Bytengine Documentation

Release 0.2.0

JKE Wilson

December 18, 2014

Contents

Bytengine is an HTTP based content repository API. It enables you to store JSON and Binary Data in a pseudo hierarchical file system of files and folders.

Typical usage:

- Content Management System backend
- Document Management System backend
- Digital assets storage and delivery server
- Questionnaire application repository

CHAPTER 1

Quick Tutorial - Python

This guide will give you a quick overview of a few Bytengine api calls.

This guide assumes that you be running Bytengine locally on its default ports. The python code uses the excellent requests module.

Create admin user and start Bytengine server

```
cd $YOUR_BYTENGINE_SERVER_EXECUTABLE_DIR
./bytengine createadmin -u=admin -p=password
./bytengine run
```

Connect to Bytengine

```
>>> import requests
>>> url = "http://localhost:8500/"
>>> r = requests.get(url)
>>> print r.text
{"bytengine":"Welcome","version":"0.2.0"}
```

Login and create database

```
>>> url = "http://localhost:8500/bfs/token"
>>> data = {"username":"admin","password":"password"}
>>> r = requests.post(url, data=data)
>>> j = r.json()
>>> print j["status"]
ok
>>> token = j["data"]
>>> cmd = 'server.newdb "test"; server.listdb;'  # issue two commands
>>> url = "http://localhost:8500/bfs/query"
>>> data = {"token":token,"query":cmd}
>>> r = requests.post(url, data=data)
>>> j = r.json()
>>> print j["status"]
ok
>>> print j["data"][-1]  # get last result
[u'test']
```

Content creation: BQL script: ch1_script.bql

```
/* ============================================
   This is a BQL comment:

   Multiple commands can be issued in a single
```

```
   script but must be separated by a ';'
   Results from the all commands will be
   returned in an array.
============================================== */

/*---- create directories ----*/

@test.newdir /myapp ;
@test.newdir /myapp/users ;

/*---- create file with valid JSON data ----*/

@test.newfile /myapp/users/u1 {"name":"justin","age":24} ;
@test.newfile /myapp/users/u2 {"name":"lola","age":57} ;
@test.newfile /myapp/users/u3 {"name":"jenny","age":33} ;
@test.newfile /myapp/users/u4 {"name":"sam","age":16} ;

/*---- search for users ----*/

@test.select "name" "age" in /myapp/users
where "age" < 20 or regex("name","i") == "^j";
```

Load and execute script

```
>>> with open("ch1_script.bql","r") as f:
...     cmd = f.read()
...
>>> url = "http://localhost:8500/bfs/query"
>>> data = {"token":token,"query":cmd}
>>> r = requests.post(url, data=data)
>>> j = r.json()
>>> j["status"]
u'ok'
>>> len(j["data"][-1])
3
```

CHAPTER 2

Quick Tutorial - bshell

We are going to use bshell. this time round to execute the commands issued in the previous python tutorial. bshell (Bytengine Shell) makes it much easier to test and build up scripts that can later be used in an application.

We'll first have to install bshell from source (or download the binaries).

Build from source (requires Go)

This assumes you have setup $GOPATH and have added $GOPATH/bin to $PATH. We'll also assume that you have Bytengine installed.

```
go get github.com/johnwilson/bytengine/cmd/bshell
bshell run -u=admin -p=password
```

Create database

```
bql> server.newdb "test"; server.listdb;
{
    "data": [
        true,
        [
            "test"
        ]
    ],
    "status": "ok"
}
```

Content creation: BQL script

We are going to execute the following script using bshell's editor command

```
/* ============================================
   This is a BQL comment:

   Multiple commands can be issued in a single
   script but must be separated by a ';'
   Results from the all commands will be
   returned in an array.
============================================== */

/*---- create directories ----*/

@test.newdir /myapp ;
@test.newdir /myapp/users ;

/*---- create file with valid JSON data ----*/
```

```
@test.newfile /myapp/users/u1 {"name":"justin","age":24} ;
@test.newfile /myapp/users/u2 {"name":"lola","age":57} ;
@test.newfile /myapp/users/u3 {"name":"jenny","age":33} ;
@test.newfile /myapp/users/u4 {"name":"sam","age":16} ;

/*---- search for users ----*/

@test.select "name" "age" in /myapp/users
where "age" < 20 or regex("name","i") == "^j";
```

Open bshell editor ('vim' by default) and type/save the above script

```
bql> \e
```

You should see the following response

```
{
    "data": [
        true,
        true,
        true,
        true,
        true,
        true,
        [
            {
                "content": {
                    "age": 24,
                    "name": "justin"
                },
                "path": "/myapp/users/u1"
            },
            {
                "content": {
                    "age": 33,
                    "name": "jenny"
                },
                "path": "/myapp/users/u3"
            },
            {
                "content": {
                    "age": 16,
                    "name": "sam"
                },
                "path": "/myapp/users/u4"
            }
        ]
    ],
    "status": "ok"
}
```

As we can see, the result for each of the commands issued is returned by the server.

The bshell example is less verbose than the python one which makes it an essential tool when working with Bytengine.

CHAPTER 3

Why Bytengine

Bytengine is designed to be a lightweight content repository for storing JSON documents as well as digital assets. Your data is stored in files that can be organised using directories, similar to a regular OS file system.

A BFS (Bytengine File System) file is made up of two layers, the JSON data layer and the bytes data layer. This allows you for example to store digital assets along with their metadata that can be queried using Bytengine Query Language (BQL).

Bytengine's role in your application stack is to store form data in JSON format or as raw bytes (if it happens to be a scanned document, word document or pdf document) and to allow you to organise and query that data easily. This means that it is the ideal repository for questionnaire data, cv and application form data, and general business form data. The ability to structure your data into directories also means you can build a sophisticated workflow layer which goes beyond just adding a status field to track a forms progress within a process.

Bytengine is written in Go and is open source so you can extended it to include features particular to your application's needs.

CHAPTER 4

HTTP API

Bytengine HTTP API allows you to interact with the server using your favorite HTTP client library or application.

4.1 Quick reference

URL	HTTP Verb	Functionality
/	GET	*Welcome message*
/bfs/token	POST	*Get an authentication token*
/bfs/query	POST	Send a BQL request
/bfs/uploadticket	POST	Get a file upload ticket
/bfs/writebytes/:ticket	POST	Upload file
/bfs/readbytes	POST	Download file
*/bfs/direct/:layer/:database/*path*	GET	Content static serve

4.2 Welcome message

Returns a welcome message from Bytengine. This url is usefull for checking if you're connected to the server when writing clients in your prefered language.

Example:

```
curl -X GET http://localhost:8500/
```

Return Value:

```
{
    "bytengine": "Welcome",
    "version": "0.2.0"
}
```

4.3 Get an authentication token

Gets an authentication token from the server that must be included in POST requests to interact with content.

Example:

```
curl -X POST \
    -d 'username=admin&password=password' \
    http://localhost:8500/bfs/token
```

Return Value:

```
{
    "data": [token as a string],
    "status": "ok"
}
```

CHAPTER 5

Server Commands

Server commands can only be run by a user who has **root** access.

5.1 server.init

server.init removes all file system content and returns the names of databases deleted from the server.

Return Value:

```
{
    "status": "ok",
    "data": [
        db_1,
        db_2,
        ...,
        db_n
    ]
}
```

Example:

```
server.init
```

5.2 server.newdb

server.newdb creates a new database.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
server.newdb "db1"
```

5.3 server.listdb

server.listdb list all databases on the server.

Options:

```
--regex: regular expression string
```

Return Value:

```
{
    "status": "ok",
    "data": [
        db_1,
        db_2,
        ...,
        db_n
    ]
}
```

Example:

```
server.listdb
server.listdb --regex="^j"
```

5.4 server.dropdb

server.dropdb deletes a database.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
server.dropdb "db1"
```

CHAPTER 6

User Commands

User commands can only be run by a user who has **root** access except **user.whoami**.

6.1 user.new

user.new creates a new (non-root) server user.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
user.new "username" "password"
```

6.2 user.all

user.all lists all server users (usernames).

Options:

```
--regex: regular expression string
```

Return Value:

```
{
    "status": "ok",
    "data": [
        "user_1",
        ...,
        "user_n"
    ]
}
```

Example:

```
user.all
user.all --regex="^ad"
```

6.3 user.about

user.about returns information about a user (databases, status, etc...).

Return Value:

```
{
    "status": "ok",
    "data": {
        "active": true,
        "databases": [],
        "root": false,
        "username": "user1"
    }
}
```

Example:

```
user.about "username"
```

6.4 user.delete

user.delete deletes a user from the server.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
user.delete "username"
```

6.5 user.passw

user.passw changes user's password.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
user.passw "username" "newpassword"
```

6.6 user.access

user.access grants/denies user access to the server.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
user.access "username" grant
user.access "username" deny
```

6.7 user.db

user.db grants/denies user access to a database on the server.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
user.db "username" "database" grant
user.db "username" "database" deny
```

6.8 user.whoami

user.whoami show current user's information.

Return Value:

```
{
    "status": "ok",
    "data": {
        "databases": [],
        "root": false,
        "username": "user1"
    }
}
```

Example:

```
user.whoami
```

CHAPTER 7

File System Commands

File system commands enable the creation and querying of content. All commands commands must be preceeded by the name of the database with a '@' prefix. File and Directory names cannot include spaces.

7.1 database.newdir

database.newdir creates a directory at the given path. Paths are unix/linux style paths (i.e. with forward slash separator) and the last element in the path will be the name of the directory. The **root directory ('/')** is created by default and cannot be modified.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.newdir /var
@db1.newdir /var/log
```

7.2 database.newfile

database.newfile creates a file and writes the data to the JSON layer. Similarily to directories the file will be created at the given path. Paths are unix/linux style paths (i.e. with forward slash separator) and the last element in the path will be the name of the file. The data must be a valid JSON object.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.newfile /var/log/file1.log {"events": []}
@db1.newfile /var/log/file2.log {}
```

7.3 database.readfile

database.readfile returns the JSON layer of a file. An array of fields can be added to limit the returned data.

Return Value:

```
{
    "status": "ok",
    "data": {...}
}
```

Example:

```
@db1.readfile /var/logs/file1.log
@db1.readfile /var/logs/file1.log ["field1", "field1.field1_1"]
```

7.4 database.updatefile

database.updatefile overwrites the JSON layer of a file.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.updatefile /var/logs/file1.log {"field1":{"field1_1": "value"}}
```

7.5 database.deletebytes

database.deletebytes deletes the Bytes layer content of a file.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.deletebytes /var/logs/file1.log
```

7.6 database.listdir

database.listdir lists the contents of a directory. Content is seperated into directories (dirs), files (files) and files with non-empty 'bytes layer' (bfiles). The 'regex' option filters the return values.

Options:

```
--regex: regular expression string
```

Return Value:

```
{
    "status": "ok",
    "data": [
        "dirs": [],
        "files": [],
        "bfiles": []
    ]
}
```

Example:

```
@db1.listdir / --regex="^v"
@db1.listdir /var/log/
```

7.7 database.rename

database.rename renames a file or directory. The root directory '/' cannot be renamed.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.rename /var/log "logs"
@db1.rename /var/logs/file1.log "filelog.1"
```

7.8 database.move

database.move moves a file or directory to a new directory.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.move /var/logs/filelog.1 /var
```

7.9 database.copy

database.copy makes a copy of a file or directory. The last element in the new path will be the name of the copied file/directory.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.copy /var/logs /var/file_logs
```

7.10 database.delete

database.delete deletes a file or directory. In the case of a directory it performs a recursive delete.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.delete /var/file_logs
```

7.11 database.info

database.info returns metadata for files/directories such as creation date, parent directory, type etc...

Return Value:

```
{
    "status": "ok",
    "data": {
        "created": "2014:10:23-17:42:46.5623",
        "type": "file",
        ...
    }
}
```

Example:

```
@db1.info /
@db1.info /var/logs/filelog.1
```

7.12 database.makepublic

database.makepublic makes a file publicly available which means it can be accessed directly without authentication. View *Direct Content Access* for further details. All files are private by default.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.makepublic /var/logs/file1.log
```

7.13 database.makeprivate

database.makeprivate makes a file private.

Return Value:

```
{
    "status": "ok",
    "data": true
}
```

Example:

```
@db1.makeprivate /var/logs/file1.log
```

7.14 database.counter

database.counter creates a **'counter'** or global integer value for the database which can be incremented **'incr'**, decremented **'decr'** or reset **'reset'**. This command can be used to create primary keys for content. 'database.counter list' returns all counters and their values in the database.

Return Value:

```
// database.counter [name] incr [value]
{
    "status": "ok",
    "data": [incremented integer value]
}

// database.counter [name] decr [value]
{
    "status": "ok",
    "data": [decremented integer value]
}

// database.counter [name] reset [value]
{
    "status": "ok",
    "data": [incremented integer value]
}

// database.counter list
{
    "status": "ok",
    "data": [
        {"name":"...", "value": [current integer value]}
```

```
    ]
}
```

Example:

```
@db1.counter "logs" incr 1
@db1.counter "logs" incr 5
@db1.counter "logs" decr 2
@db1.counter "logs" reset 0

@db1.counter list
@db1.counter list --regex="^l"
```

CHAPTER 8

Advanced File System Commands

8.1 database.select

database.select retrieves fields from files in directories based on field values or file metadata values specified in the **Where statement**. Additional statements such as **Limit, Sort, Count or Distinct** can be used to further filter the query result. Multiple field comparison can be seperated by **Or** (**And** is implied).

Field value comparison operators are:

- equal: `==`
- not equal: `!=`
- greater than: `>`
- greater than or equal: `>=`
- less than: `<`
- less than or equal: `<=`

Field inclusion/exclusion operators are:

- inclusive of: `in`
- exclusive of: `nin`

Field functions available are:

- type of field: `typeof([FIELD])` (supported types are 'string', 'int')
- existance of field: `exists([FIELD])`
- field regular expression: `regex([FIELD], REGEX_OPTION)`. Valid REGEX_OPTION values can be found in the mongodb documentation.

Special Field values are:

- file name: `file_name`
- file privacy: `file_ispublic`

Return Value:

```
{
    "status": "ok",
    "data": [
        {
            "path": "/.../...",
```

```
            "content": {...}
        },
        ...
    ]
}
```

Example:

```
@db1.select "age" in /users /staff limit 20

@db1.select "name" "course.room" in /students where "year">=1 "status"=="active"

@db1.select "date" in /logs where "event"=="failure" limit 50

@db1.select "" in /users where regex(file_name, "i") == "^j" distinct "age"

@db1.select "" in /members count
@db1.select "" in /members distinct "country"

@db1.select "name" in /banned_users sort asc "date_joined"
@db1.select "name" in /banned_users sort desc "date_joined"

@db1.select "status" in /users where typeof("children")=="int"
```

8.2 database.set

database.set sets file field values in a directory based on **Where statement** as specified in the **database.select** command.

Return Value:

```
{
    "status": "ok",
    "data": [number of affected files]
}
```

Example:

```
@db1.set "country"="Ghana" in /users where "country"=="gh"
```

8.3 database.unset

database.unset unsets (or removes) file field values in a directory based on **Where statement** as specified in the **database.select** command.

Return Value:

```
{
    "status": "ok",
    "data": [number of affected files]
}
```

Example:

```
@db1.unset "country" in /users where "country"=="gh"
```

CHAPTER 9

Data Filter Functions

Bytengine allows you to filter/modify commands results by redirecting their output to **Filter Functions** (or user defined functions). This feature comes in handy especially when you want to process the data server-side before returning it to your application. The syntax to redirect output is `>> function_name` and can be used after any bfs command.

Currently only a sample function `pretty` which pretty prints bfs commands is included as an example but further useful ones will be added eventually. You can view the filter function plugin code implementation in /bytengine/fltcore/fltcore.go.

Example:

```
@db1.select "name" in /users >> pretty

@db1.select "name" in /users >> my_custom_function
```

CHAPTER 10

Direct Content Access

Bytengine allows you to server your content directly (as static content) via Http GET. You have to first make the bytengine file **public** by issuing a *database.makepublic* command which will make the content available at the following url:

```
http://[your bytengine server address]/bfs/direct/[layer]/[database]/[path]
```

Where:

- **[layer]** is the bytengine file layer you want to retrieve (JSON or Bytes)
- **[database]** is the database name
- **[path]** is the bytengine file path

The response header will either be `application/json` if the layer is `json` and `application/octet-stream` if the layer is `bytes`

Example:

```
http://localhost:8500/bfs/direct/json/db1/users/active/file1
```

```
http://localhost:8500/bfs/direct/bytes/db1/users/active/file1_profile_picture
```

CHAPTER 11

Configuring Bytengine

Bytengine configuration file is in JSON format and can be found at /bytengine/bytengine/config.json.

The authentication (authentication) and bytengine file system (filesystem) plugins use Mongodb. as their database and the mgo driver.

```
{
    "plugin": "mongodb",                // authentication plugin name
    "addresses":["localhost:27017"],    // mongodb server(s)
    "authdb":"",                        // mongodb authentication database
    "username":"",                      // mongodb authentication username
    "password":"",                      // mongodb authentication password
    "timeout":60                        // mongodb client timeout
}
```

The state store plugin (statestore), for tokens and cache, relies on Redis.

```
{
    "plugin": "redis",              // state store plugin name
    "address": "localhost:6379",    // redis server address
    "password": "",                 // redis server password
    "timeout": 60,                  // redis client timeout
    "database": 1                   // database index
}
```

The byte store (bytestore) plugin uses Diskv. for storage.

```
{
    "rootdir":"/tmp/bytengine_bst", // directory to store content
    "cachesize":1                   // cache size in mb
}
```

Other configurations

```
{
    ...
    "workers": 2,           // number of goroutines
    "port": 8500,           // bytengine server port
    "address": "localhost", // bytengine server address
    "timeout": {
        "authtoken": 60,    // authentication cache timeout in minutes
        "uploadticket": 60  // upload ticket cache timeout in minutes
    }
}
```

CHAPTER 12

Customising Bytengine