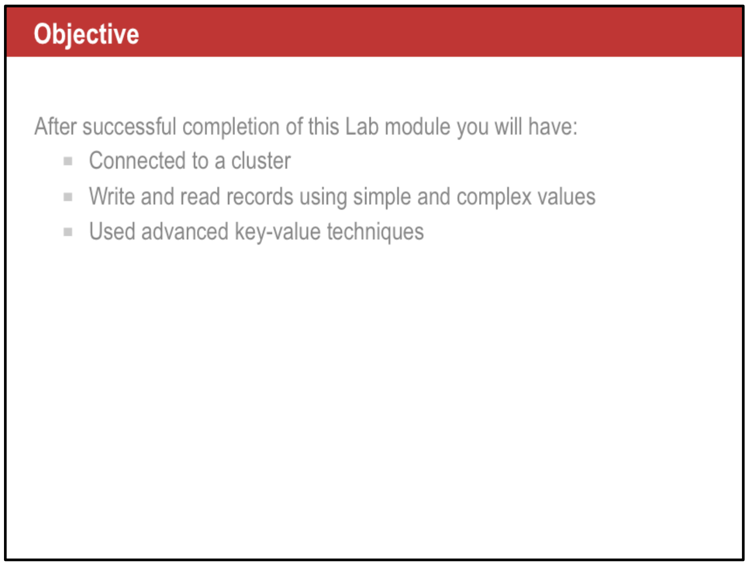
# AS101 Lab Exercises

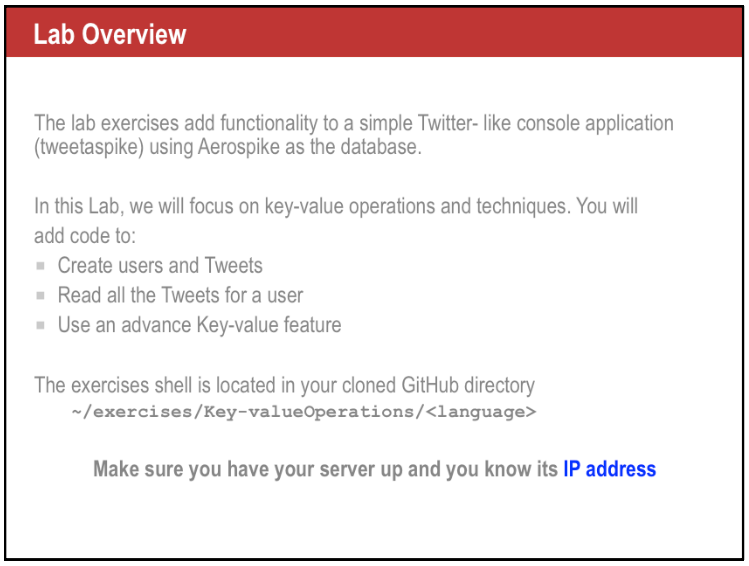
## Lab: Key-value Operations



## Objective

After successful completion of this Lab module you will have:

- Connected to a cluster
- Write and read records using simple and complex values
- Used advanced key-value techniques



## Lab Overview

The lab exercises add functionality to a simple Twitter- like console application (tweetaspike) using Aerospike as the database.

In this Lab, we will focus on key-value operations and techniques. You will add code to:

- Create users and Tweets
- Read all the Tweets for a user
- Use an advance Key-value feature

The exercises shell is located in your cloned GitHub directory

```
~/exercises/Key-valueOperations/<language>
```

**Make sure you have your server up and you know its IP address**

In your cloned or downloaded repository, you will find the following directories:

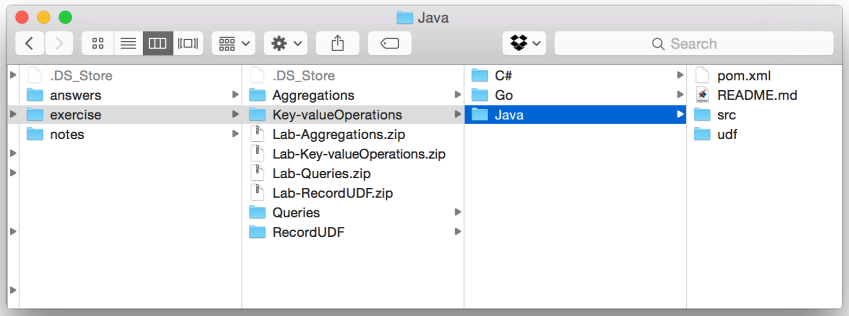
- Answers
- Exercise
- Notes

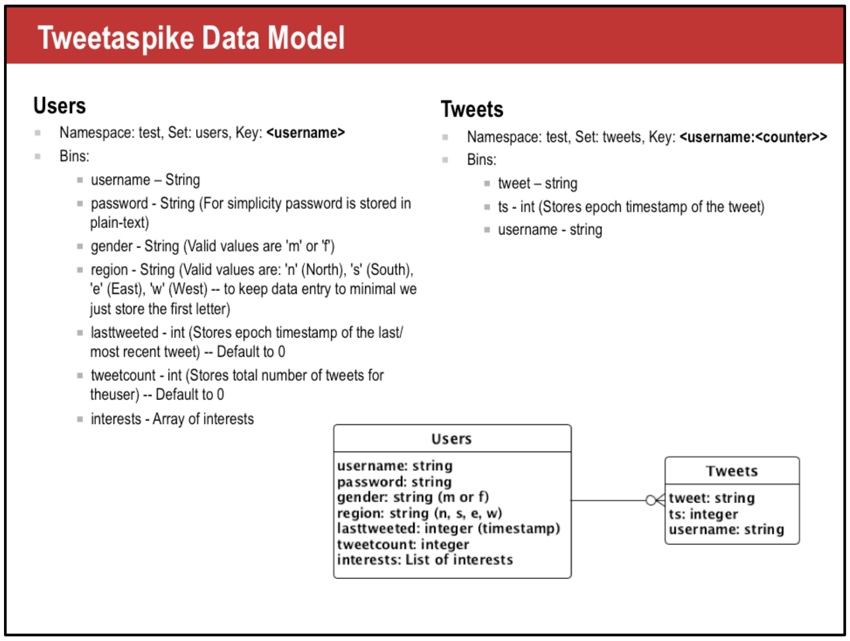
In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Go
- PHP
- Ruby
- Node.js
- Python

The exercises for this module are in the Key-valueOperations directory and your will find a Project/ Solution/Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.

**Make sure you have your server up and you know its IP address**





## Tweetaspike Data Model

**Users**

- Namespace: test, Set: users, Key: **<username>**
- Bins:
  - username – String
  - password - String (For simplicity password is stored in plain-text)
  - gender - String (Valid values are 'm' or 'f')
  - region - String (Valid values are: 'n' (North), 's' (South), 'e' (East), 'w' (West) -- to keep data entry to minimal we just store the first letter)
  - lasttweeted - int (Stores epoch timestamp of the last/ most recent tweet) -- Default to 0
  - tweetcount - int (Stores total number of tweets for theuser) -- Default to 0
  - interests - Array of interests

**Tweets**

- Namespace: test, Set: tweets, Key: **<username:<counter>>**
- Bins:
  - tweet – string
  - ts - int (Stores epoch timestamp of the tweet)
  - username - string

**Users**

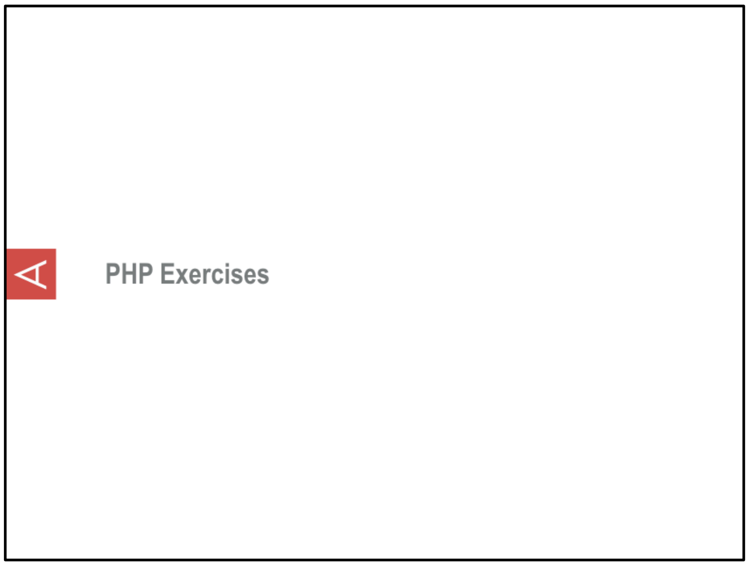
Namespace: test, Set: users, Key: <username>

| Bin name | Type | Comment |
|---|---|---|
| username | String | |
| password | String | For simplicity password is stored in plain text |
| region | String | Valid values are: 'n' (North), 's' (South), 'e' (East), 'w' (West) to keep data entry to minimal we just store the first le] er |
| las] weeted | Integer | Stores epoch Nmestamp of the last/most recent tweet Default to 0 |
| tweetcount | Integer | Stores total number of tweets for the user – Default 0 |
| Interests | List | A list of interests |

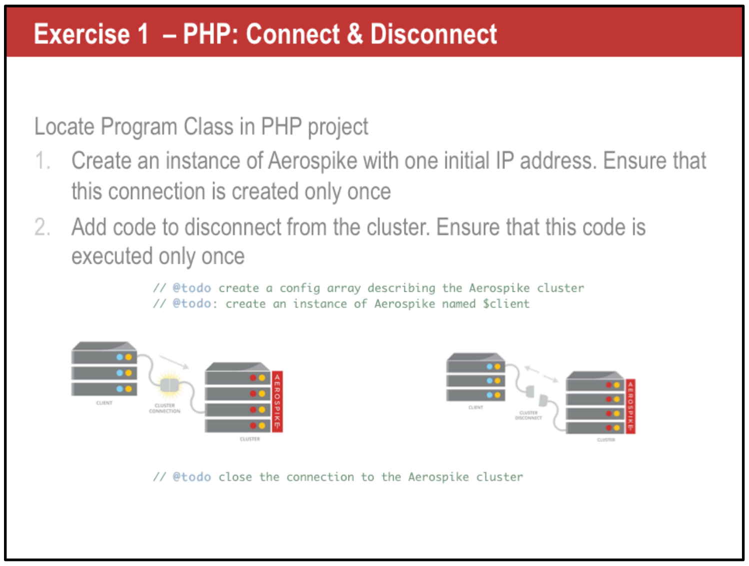
**Tweets**

Namespace: test, Set: tweets, Key: <username:<counter>>

| Bin name | Type | Comment |
|---|---|---|
| tweet | String | Tweet text |
| ts | Integer | Stores epoch Nmestamp of the tweet |
| username | String | User name of the tweeter |



## PHP Exercises



## Exercise 1 – PHP: Connect & Disconnect

Locate Program Class in PHP project

1. Create an instance of Aerospike with one initial IP address. Ensure that this connection is created only once
2. Add code to disconnect from the cluster. Ensure that this code is executed only once

```
// @todo create a config array describing the Aerospike cluster
// @todo: create an instance of Aerospike named $client
```

```
// @todo close the connection to the Aerospike cluster
```

In this exercise you will connect to a Cluster by creating an Aerospike instance, passing a $config structure  to the constructor, that has a single IP address and port.  These should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The Aerospike is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connection or thread pool.

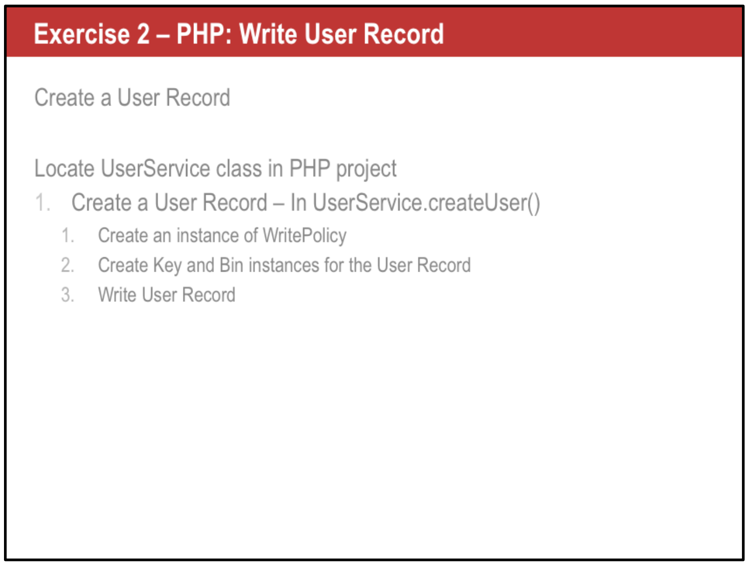
1. In the constructor  for the class Program, add code similar to this;

```
$config = array("hosts" => array(array("addr" => $HOST_ADDR,
                      "port" => $HOST_PORT)));
$client = new Aerospike($config, false);
if (!$client->isConnected()) {
    echo standard_fail($client);
    echo colorize("Connection to Aerospike cluster failed!
Please check the server settings and try again!\n", 'red',
true);
    exit(2);
}
```

**Make sure you have your server up and you know its IP address**

2. At the end of the program, add code, similar to this, to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

```
$client->close();
```


## Exercise 2 – PHP: Write User Record

Create a User Record

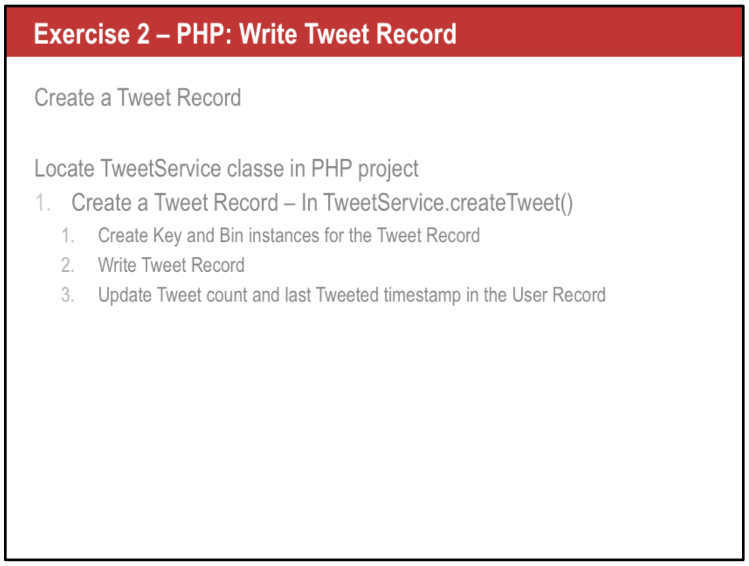
Locate UserService class in PHP project

1. Create a User Record – In UserService.createUser()
   1. Create an instance of WritePolicy
   2. Create Key and Bin instances for the User Record
   3. Write User Record

Create a User Record. In UserService.createUser(), add code similar to this:

1. Create an Array of Bin values from the user input
2. Create a Key
3. Write a user record using the Key and Bins
4. Check result code for errors

```
// Get username
echo colorize("Enter username (or hit Return to skip): ");
$username = trim(readline());
if ($username == '') return;
$bins = array('username' => $username);
// Get password
echo colorize("Enter password for $username: ");
$bins['password'] = trim(readline());
// Get gender
echo colorize("Select gender (f or m) for $username: ");
$bins['gender'] = substr(trim(readline()), 0, 1);
// Get region
echo colorize("Select region (north, south, east or west) for
$username: ");
$bins['region'] = substr(trim(readline()), 0, 1);
// Get interests
echo colorize("Enter comma-separated interests for  $username:
");
$bins['interests'] = explode(',', trim(readline()));
echo colorize("Creating user record >", 'black', true);
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 6);
$status = $this->client->put($key, $bins);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to create user
$username");
}
```



## Exercise 2 – PHP: Write Tweet Record

Create a Tweet Record

Locate TweetService classe in PHP project

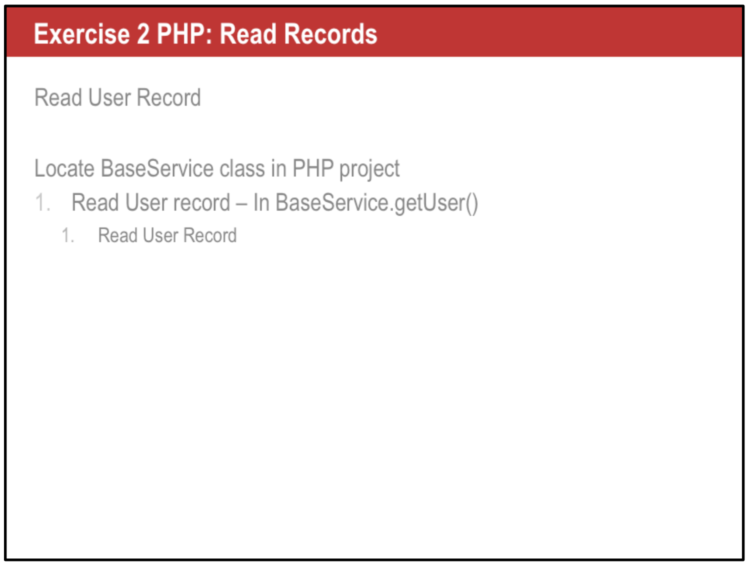
1. Create a Tweet Record – In TweetService.createTweet()
   1. Create Key and Bin instances for the Tweet Record
   2. Write Tweet Record
   3. Update Tweet count and last Tweeted timestamp in the User Record

Create a Tweet Record. In TweetService.createTweet(), add code similar to this:

1. Read user record
2. Create  ey and Bin instances
3. Write a tweet record using the Key and Bins
4. Update the user record with tweet count

```php
// Check if username exists
$record = $this->getUser($username);
$ubins = $record['bins'];
$tweet_count = isset($ubins['tweetcount']) ? $ubins['tweetcount'] : 0;
$tweet_count++;

$bins = array();
// Get a tweet
echo colorize("Enter tweet for $username: ");
$bins['tweet'] = trim(readline());
$bins['ts'] = time() * 1000;
$bins['username'] = $username;
echo colorize("Creating tweet record >", 'black', true);
$key = $this->getTweetKey($username, $tweet_count);
$status = $this->client->put($key, $bins);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to create the tweet");
}
echo success();
return $this->updateUser($username, $bins['ts'], $tweet_count);
```



## Exercise 2 PHP: Read Records

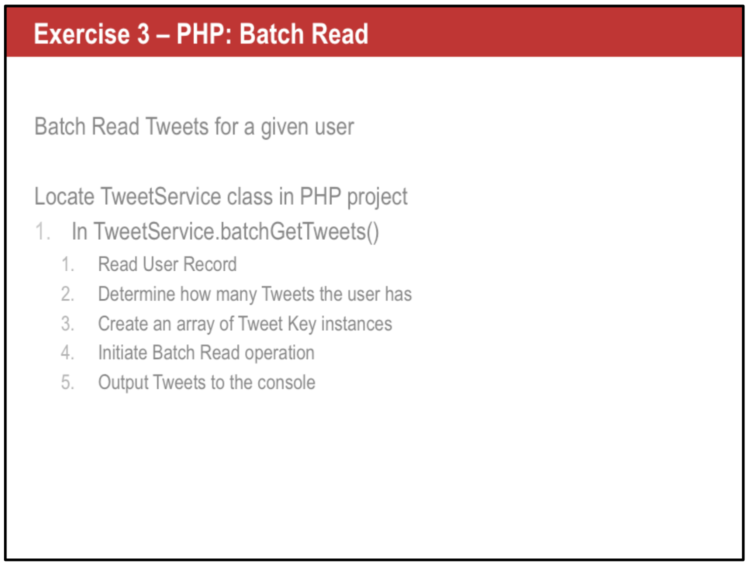
Read User Record

Locate BaseService class in PHP project

1. Read User record – In BaseService.getUser()
   1. Read User Record

Read a User Record. In BaseService.getUser(), add code, similar to this, to:

```
if ($username == '') {
    // Get username
    echo colorize("Enter username (or hit Return to skip): ");
    $username = trim(readline());
}
if ($username != '') {
    $key = $this->getUserKey($username);
    if ($this->annotate) display_code(__FILE__, __LINE__, 6);
    $status = $this->client->get($key, $record);
    if ($status !== Aerospike::OK) {
// throwing an \Aerospike\Training\Exception
throw new Exception($this->client, "Failed to get the user
$username");
    }
    return $record;
} else {
    throw new \Exception("Invalid input provided for username in
UserService::getUser()");
}
```



## Exercise 3 – PHP: Batch Read

Batch Read Tweets for a given user

Locate TweetService class in PHP project

1. In TweetService.batchGetTweets()
   1. Read User Record
   2. Determine how many Tweets the user has
   3. Create an array of Tweet Key instances
   4. Initiate Batch Read operation
   5. Output Tweets to the console

Read all the tweets for a given user. In TweetService.batchGetTweets(), add code similar to this:

1. Read a user record
2. Get the tweet count
3. Create a "list" of tweet keys

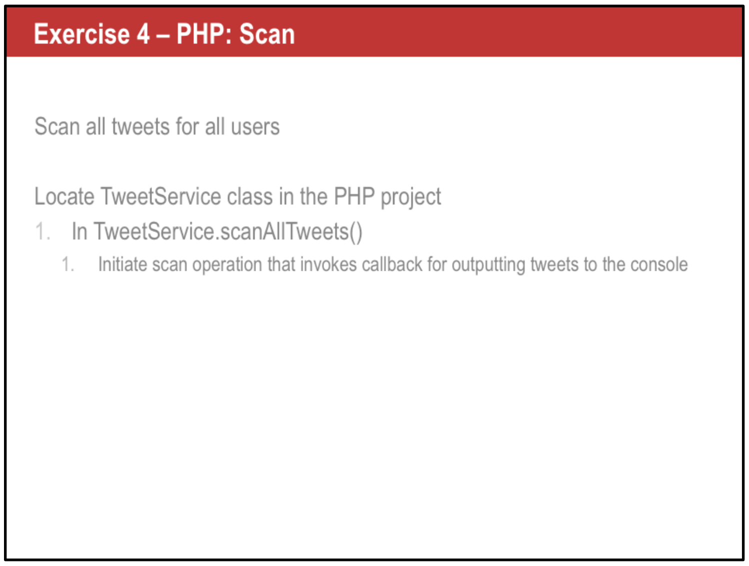
```
$record = $this->getUser($username, array('tweetcount'));
$tweet_count = intval($record['bins']['tweetcount']);
if ($this->annotate) display_code(__FILE__, __LINE__, 4);
$keys = array();
for ($i = 1; $i <= $tweet_count; $i++) {
    $keys[] = $this->getTweetKey($username, $i);
}
echo colorize("Batch-reading the user's tweets >", 'black',
true);
```

4. Perform a Batch operation to read all the tweets

```
if ($this->annotate) display_code(__FILE__, __LINE__, 6);
$status = $this->client->getMany($keys, $records);
if ($status !== Aerospike::OK) {
    echo fail();
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to batch-read the
tweets for $username");
}
echo success();
```

5. Then print out the tweets

```
echo colorize("Here are $username's tweets:\n", 'blue', true);
foreach ($records as $record) {
    echo colorize($record['bins']['tweet'], 'black')."\n";
}
```



## Exercise 4 – PHP: Scan

Scan all tweets for all users

Locate TweetService class in the PHP project

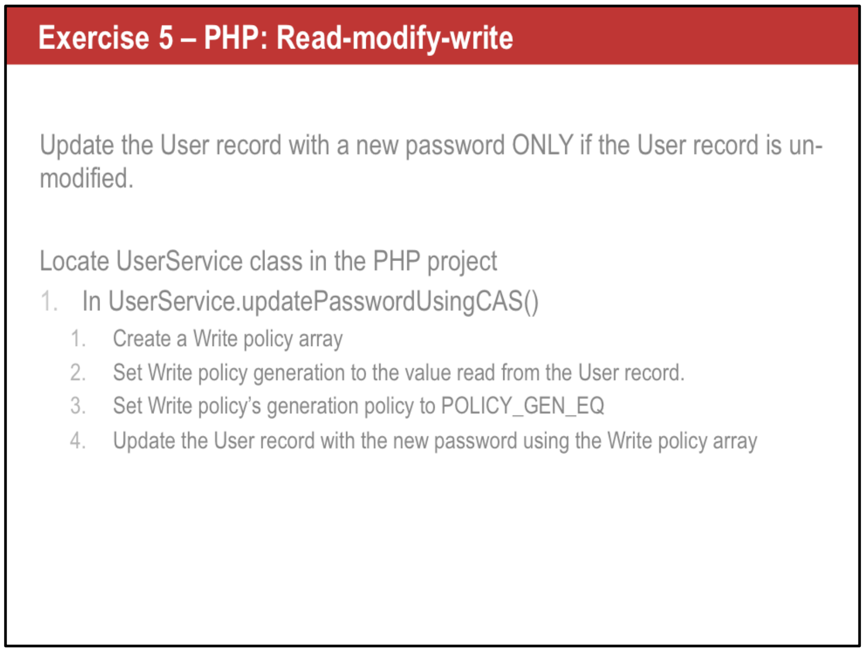
1. In TweetService.scanAllTweets()
    1. Initiate scan operation that invokes callback for outputting tweets to the console

Scan all the tweets for all users – warning – there could be a large result set.

In the TweetService.scanAllTweets(), add code similar to this:

1. Perform a Scan operation
2. process the results

```
$status = $this->client->scan('test','tweets', function
($record) {
    var_dump($record['bins']['tweet']);
}, array('tweet'));
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to scan
test.tweets");
}
```



## Exercise 5 – PHP: Read-modify-write

Update the User record with a new password ONLY if the User record is un-modified.

Locate UserService class in the PHP project

1. In UserService.updatePasswordUsingCAS()
   1. Create a Write policy array
   2. Set Write policy generation to the value read from the User record.
   3. Set Write policy's generation policy to POLICY_GEN_EQ
   4. Update the User record with the new password using the Write policy array

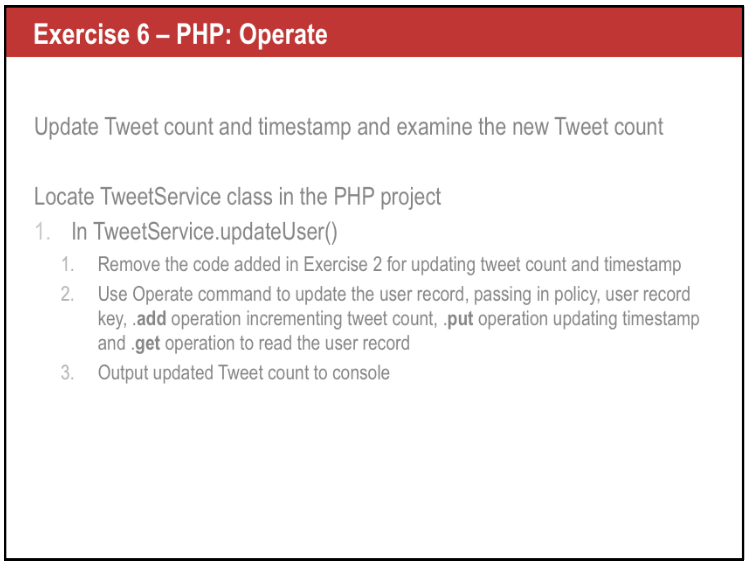
Update the User record with a new password ONLY if the User record is un-modified

In UserService.updatePasswordUsingCAS(), add code similar to this:

1. Create a Write policy array
2. Set Write policy generation to the value read from the User record.
3. Set Write policy's generation policy to POLICY_GEN_EQ
4. Update the User record with the new password using the Generation count from the meta data

```
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 1);
$status = $this->client->exists($key, $metadata);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to retrieve 
metadata for the record");
}
echo success();
var_dump($metadata);

echo colorize("Updating the user's password if the generation 
matches >", 'black', true);
if ($this->annotate) display_code(__FILE__, __LINE__, 4);
$bins = array('password' => $new_password);
$policy = array(Aerospike::OPT_POLICY_GEN =>
    array(Aerospike::POLICY_GEN_EQ, $metadata['generation']));
$status = $this->client->put($key, $bins, $metadata['ttl'], 
$policy);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Writing with 
POLICY_GEN_EQ failed due to generation mismatch");
}
```



## Exercise 6 – PHP: Operate

Update Tweet count and timestamp and examine the new Tweet count

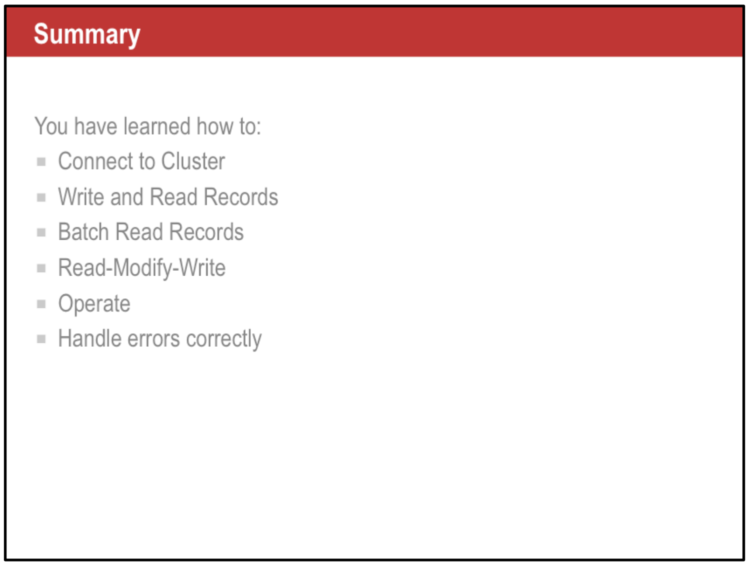
Locate TweetService class in the PHP project

1. In TweetService.updateUser()
   1. Remove the code added in Exercise 2 for updating tweet count and timestamp
   2. Use Operate command to update the user record, passing in policy, user record key, .**add** operation incrementing tweet count, .**put** operation updating timestamp and .**get** operation to read the user record
   3. Output updated Tweet count to console

Aerospike can perform multiple operations on a record in one transaction. Update the tweet count and timestamp in a user record and read the new tweet count.

In TweetService.updateUser()

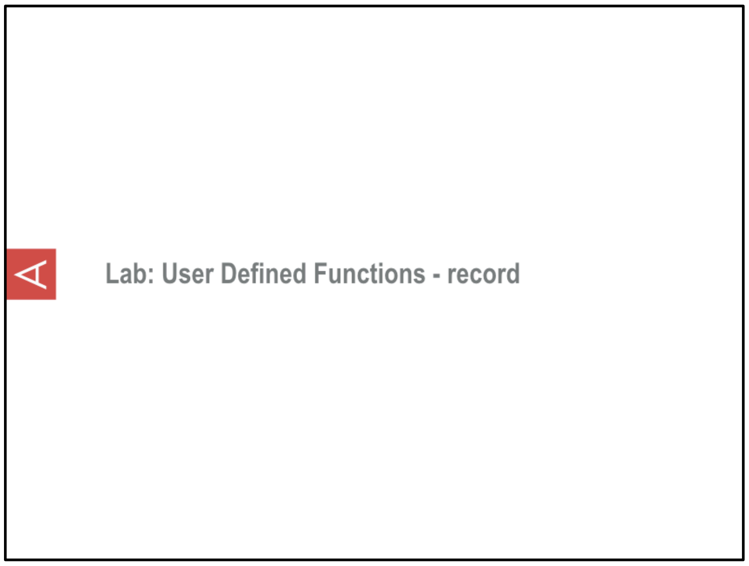
1. Delete the code added in Exercise 2



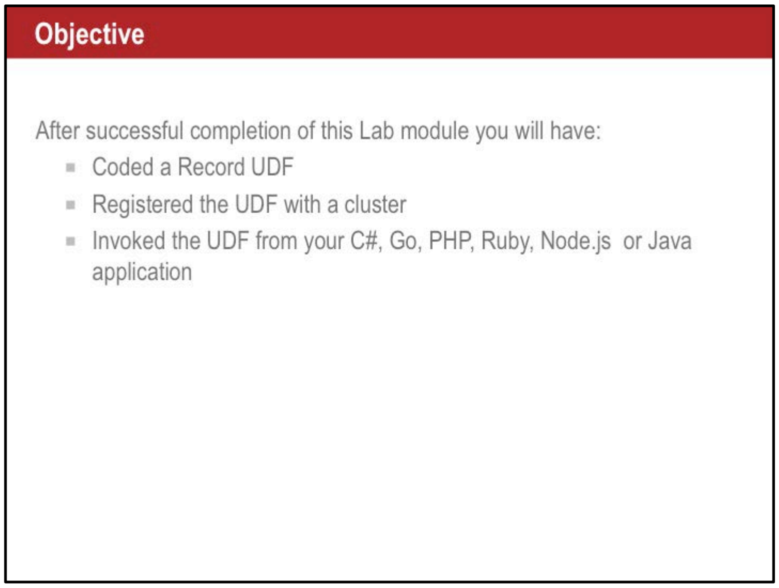
## Summary

You have learned how to:

- Connect to Cluster
- Write and Read Records
- Batch Read Records
- Read-Modify-Write
- Operate
- Handle errors correctly



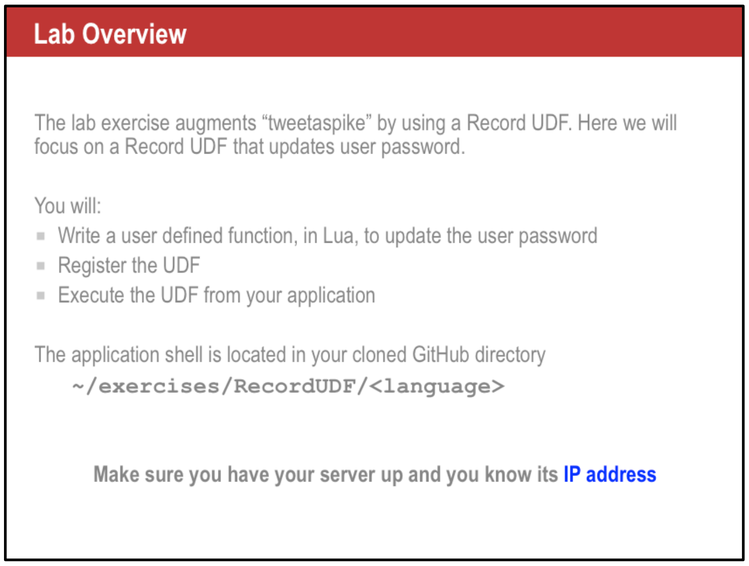
## Lab: User Defined Functions - record



## Objective

After successful completion of this Lab module you will have:

- Coded a Record UDF
- Registered the UDF with a cluster
- Invoked the UDF from your C#, Go, PHP, Ruby, Node.js  or Java application



## Lab Overview

The lab exercise augments "tweetaspike" by using a Record UDF. Here we will focus on a Record UDF that updates user password.

You will:

- Write a user defined function, in Lua, to update the user password
- Register the UDF
- Execute the UDF from your application

The application shell is located in your cloned GitHub directory

`~/exercises/RecordUDF/<language>`

**Make sure you have your server up and you know its IP address**

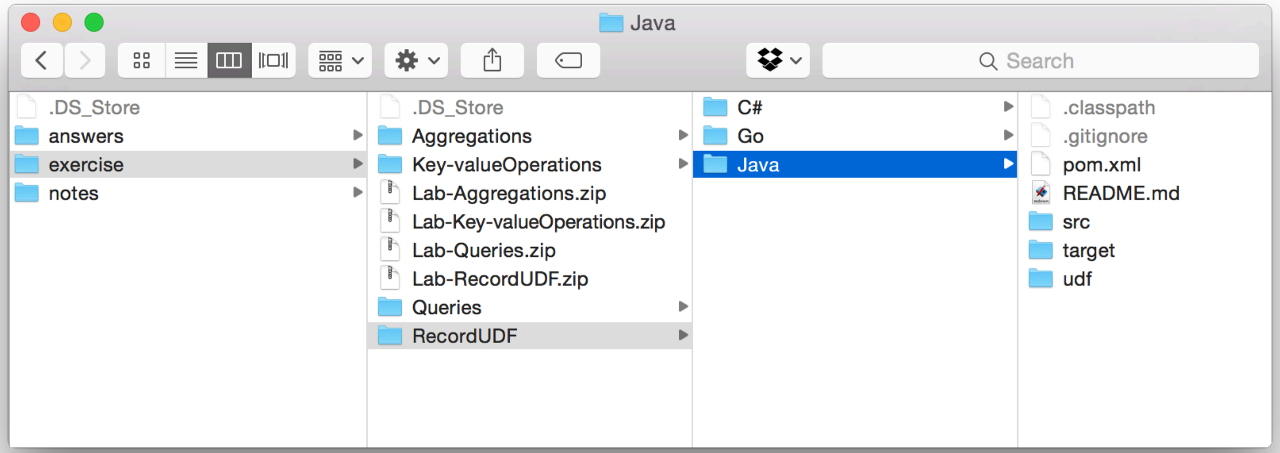
In your cloned or downloaded repository you will find the following directories:

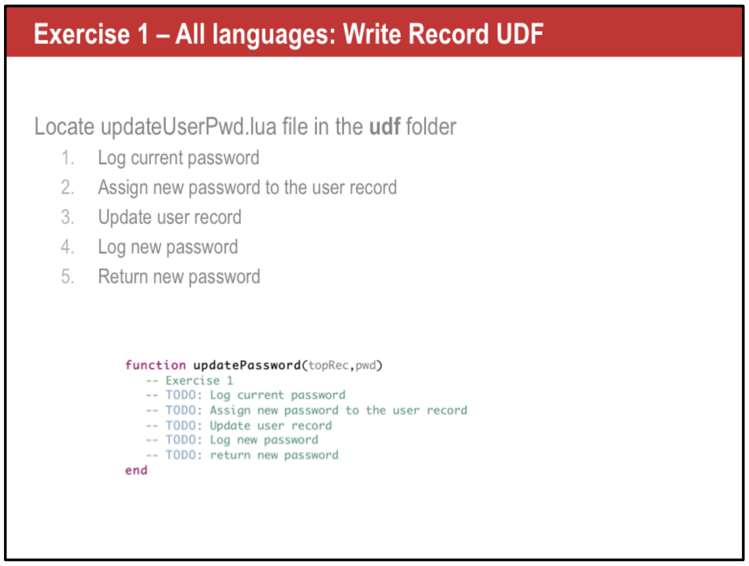
- Answers
- Exercise
- Notes

In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Node.js
- Go
- PHP
- Ruby

The exercises for this module are in the UDF directory and your will find a Project/SoluNon/Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.




## Exercise 1 – All languages: Write Record UDF

Locate updateUserPwd.lua file in the **udf** folder

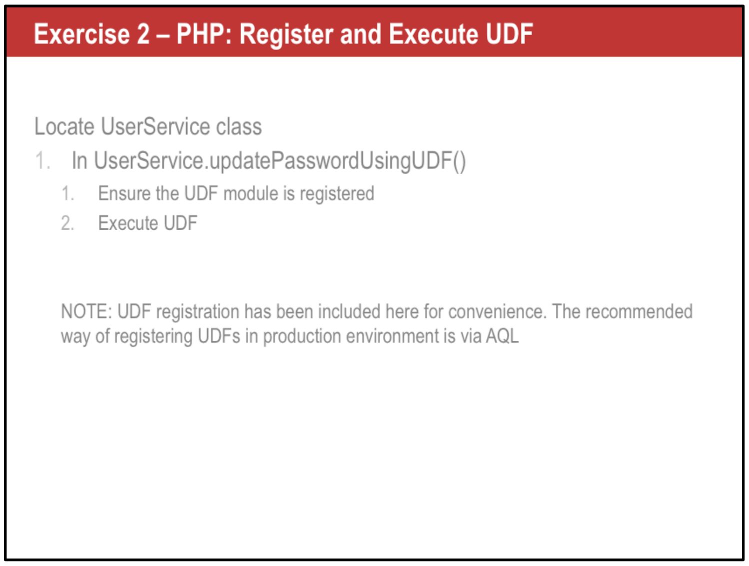
1. Log current password
2. Assign new password to the user record
3. Update user record
4. Log new password
5. Return new password

```
function updatePassword(topRec,pwd)
  -- Exercise 1
  -- TODO: Log current password
  -- TODO: Assign new password to the user record
  -- TODO: Update user record
  -- TODO: Log new password
  -- TODO: return new password
end
```

In this exercise you will create a record UDF that:

1. Logs the current password
2. Assigns a new password to the record, passed in via the pwd parameter
3. Updates the user record by calling **aerospike:update**(topRec)
4. Logs the new password
5. Returns the new password to the client

```
function updatePassword(topRec,pwd)
   -- Log current password
   debug("current password: " .. topRec['password'])
   -- Assign new password to the user record
   topRec['password'] = pwd
   -- Update user record
   aerospike:update(topRec)
   -- Log new password
   debug("new password: " .. topRec['password'])
   -- return new password
   return topRec['password']
end
```



## Exercise 2 – PHP: Register and Execute UDF

Locate UserService class

1. In UserService.updatePasswordUsingUDF()
   1. Ensure the UDF module is registered
   2. Execute UDF

NOTE: UDF registration has been included here for convenience. The recommended way of registering UDFs in production environment is via AQL

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma  cally register the UDF for convenience.

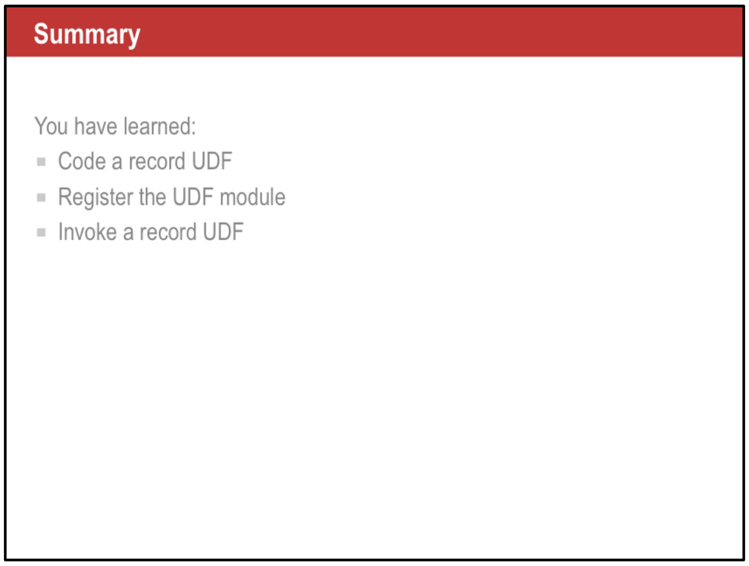
In UserService.updatePasswordUsing UDF(), locate these comments and add your code:

1. Ensure the UDF module is registered

```
$ok = $this->ensureUdfModule('udf/updateUserPwd.lua',
'updateUserPwd.lua');
if ($ok) echo success();
else echo fail();
$this->display_module('udf/updateUserPwd.lua');
```

2. Execute the UDF passing the new password as a parameter, to the UDF

```
echo colorize("Updating the user record >", 'black', true);
$key = $this->getUserKey($username);
if ($this->annotate) display_code(__FILE__, __LINE__, 7);
$args = array($new_password);
$status = $this->client->apply($key, 'updateUserPwd',
'updatePassword', $args);
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    echo fail();
    throw new Exception($this->client, "Failed to update
password for user $username");
}
```

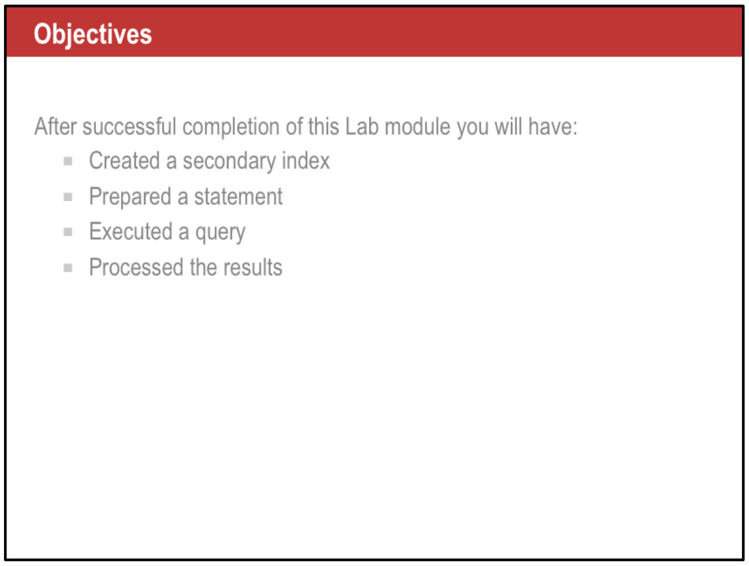


## Summary

You have learned:

- Code a record UDF
- Register the UDF module
- Invoke a record UDF

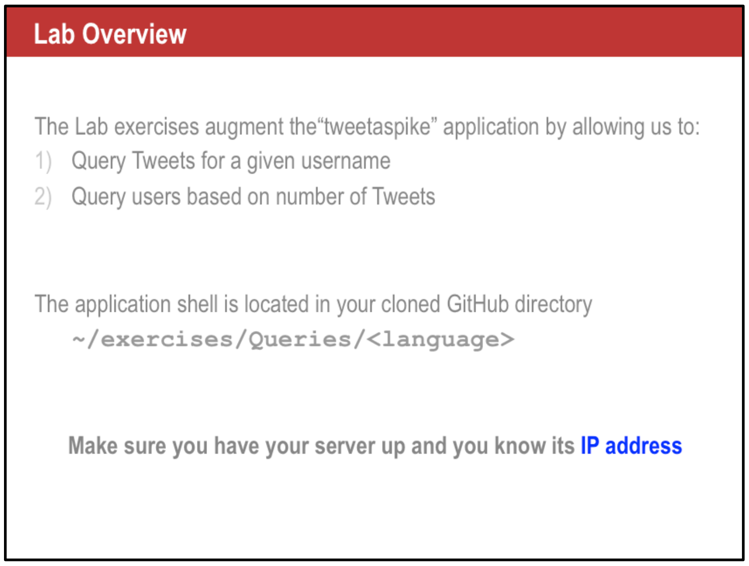
## Lab: Queries



## Objectives

After successful completion of this Lab module you will have:

- Created a secondary index
- Prepared a statement
- Executed a query
- Processed the results



## Lab Overview

The Lab exercises augment the"tweetaspike" application by allowing us to:

1) Query Tweets for a given username
2) Query users based on number of Tweets

The application shell is located in your cloned GitHub directory

`~/exercises/Queries/<language>`

**Make sure you have your server up and you know its IP address**

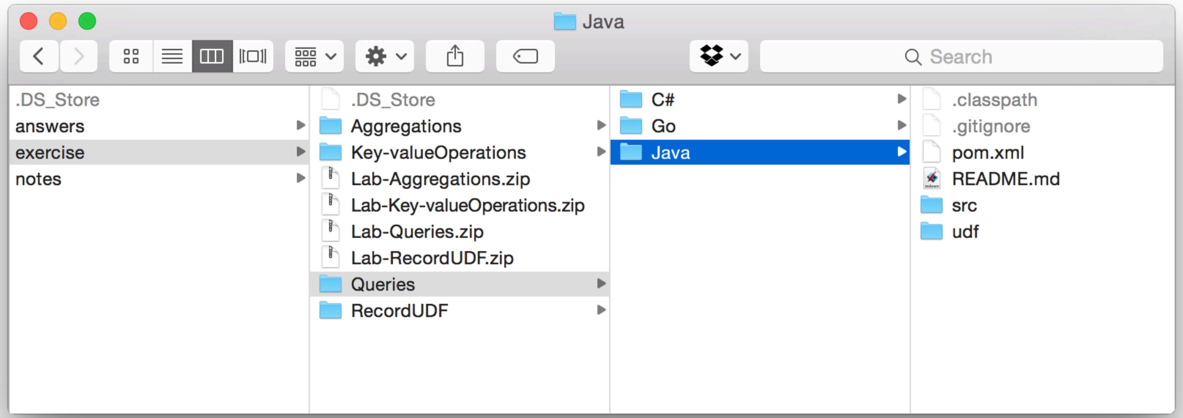
On your cloned or downloaded repository, you will find the following directories:

- Answers
- Exercise
- Notes

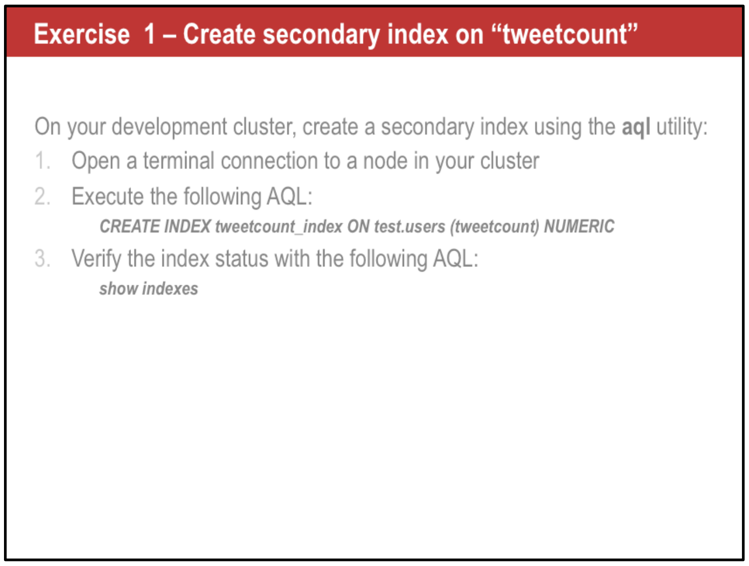
In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Go
- Node.js
- PHP
- Python

The exercises for this module are in the Queries directory and your will find a Project/SoluNon/ Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.



```
.DS_Store     .DS_Store                     C#     .classpath
answers       Aggregations                  Go     .gitignore
exercise      Key-valueOperations           Java   pom.xml
notes         Lab-Aggregations.zip                 README.md
              Lab-Key-valueOperations.zip          src
              Lab-Queries.zip                      udf
              Lab-RecordUDF.zip
              Queries
              RecordUDF
```


## Exercise 1 – Create secondary index on "tweetcount"

On your development cluster, create a secondary index using the **aql** utility:

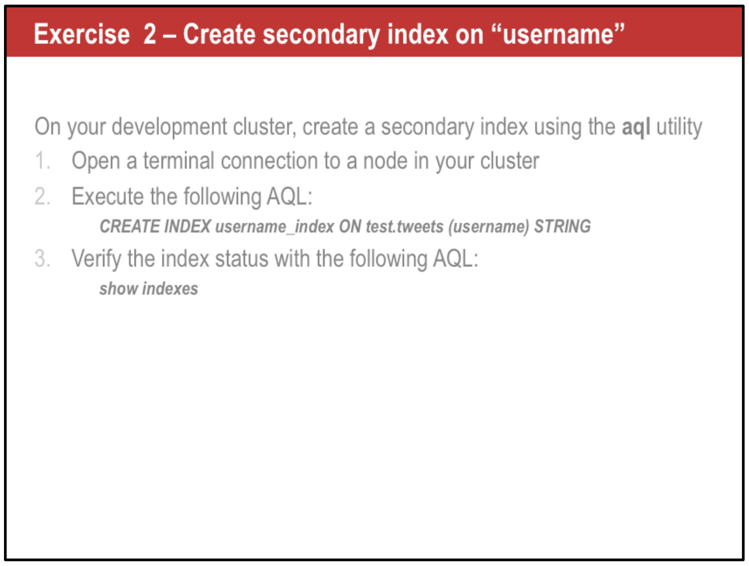
1. Open a terminal connection to a node in your cluster
2. Execute the following AQL:
   ***CREATE INDEX tweetcount_index ON test.users (tweetcount) NUMERIC***
3. Verify the index status with the following AQL:
   ***show indexes***

Logon on to your server instance and run **aql** to create a numeric index on *tweetcount*. At the prompt, enter the command:

***CREATE INDEX tweetcount_index ON test.users (tweetcount) NUMERIC***

Verify that the index has been created with the command:

***show indexes***


## Exercise 2 – Create secondary index on “username”

On your development cluster, create a secondary index using the **aql** utility

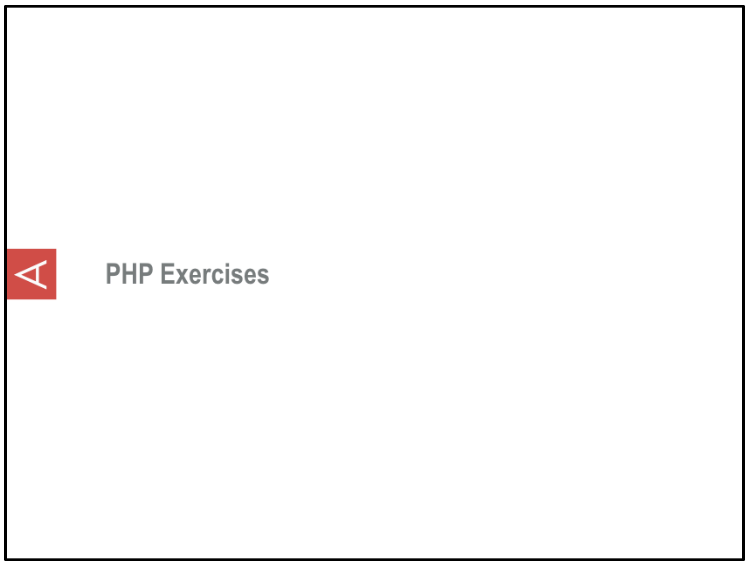
1. Open a terminal connection to a node in your cluster
2. Execute the following AQL:
   ***CREATE INDEX username_index ON test.tweets (username) STRING***
3. Verify the index status with the following AQL:
   ***show indexes***

Logon on to your server instance and run **aql** to create a string index on *username*. At the prompt, enter the command:

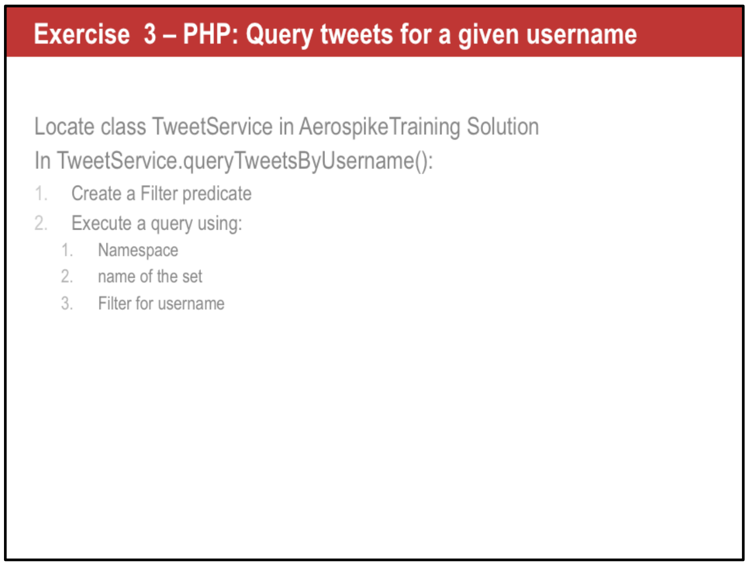
***CREATE INDEX username_index ON test.tweets (username) STRING***

Verify that the index has been created with the command:

***show indexes***



## PHP Exercises



## Exercise 3 – PHP: Query tweets for a given username

Locate class TweetService in AerospikeTraining Solution

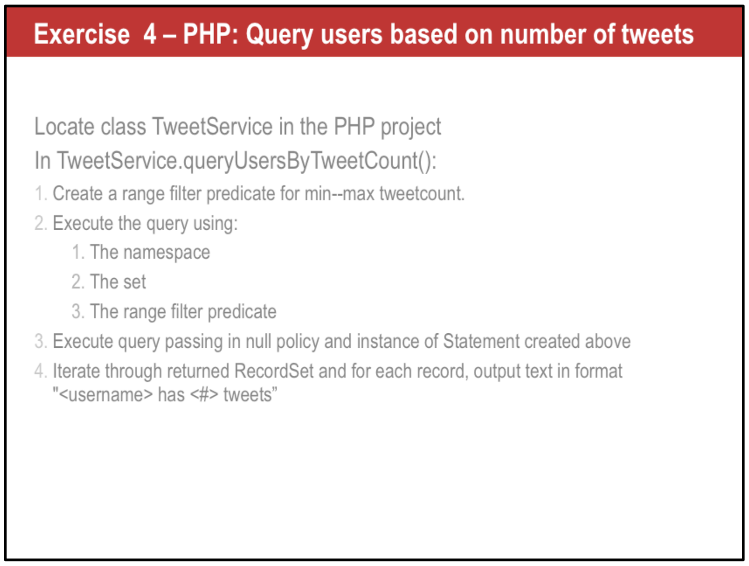
In TweetService.queryTweetsByUsername():

1. Create a Filter predicate
2. Execute a query using:
   1. Namespace
   2. name of the set
   3. Filter for username

In TweetService.queryTweetsByUsername() add your code:

1. Create a Filter predicate
2. Execute a query using:
   1. the Namespace
   2. the Set name
   3. the Filter predicate to qualify the user name

```
$where = $this->client->predicateEquals('username', $username);
$status = $this->client->query('test','tweets', $where, function
($record) {
    var_dump($record['bins']['tweet']);
}, array('tweet'));
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to query
test.tweets");
}
```



## Exercise 4 – PHP: Query users based on number of tweets

Locate class TweetService in the PHP project

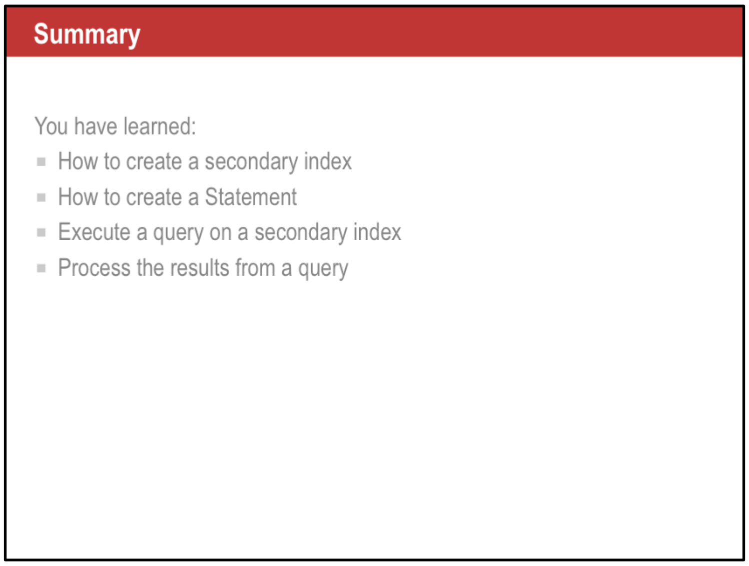
In TweetService.queryUsersByTweetCount():

1. Create a range filter predicate for min--max tweetcount.
2. Execute the query using:
    1. The namespace
    2. The set
    3. The range filter predicate
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

In TweetService.queryUsersByTweetCount(), add your code:

1. Create a range filter predicate for min max tweetcount.
2. Execute the query using:
    1. The namespace
    2. The set
    3. The range filter predicate
3. Execute query passing in null policy and instance of Statement created above
4. Iterate through returned RecordSet and for each record, output text in format "<username> has <#> tweets"

```
$where = $this->client->predicateBetween('tweetcount', $min,
$max);
$status = $this->client->query('test','users', $where, function
($rec) {
    echo colorize("{$rec['bins']['username']} has
{$rec['bins']['tweetcount']} tweets", 'black')."\n";
});
if ($status !== Aerospike::OK) {
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to query
test.users");
}
```

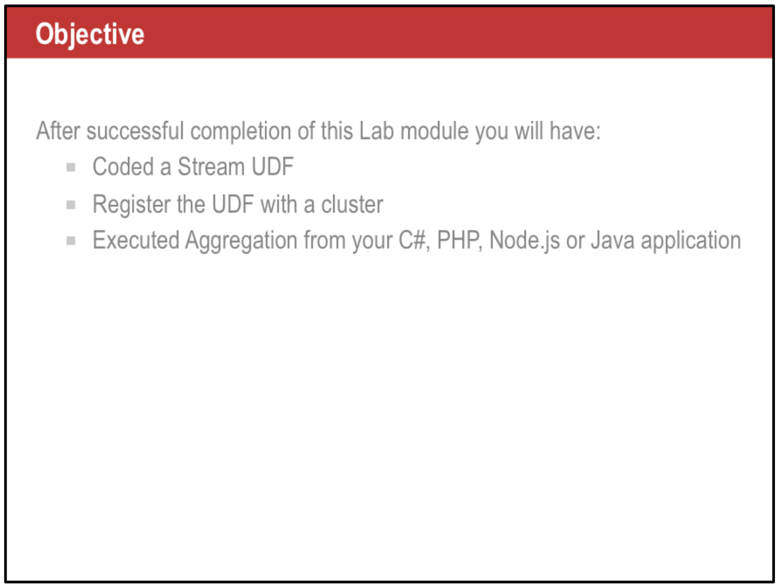


## Summary

You have learned:

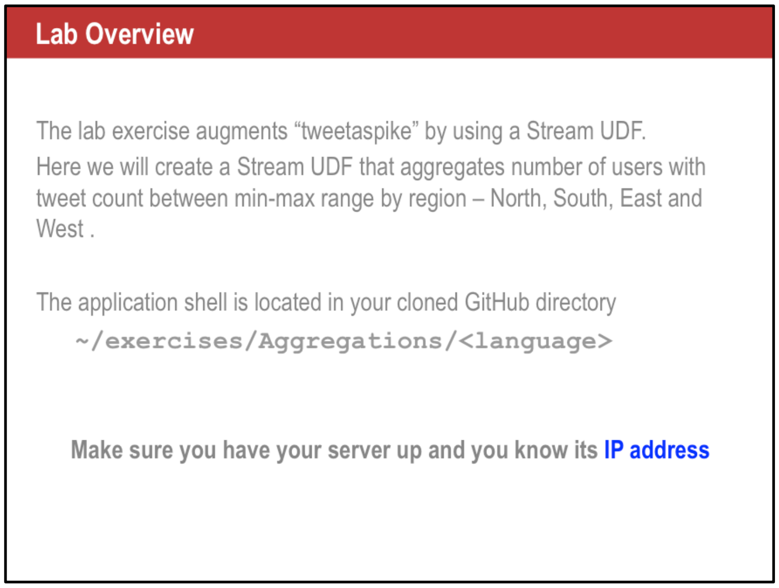
- How to create a secondary index
- How to create a Statement
- Execute a query on a secondary index
- Process the results from a query

## Lab: Aggregations


## Objective

After successful completion of this Lab module you will have:

- Coded a Stream UDF
- Register the UDF with a cluster
- Executed Aggregation from your C#, PHP, Node.js or Java application



## Lab Overview

The lab exercise augments "tweetaspike" by using a Stream UDF.
Here we will create a Stream UDF that aggregates number of users with tweet count between min-max range by region – North, South, East and West .

The application shell is located in your cloned GitHub directory

```
~/exercises/Aggregations/<language>
```

**Make sure you have your server up and you know its IP address**

In your cloned or downloaded repository, you will find the following directories:

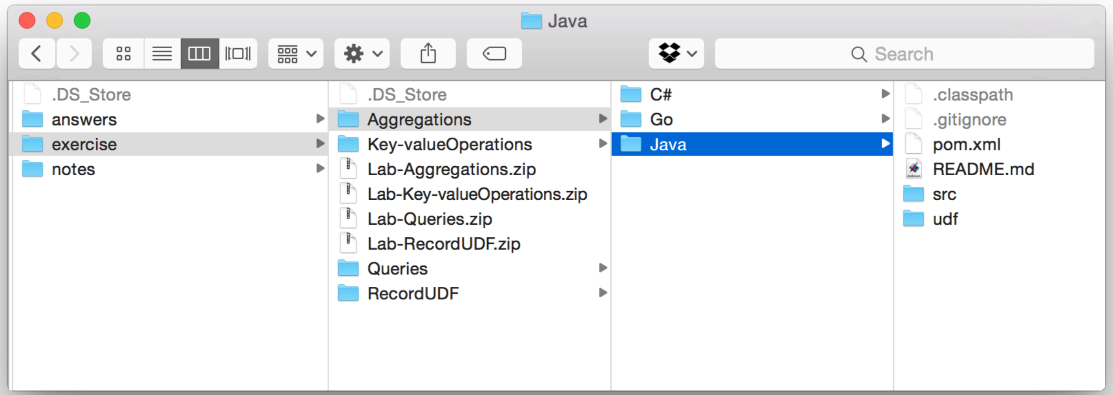
- Answers
- Exercise
- Notes

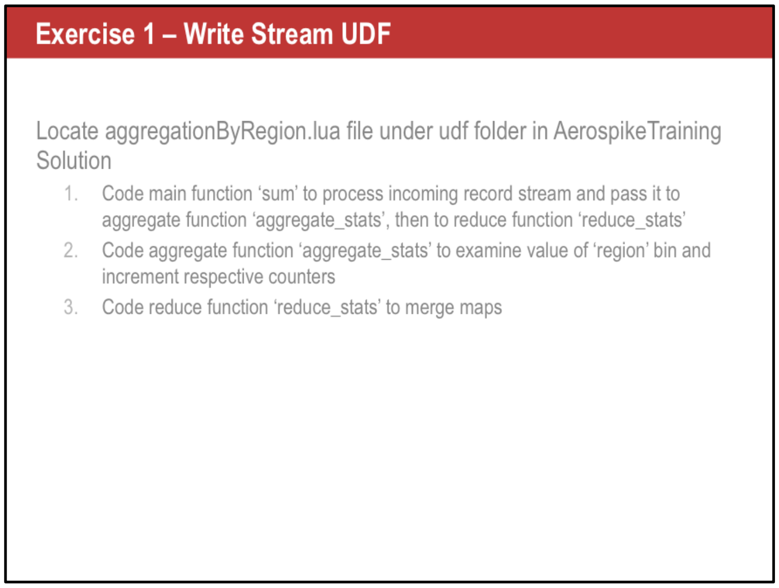
In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Node.js
- PHP
- Python

The exercises for this module are in the Aggrega  ons directory and your will find a Project/SoluNon/ Codebase that is partly complete. Your tasks is to complete the code as outlined in each exercise.

**Make sure you have your server up and you know its IP address**





## Exercise 1 – Write Stream UDF

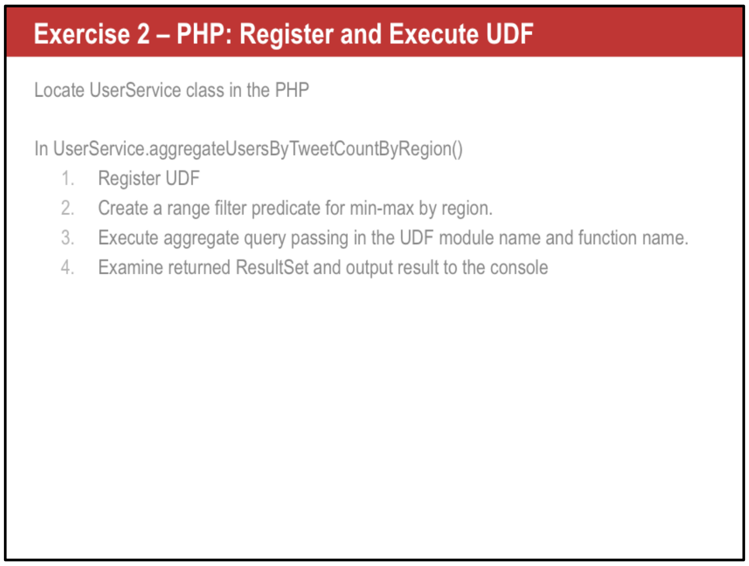
Locate aggregationByRegion.lua file under udf folder in AerospikeTraining Solution

1. Code main function 'sum' to process incoming record stream and pass it to aggregate function 'aggregate_stats', then to reduce function 'reduce_stats'
2. Code aggregate function 'aggregate_stats' to examine value of 'region' bin and increment respective counters
3. Code reduce function 'reduce_stats' to merge maps

In this exercise you will create a Stream UDF module that:

- Aggregates (sums) tweets by region – The aggregate_stats() funcNon is invoked one for each element in the stream.
- Reduces the aggrega ons into a single Map of values – The reduce_stats() funcNon is invoked once for each data parNNon, once for each node in the cluster, and finally once on the client.
- The sum() funcNon configures the stream processing, and it is the funcNon invoked by the Client.

```lua
local function aggregate_stats(map,rec)
  -- Examine value of 'region' bin in record rec and increment respective counter in the map
  if rec.region == 'n' then
      map['n'] = map['n'] + 1
  elseif rec.region == 's' then
      map['s'] = map['s'] + 1
  elseif rec.region == 'e' then
      map['e'] = map['e'] + 1
  elseif rec.region == 'w' then
      map['w'] = map['w'] + 1
  end
  -- return updated map
  return map
end
local function reduce_stats(a,b)
  -- Merge values from map b into a
  a.n = a.n + b.n
  a.s = a.s + b.s
  a.e = a.e + b.e
  a.w = a.w + b.w
  -- Return updated map a
  return a
end
function sum(stream)
  -- Process incoming record stream and pass it to aggregate function, then to reduce function
return stream : aggregate(map{n=0,s=0,e=0,w=0},aggregate_stats) : reduce(reduce_stats)
end
```



## Exercise 2 – PHP: Register and Execute UDF

Locate UserService class in the PHP

In UserService.aggregateUsersByTweetCountByRegion()

1. Register UDF
2. Create a range filter predicate for min-max by region.
3. Execute aggregate query passing in the UDF module name and function name.
4. Examine returned ResultSet and output result to the console

In this exercise you will register and invoke the UDF created in Exercise 1.

We will programma  cally register the UDF for.

In UserService.aggregateUsersByTweetCountByRegion(), add your code to look like this:

1. Register the UDF with an API call

```
$ok = $this->ensureUdfModule('udf/aggregationByRegion.lua',
'aggregationByRegion.lua');
```

2. Create a range filter predicate

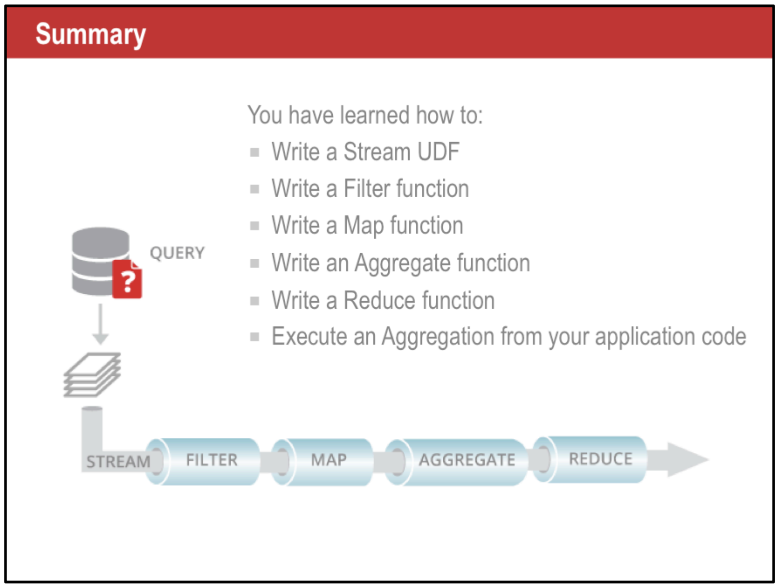
```
$where = $this->client->predicateBetween('tweetcount',
$min, $max);
$args = array();
echo colorize("Call the aggregation stream UDF >", 'black',
true);
```

3. Execute the aggrega  on

```
$status = $this->client->aggregate('test', 'users', $where,
'aggregationByRegion', 'sum', $args, $returned);
if ($status !== Aerospike::OK) {
    echo fail();
    // throwing an \Aerospike\Training\Exception
    throw new Exception($this->client, "Failed to execute
the stream UDF");
}
```

4. Examine the ResultSet

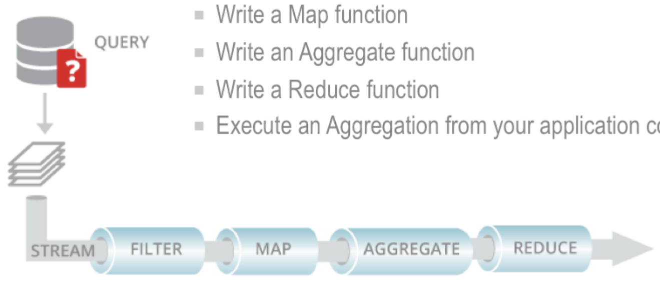
```
echo success();
var_dump($returned);
```



## Summary

You have learned how to:

- Write a Stream UDF
- Write a Filter function
- Write a Map function
- Write an Aggregate function
- Write a Reduce function
- Execute an Aggregation from your application code



AEROSPIKE