LECTURE13: ELASTICSEARCH-PHP

CS418/518: WEB PROGRAMMING

BY DR. JIAN WU

OUTLINE

- Installation of ElasticSearch (ES) on Windows and Mac OS
- Indexing the first document
- Configuration
- Search via Kibana
- Using ES within PHP: elasticsearch-php

INSTALL ELASTICSEARCH-PHP

- Elasticsearch-php has three requirements that you need to worry about:
  - PHP 7.0.0 or higher
  - Composer: a package and dependency manager for PHP.
  - ext-curl: (optional) the Libcurl extension for PHP (to make query faster)
  - Native JSON Extensions (ext-json) 1.3.7 or higher

reference: https://github.com/elastic/elasticsearch-php

INSTALL ELASTICSEARCH-PHP

- Installing php composer (covered previously)
- Include elasticsearch-php in your composer.json file:

```
{
    "require": {
        "elasticsearch/elasticsearch": "~7.0"
    }
}
```

- Install the client with composer

```
curl -s http://getcomposer.org/installer | php
php composer.phar install --no-dev
```

INCLUDE GENERATED AUTOLOADER IN PHP PROJECT

```php
<?php
require 'vendor/autoload.php';

$client = Elasticsearch\ClientBuilder::create()->build();

$params = [
  'index' => 'my_index',
  'id'    => 'my_id',
  'body'  => ['testField' => 'abc']
];

$response = $client->index($params);
echo "<h3>We indexed these.</h3>";
print_r($params);
echo "<h3><Response/h3>";
print_r($response);
echo "<br>";
?>
```

indexing.php

INCLUDE GENERATED AUTOLOADER IN PHP PROJECT

- indexing.php

```php
require 'vendor/autoload.php';

$client = Elasticsearch\ClientBuilder::create()->build();
```

CONFIGURATION

- If no hosts are specified, the client will attempt to connect to `localhost:9200`.

possible forms of host names

```
$hosts = [
    '192.168.1.1:9200',         // IP + Port
    '192.168.1.2',              // Just IP
    'mydomain.server.com:9201', // Domain + Port
    'mydomain2.server.com',     // Just Domain
    'https://localhost',        // SSL to localhost
    'https://192.168.1.3:9200'  // SSL to IP + Port
];
$client = ClientBuilder::create()           // Instantiate a new ClientBu
                    ->setHosts($hosts)      // Set the hosts
                    ->build();              // Build the client object
```

You can specify a host name

the ClientBuilder object allows chaining method calls for brevity. It is also possible to call the methods individually:

Call ClientBuilder methods individually.

```
$clientBuilder = ClientBuilder::create();    // Instantiate a new ClientB
$clientBuilder->setHosts($hosts);            // Set the hosts
$client = $clientBuilder->build();           // Build the client object
```

INDEX A DOCUMENT FROM PHP

these are fields to be specified.

```
$params = [
      'index' => 'my_index’,
      ‘id' => 'my_id’,
      'body' => ['testField' => 'abc’]
];
$response = $client->index($params);
print_r($response);
```

response on a web browser:

```
Array
(
      [_index] => my_index
      [_type] => _doc
      [_id] => my_id
      [_version] => 1
      [created] => 1
)
```

See indexing.php

SEE IF YOU ARE SUCCESSFUL

```
GET /_cat/indices/?v
```

```
health status index                            uuid                   pri rep
  docs.count docs.deleted store.size pri.store.size
green  open   .kibana-event-log-7.9.2-000001 VuDf7YEmRSOaHR6o7ICe-w   1   0
           1            0      5.5kb          5.5kb
yellow open   bank                           xODe93itSgOfUbRYwWUvBg   1   1
        1000            0    382.2kb        382.2kb
green  open   .apm-custom-link               9n1Ov5UPQmCusmgTlEaxcg   1   0
           0            0       208b           208b
green  open   kibana_sample_data_ecommerce   5do9q-nNQp6QISuC9eq62w   1   0
        4675            0      4.6mb          4.6mb
green  open   .kibana_task_manager_1         fHLbQS7eT8yyTaCShzUSKg   1   0
           6         6261      674kb          674kb
yellow open   my_index                       IVvktz0QQleqln_095yLhA   1   1
           1            0      3.7kb          3.7kb
                                             s3kmKCVcTj2xwE9g5W5g8w   1   0
                                               208b
                                             MFRlo219TnK5jQWO4dH57Q   1   0
                                              3.3kb
                                             NQp7FjXyTHOBAQPkdk86sA   1   0
                                             11.3mb
                                             A8GzYQwlRgSyZalvp6iAhA   1   1
                                              3.8kb
```

- **RED:** Damnit. Some or all of (primary) shards are not ready.
- **YELLOW:** Elasticsearch has allocated all of the primary shards, but some/all of the replicas have not been allocated.
- **GREEN:** Great. Your cluster is fully operational. Elasticsearch is able to allocate all shards and replicas to machines within the cluster.

GET A DOCUMENT

simply return the document we just indexed

getting.php

```php
$params = [
    'index' => 'my_index',
    'id'    => 'my_id'
];

$response = $client->get($params);
print_r($response);
```

```
Array
(
    [_index] => my_index
    [_type] => _doc
    [_id] => my_id
    [_version] => 1
    [found] => 1
    [_source] => Array
        (
            [testField] => abc
        )

)
```

metadata

original document

Get operations are performed by requesting a document by its full **index/type/id** path. The above query gets doc at /my_index/_doc/my_id

UPDATE DOCUMENT

- Two cases
  - Completely replace the contents of the existing document
  - Partial update to just some fields (changing values or adding new fields)

existing indexed content

```
$params = [
      'index' => 'my_index’,
      'id' => 'my_id’,
      'body' => ['testField' => 'abc’]
];
$response = $client->index($params);
print_r($response);
```

adding a new field

```
$params = [
    'index' => 'my_index',
    'id'    => 'my_id',
    'body'  => [
        'doc' => [
            'new_field' => 'abc'
        ]
    ]
];

// Update doc at /my_index/_doc/my_id
$response = $client->update($params);
```

- Check if you are successful

SCRIPTED DOCUMENT UPDATE

```
$params = [
    'index' => 'test',
    'id'    => '1',
    'body'  => [
        'script' => [
            'source' => 'ctx._source.counter += params.count',
            'params' => [ 'count' => 4 ]
        ]
    ]
];

echo "<h3>Update index</h3>";
$response=$client->update($params);
print_r($response);
```

- What does update do?

  1.Gets the document from the index.

  2.Runs the specified script.

  3.Indexes the result.

ctx is a special variable that allows you to access the source of the object that you want to update.

first run indextest.php
and then run scripting.php

UPSERT

```
$params = [
    'index' => 'test',
    'id'    => '3',
    'body'  => [
        'script' => [
            'source' => 'ctx._source.counter += params.count',
            'params' => [ 'count' => 4 ]
        ],
        'upsert' => [
            'counter' => 1,
            'tag' => "black"
        ],
    ]
];
```

- An upsert attempts to run your update script, but if the document does not exist, default values will be inserted instead.
- see upserting.php

More examples on document update: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update.html

SEARCH USING “MATCH”

```php
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'match' => [
                'testField' => 'abc'
            ]
        ]
    ]
];

$response = $client->search($params);
print_r($response);
```

```
Array
(
    [took] => 1
    [timed_out] =>
    [_shards] => Array
        (
            [total] => 5
            [successful] => 5
            [failed] => 0
        )

    [hits] => Array
        (
            [total] => 1
            [max_score] => 0.30685282
            [hits] => Array
                (
                    [0] => Array
                        (
                            [_index] => my_index
                            [_type] => _doc
                            [_id] => my_id
                            [_score] => 0.30685282
                            [_source] => Array
                                (
                                    [testField] => abc
                                )
                        )
                )
        )
)
```

Inside of **hits** is another array named **hits**, which contains individual search results

QUERIES IN JSON VS PHP

A standard curl for a Match query using a JSON document

```
curl -X GET "localhost:9200/my_index/_search" -H
'Content-Type: application/json' -d '{
    "query" : {
        "match" : {
            "testField" : "abc"
        }
    }
}'
```

Note you must use –H to specify the content type. Otherwise, you will get a Content-Type error. It looks like by default, the content type is application/x-www-form-urlencoded, not application/json.

response


```
{"took":0,"timed_out":false,"_shards":{"total":1,"successful":1,
"skipped":0,"failed":0},"hits":{"total":{"value":0,"relation":"e
q"},"max_score":null,"hits":[]}}
```


The structure and layout of the PHP array is identical to that of the JSON request body.

The same query constructed in the PHP client.

```
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'match' => [
                'testField' => 'abc'
            ]
        ]
    ]
];

$results = $client->search($params);
```

CHECK JSON QUERY USING JSON_ENCODE()

- A quick method to check your PHP array (for more complex examples) is to encode it back to JSON and check by eye

```
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'match' => [
                'testField' => 'abc'
            ]
        ]
    ]
];

print_r(json_encode($params['body']));


{"query":{"match":{"testField":"abc"}}}
```

USE RAW JSON FILES FOR TESTING PURPOSES

- Use raw JSON as a string in the body, and the client will detect this automatically

```
$json = '{
    "query" : {
        "match" : {
            "testField" : "abc"
        }
    }
}';

$params = [
    'index' => 'my_index',
    'body'  => $json
];

$results = $client->search($params);
```

RETRIEVE FIELDS FROM SEARCH RESULTS

JSON response is serialized back into PHP arrays. Working with the search results is as simple as iterating over the array values

```php
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'match' => [
                'testField' => 'abc'
            ]
        ]
    ]
];
```

```php
$results = $client->search($params);

$milliseconds = $results['took'];
$maxScore     = $results['hits']['max_score'];

$score = $results['hits']['hits'][0]['_score'];
$doc   = $results['hits']['hits'][0]['_source'];
```

```
Array
(
    [took] => 1
    [timed_out] =>
    [_shards] => Array
        (
            [total] => 5
            [successful] => 5
            [failed] => 0
        )

    [hits] => Array
        (
            [total] => 1
            [max_score] => 0.30685282
            [hits] => Array
                (
                    [0] => Array
                        (
                            [_index] => my_index
                            [_type] => _doc
                            [_id] => my_id
                            [_score] => 0.30685282
                            [_source] => Array
                                (
                                    [testField] => abc
                                )
                        )
                )
        )
)
```

BOOLEAN QUERIES

A standard curl for a Match query using a JSON document
Must specify –H for JSON (ignored below)

```
curl -XGET 'localhost:9200/my_index/_search' -d '{
    "query" : {
        "bool" : {
            "must": [
                {
                    "match" : { "testField" : "abc" }
                },
                {
                    "match" : { "testField2" : "xyz" }
                }
            ]
        }
    }
}'
```

The same query constructed in the PHP client.

```
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'bool' => [
                'must' => [
                    [ 'match' => [ 'testField' => 'abc' ] ],
                    [ 'match' => [ 'testField2' => 'xyz' ] ],
                ]
            ]
        ]
    ]
];

$results = $client->search($params);
```

For more details about arrays vs objects in PHP, such as adding a highlight, see Dealing with JSON Arrays and Objects in PHP

A MORE COMPLICATED EXAMPLE

The curl version of the query:

```
curl -XGET 'localhost:9200/my_index/_search' -d '{
    "query" : {
        "bool" : {
            "filter" : {
                "term" : { "my_field" : "abc" }
            },
            "should" : {
                "match" : { "my_other_field" : "xyz" }
            }
        }
    }
}'
```

must means: Clauses that *must* match the document to be included.
should means: If these clauses match, they increase the _score; otherwise, they have no effect. They are simply used to refine the relevance score for each document.

The same query using PHP

```
$params = [
    'index' => 'my_index',
    'body'  => [
        'query' => [
            'bool' => [
                'filter' => [
                    'term' => [ 'my_field' => 'abc' ]
                ],
                'should' => [
                    'match' => [ 'my_other_field' => 'xyz' ]
                ]
            ]
        ]
    ]
];

$results = $client->search($params);
```

curl braces {} mapped to square brackets []; colons : mapped to =>

DELETE A DOCUMENT

this is identical syntax to the get syntax

```
$params = [
    'index' => 'my_index',
    'id'    => 'my_id'
];

$response = $client->delete($params);
print_r($response);
```

The response will confirm the document was deleted:

```
Array
(
    [found] => 1
    [_index] => my_index
    [_type] => _doc
    [_id] => my_id
    [_version] => 2
)
```

DELETE AN INDEX

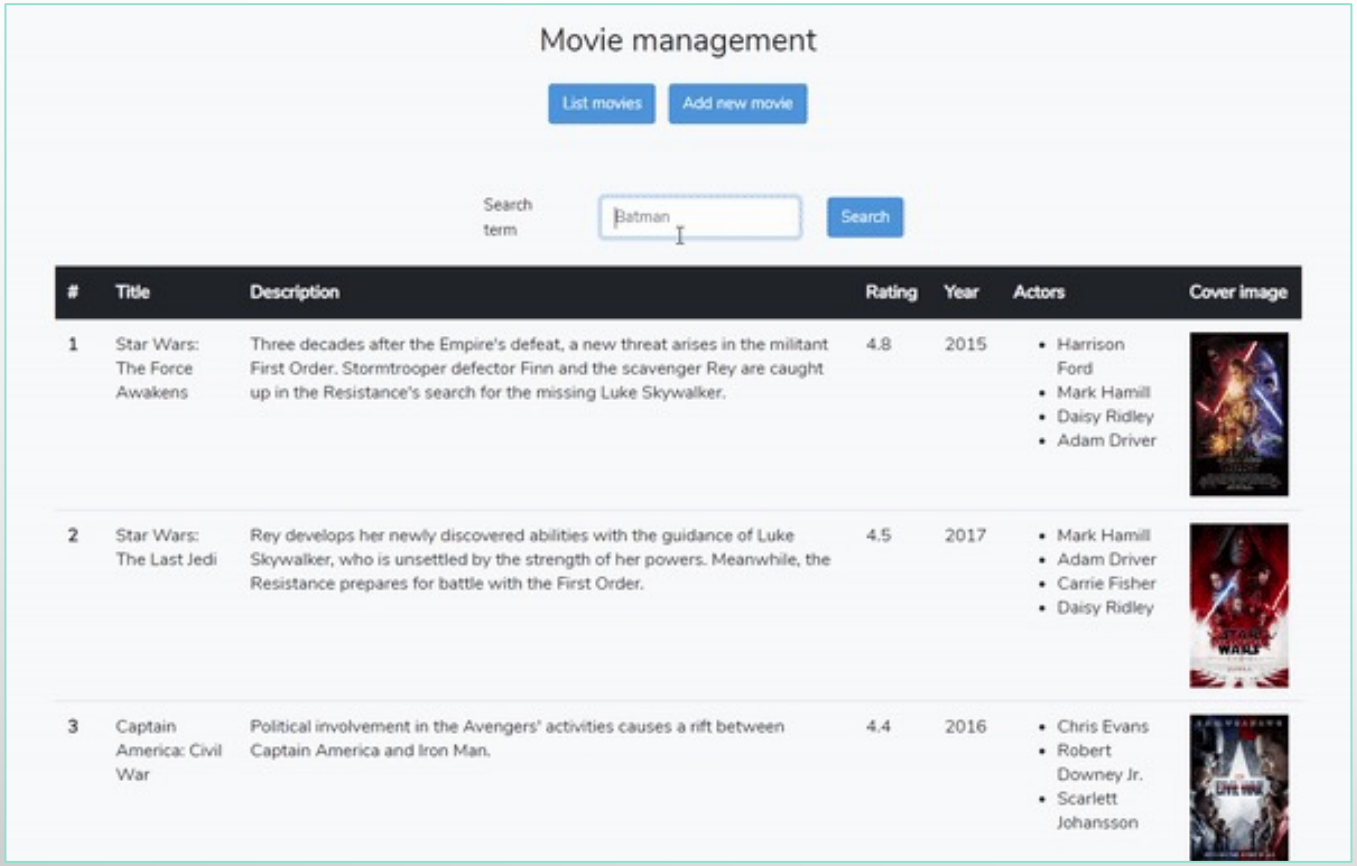
Be very careful!

```
$deleteParams = [
    'index' => 'my_index'
];
$response = $client->indices()->delete($deleteParams);
print_r($response);
```

The response:

```
Array
(
    [acknowledged] => 1
)
```

A MOVIE SEARCH EXAMPLE ON GITHUB



https://medium.com/@factoryhr/elasticsearch-introduction-implementation-and-example-17dd66c35c35

How To Update And Delete an Elasticsearch Document Using The PHP Client

https://kb.objectrocket.com/elasticsearch/how-to-update-and-delete-an-elasticsearch-document-using-the-php-client-172

How to Create a Simple Search Engine Using Elasticsearch PHP Client

https://kb.objectrocket.com/elasticsearch/how-to-create-simple-search-engine-using-elasticsearch-php-client-176

USING THE SCREEN COMMAND TO RUN JOBS ON THE BACKEND

- First install screen
- Under a terminal, under any directory, run
  - `screen -S` [screen session name]
  - This will open a new screen session with a name given by you (Note: capital S)
  - You can run any job under this session
- When the process is launched, hold Ctrl, then press "a", and then "d" to detach the session let the jobs run and back to your terminal.
  - You can run the same command above with a different session name to run other jobs
- When you want to check the jobs, under any directory, run
  - `screen -r` [screen session name]
  - to go back to the session. If you forget the session name, run
  - `screen -ls`
  - to list all sessions saved

TIP: PASSWORDLESS LOGIN TO REMOTE SERVERS

- https://linuxize.com/post/how-to-setup-passwordless-ssh-login/
- Generate a public rsa key
- Copy the key to the remote server
- Set the appropriate permissions