value specific to this file. You can specify this in the `If-None-Match` request header for cache validation.

- Cache-Control – Cache control instructions, normally telling you that you can cache this for at least a year.
- *X-Content-SHA1* – The SHA1 hash of the transmitted file

Status Codes

- 200 OK – if the file exists and can be read
- 304 Not Modified – if you provided `If-None-Match`
- 404 Not Found – if the file does not exist
- 500 Internal Server Error – on any internal errors

`PUT /` (*filename*)
Creates a new file with the given filename. You are not guaranteed that the file is actually created with the given name, you should expect to get a new name in the `Location` response header.

Request Headers

- *X-Content-SHA1* – The SHA1 hash of the transmitted file (optional)

Response Headers

- Location – The relative or abosulte URL to the file with the name that acutally has been used when storing the file.

Status Codes

- 201 Created – if the file did not exist on this server before
- 302 Found – if the file already existed on this server previously
- 400 Bad Request – if you specified a SHA1 hash and it does not match the hash calculated on the server
- 408 Request Timeout – if data is coming in to slow
- 500 Internal Server Error – on any internal errors

`DELETE /` (*filename*)
Deletes the file of the given name.

Status Codes

- 200 OK – if the file could be deleted successfully
- 404 Not Found – if the file did not exist
- 500 Internal Server Error – on any internal errors

`HEAD /` (*filename*)
Returns the meta data for the file with the given filename. This behaves exactly the same as `GET`, it just does not return the file's content.

Request Headers

- If-None-Match – A value that you obtained from the `ETag` header of a response that you still have in your cache.

Response Headers

- Content-Type – The content type of a file, determined by its extension
- ETag – A hash value specific to this file. You can specify this in the `If-None-Match` request header for cache validation.

- Cache-Control – Cache control instructions, normally telling you that you can cache this for at least a year.
- *X-Content-SHA1* – The SHA1 hash of the file

Status Codes

- 200 OK – if the file exists and can be read
- 304 Not Modified – if you provided `If-None-Match`
- 404 Not Found – if the file does not exist
- 500 Internal Server Error – on any internal errors

`GET /_status`

Returns status information on this node. This currently includes a dictionary that contains one dictonary for every neighbor node. This inner dictionary contains the current length of the replication queue, i.e. the number of operations known to this node that have not yet been sent to the respective other node.

Example response:

```
{
    "queues": {
        "http://localhost:9001": {
            "length": 4
        }
    }
}
```

Status Codes

- 200 OK – in any known case

2.4 Contribution guide

You are interesting in contributing to Cockatiel? That is awesome! If you run into any problems with the steps below, please do not hesitate to ask!

If you're new to contributing to open source software, don't be afraid of doing so. We'll happily review your code and give you constructive and friendly feedback on your changes.

2.4.1 Development setup

First of all, make sure that you have Python 3.4 installed. We highly recommend that you use a virtual environment for all of the following, to keep this project's dependencies isolated from other Python projects you might use or work on.

To get startet, first of all clone our git repository:

```
$ git clone git@github.com:raphaelm/cockatiel.git
$ cd cockatiel/
```

The second step is to make sure you have a recent version of pip and all our requirements:

```
$ pip install -U pip
$ pip install -Ur requirements.txt
```

There is no third step :)

2.4.2 Running the software

Running the cockatiel server is as easy as executing:

```
$ python3 -m cockatiel
```

within the root directory of the repository.

2.4.3 Running the test suite

Cockatiel's tests are split up into two parts. The unit tests are testing single, isolated components of the codebase, the functional tests are performing end-to-end tests of the API and they run tests on whole simulated cluster setups. Therefore, the unit tests tend to run really fast while running the functional tests might take a longer period of time. You can run them with the following commands:

```
$ py.test unit_tests
$ py.test functional_tests
```

While working on the project, it may come useful to run only part of the test suite. You can either specify a specific test file or even filter by the name of the test:

```
$ py.test unit_tests/test_queue.py
$ py.test functional_tests/test_queue.py -kdelete
```

2.4.4 Building the documentation

To build the documentation as HTML files, you need to issue the following commands:

```
$ cd docs/
$ make html
```

You can then point your browser to `<repo-path>/docs/_build/html/index.html`.

2.4.5 Sending a patch

If you improved cockatiel in any way, we'd be very happy if you contribute it back to the main code base! The easiest way to do so is to create a pull request on our GitHub repository.

Before you do so, please squash all your changes into one single commit. Please use the test suite (see above) to check whether your changes break any existing features. Please also run the following command to check for any code style issues:

```
$ flake8 cockatiel unit_tests functional_tests
```

We automatically run the tests and the code style check on every pull request on Travis CI and we won't accept any pull requets without all tests passing.

If you add a new feature, please include appropriate documentation into your patch. If you fix a bug, please include a regression test, i.e. a test that fails without your changes and passes after applying your changes.

Note: If the tests fail on the Travis CI server but succeed on your local machine most of the time, don't panic. Due to the nature of some of the functional tests, they are not completely deterministic.

/(filename)

```
HEAD /(filename),8
GET /(filename),7
PUT /(filename),8
DELETE /(filename),8
```

/_status

```
GET /_status,9
```