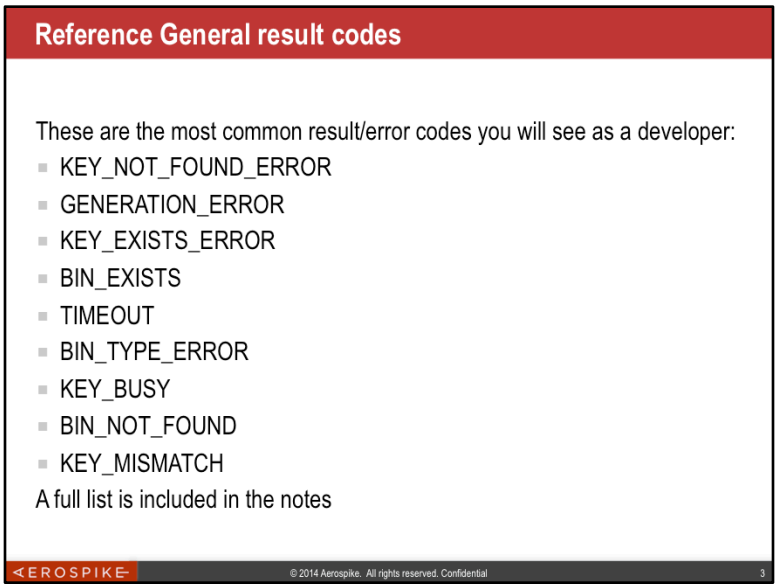

Lab: Key-value Operations
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
1

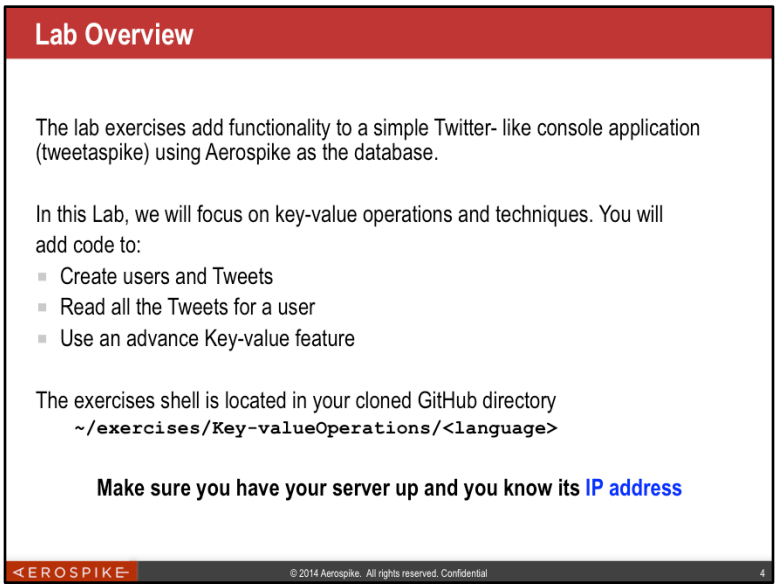
Objective

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



Code	Symbolic name	Description
0	OK	Operation was successful.
1	SERVER_ERROR	Unknown error in the server. Contact Aerospike support.
2	KEY_NOT_FOUND	On retrieving, touching or replacing a record that doesn't exist.
3	GENERATION_ERROR	On modifying a record with unexpected generation.
4	PARAMETER_ERROR	Bad parameter(s) were passed in database operation call.
5	KEY_EXISTS_ERROR	On create-only (write unique) operations on a record that already exists.
6	BIN_EXISTS_ERROR	On create-only (write unique) operations on a bin that already exists.
7	CLUSTER_KEY_MISMATCH	Expected cluster ID was not received – Cluster is changing during a Scan
8	SERVER_MEM_ERROR	Server has run out of memory.
9	TIMEOUT	Client or server has timed out.
10	NO_XDS	XDR product is not available.
11	SERVER_NOT_AVAILABLE	Server is not accepting requests.
12	BIN_TYPE_ERROR	Operation is not supported with configured bin type (single-bin or multi-bin).
13	RECORD_TOO_BIG	Record size exceeds limit.
14	KEY_BUSY	Too many concurrent operations on the same record.
15	SCAN_ABORT	Scan aborted by server.
16	UNSUPPORTED_FEATURE	Unsupported Server Feature (e.g. Scan + UDF).
17	BIN_NOT_FOUND	Specified bin name does not exist in record.
18	DEVICE_OVERLOAD	The server node's storage device(s) can't keep up with the write load.
19	KEY_MISMATCH	Key type mismatch. – Digest collision – never been seen
100	UDF_BAD_REPONSE	A user defined function returned an error code.



On your USB stick, or your unzipped directory, you will find the following directories:

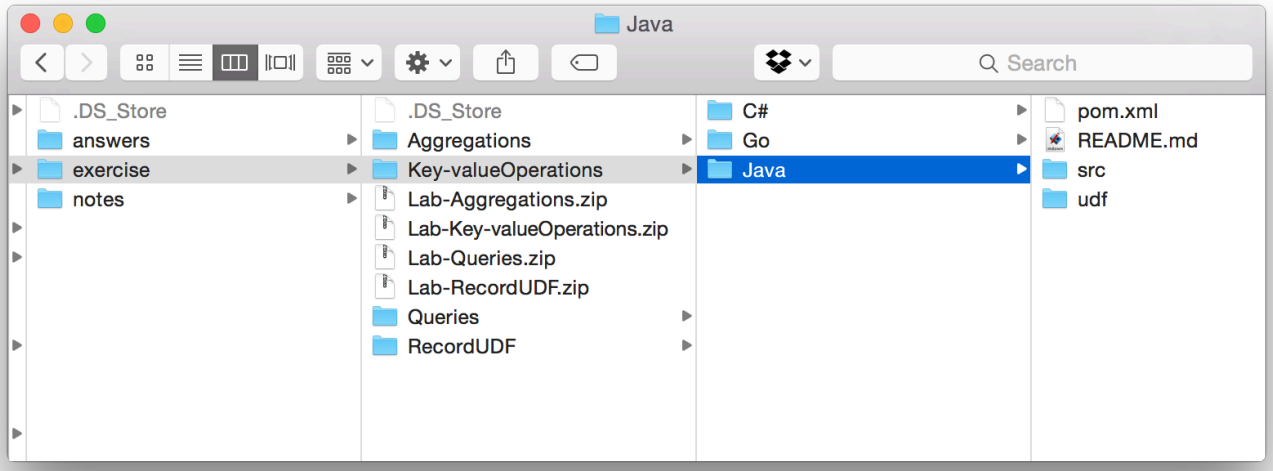
- Answers
- Exercise
- Notes

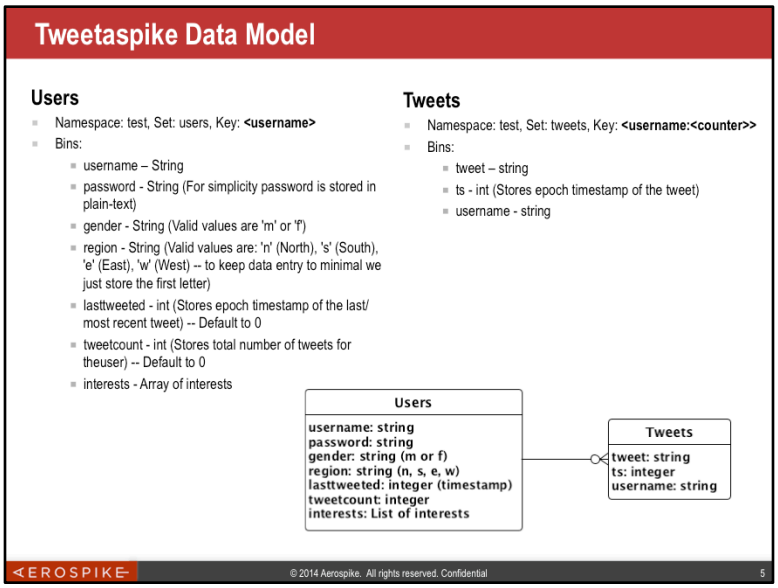
In the exercise directory, select the subdirectory for your programming language:

- C#
- Java
- Go
- PHP
- Node

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





Users

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

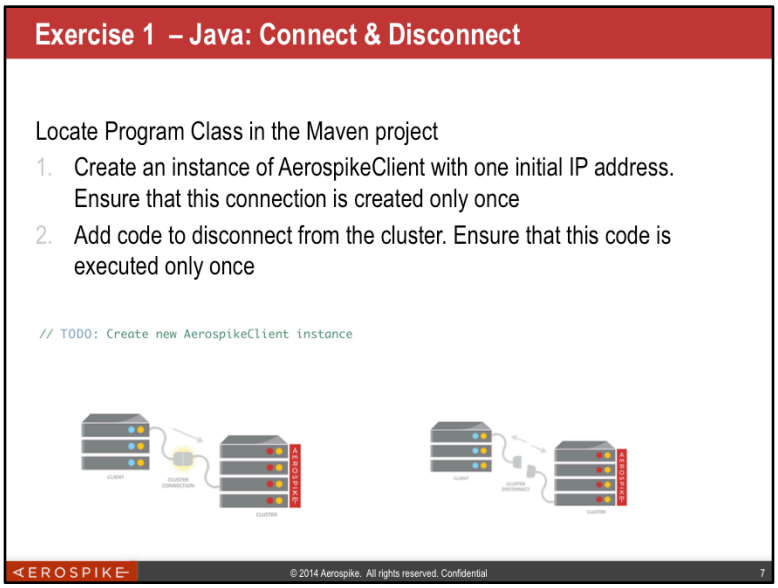
Bin name	Type	Comment
username	String	
password	String	For simplicity password is stored in plain-text
region	String	Valid values are: 'n' (North), 's' (South), 'e' (East), 'w' (West) -- to keep data entry to minimal we just store the first letter
lasttweeted	Integer	Stores epoch timestamp of the last/most 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 timestamp of the tweet
username	String	User name of the tweeter


Java Exercises
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
6



In this exercise you will connect to a Cluster by creating an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

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

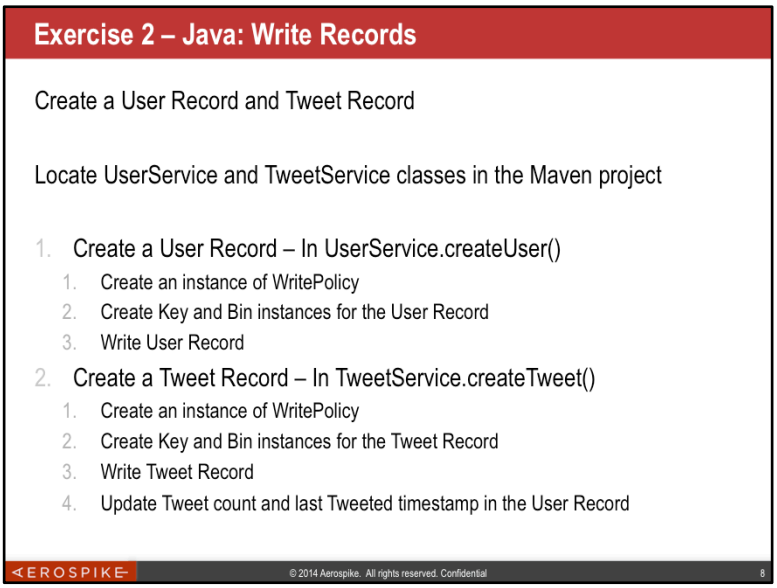
1. In the constructor for the class Program, add code similar to this;

```
// Establish a connection to Aerospike cluster
this.client = new AerospikeClient("192.168.1.15", 3000);
```

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

2. At the end of method work(), 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.

```
if (client != null && client.isConnected()) {
     // Close Aerospike server connection
     client.close();
}
```



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

1. Create a WritePolicy

```
// Write record
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

2. Create Key and Bin instances

```
Key key = new Key("test", "users", username);
Bin bin1 = new Bin("username", username);
Bin bin2 = new Bin("password", password);
Bin bin3 = new Bin("gender", gender);
Bin bin4 = new Bin("region", region);
Bin bin5 = new Bin("lasttweeted", 0);
Bin bin6 = new Bin("tweetcount", 0);
Bin bin7 = Bin.asList("interests",
           Arrays.asList(interests.split(",")));
```

3. Write a user record using the Key and Bins

```
client.put(wPolicy, key, bin1, bin2, bin3, bin4,
                         bin5, bin6, bin7);
```

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

1. Read user record

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
```

2. Create a WritePolicy

```
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

3. Create Key and Bin instances

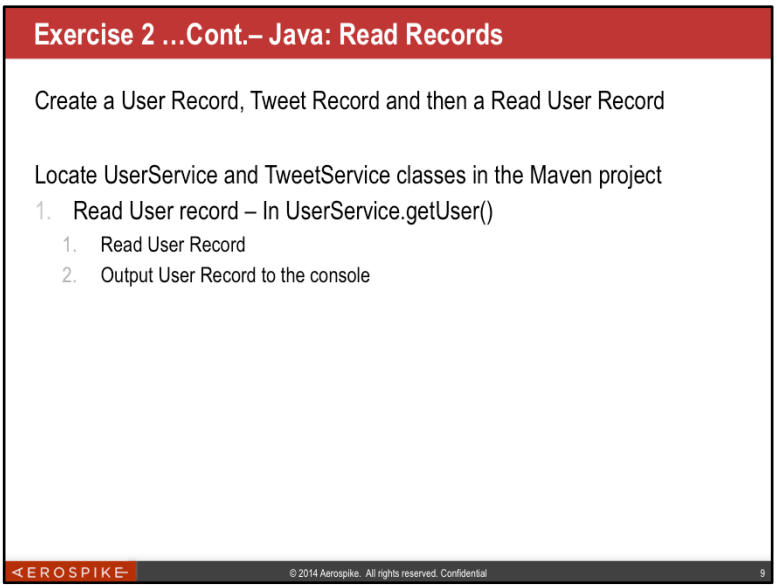
```
tweetKey = new Key("test", "tweets", username + ":"
                                   + nextTweetCount);
Bin bin1 = new Bin("tweet", tweet);
Bin bin2 = new Bin("ts", ts);
Bin bin3 = new Bin("username", username);
```

4. Write a tweet record using the Key and Bins

```
client.put(wPolicy, tweetKey, bin1, bin2, bin3);
console.printf("\nINFO: Tweet record created!\n");
```

5. Update the user record with tweet count

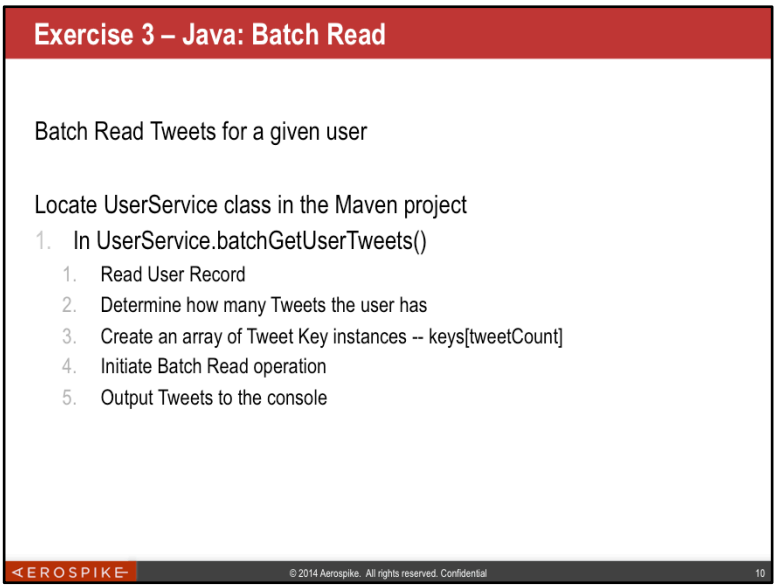
```
// Update tweet count and last tweet'd timestamp in the user
// record
long ts = getTimeStamp();
updateUser(client, userKey, wPolicy, ts, nextTweetCount);
```



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

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
if (userRecord != null) {
      console.printf("\nINFO: User record read successfully! Here are the details:\n");
      console.printf("username:   " + userRecord.getValue("username") + "\n");
      console.printf("password:   " + userRecord.getValue("password") + "\n");
      console.printf("gender:     " + userRecord.getValue("gender") + "\n");
      console.printf("region:     " + userRecord.getValue("region") + "\n");
      console.printf("tweetcount: " + userRecord.getValue("tweetcount") + "\n");
      console.printf("interests:  " + userRecord.getValue("interests") + "\n");
} else {
      console.printf("ERROR: User record not found!\n");
}
```



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

1. Read a user record

```
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
```

2. Get the tweet count

```
// Get how many tweets the user has
int tweetCount = (Integer) userRecord.getValue("tweetcount");
```

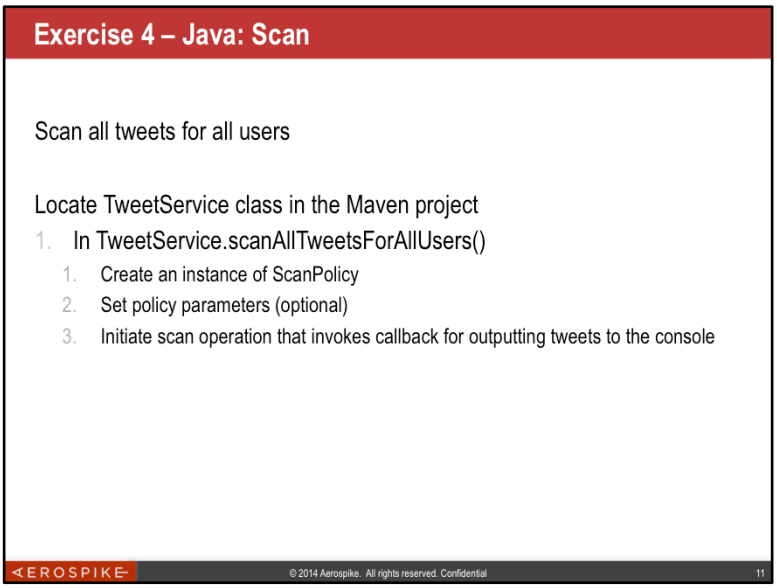
3. Create a "list" of tweet keys

```
// Create an array of keys so we can initiate batch read
// operation
Key[] keys = new Key[tweetCount];
for (int i = 0; i < keys.length; i++) {
     keys[i] = new Key("test", "tweets",
                (username + ":" + (i + 1)));
}
console.printf("\nHere's " + username + "'s tweet(s):\n");
```

4. Perform a Batch operation to read all the tweets

5. Then print out the tweets

```
// Initiate batch read operation
if (keys.length > 0){
     Record[] records = client.get(new Policy(), keys);
     for (int j = 0; j < records.length; j++) {
          console.printf(records[j].getValue("tweet").toString() + "\n");
     }
}
```



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

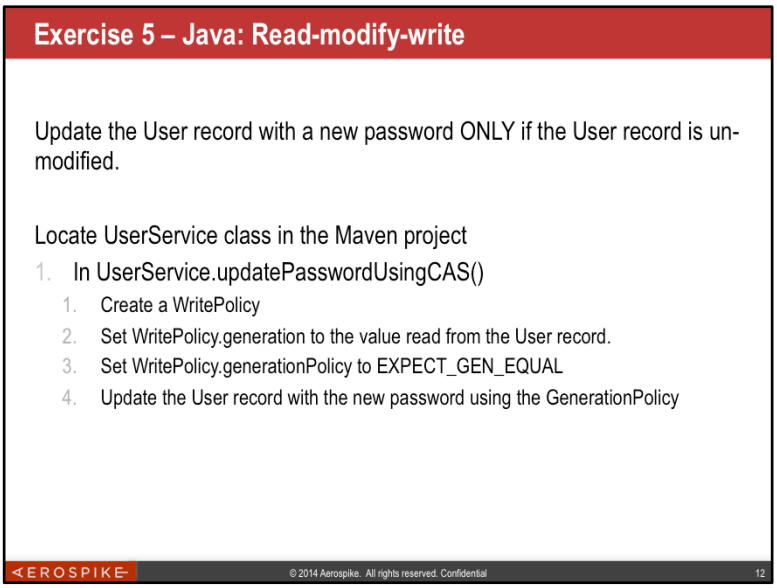
In the TweetService.scanAllTweetsForAllUsers(), add code similar to this:

1. Create a ScanPolicy
2. Set policy parameters

```
// Java Scan
ScanPolicy policy = new ScanPolicy();
policy.concurrentNodes = true;
policy.priority = Priority.LOW;
policy.includeBinData = true;
```

3. Perform a Scan operation and process the results

```
client.scanAll(policy, "test", "tweets", new ScanCallback() {

     public void scanCallback(Key key, Record record)
                                   throws AerospikeException {

     console.printf(record.getValue("tweet") + "\n");

     }
}, "tweet");
```

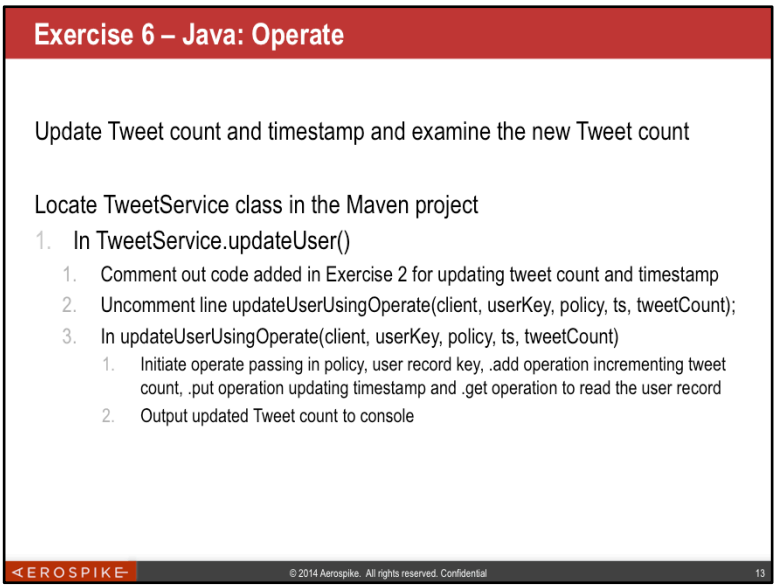


Update the User record with a new password ONLY if the User record is un-modified

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

1. Create a WritePolicy
2. Set WritePolicy.generation to the value read from the User record.
3. Set WritePolicy.generationPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the GenerationPolicy

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.get(null, userKey);
if (userRecord != null)
{
      // Get new password
      String password;
      console.printf("Enter new password for " + username + ":");
      password = console.readLine();

      WritePolicy writePolicy = new WritePolicy();
      // record generation
      writePolicy.generation = userRecord.generation;
      writePolicy.generationPolicy = GenerationPolicy.EXPECT_GEN_EQUAL;
      // password Bin
      passwordBin = new Bin("password", Value.get(password));
      client.put(writePolicy, userKey, passwordBin);

      console.printf("\nINFO: The password has been set to: " + password);
}
```



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. Comment out the code added in exercise 2

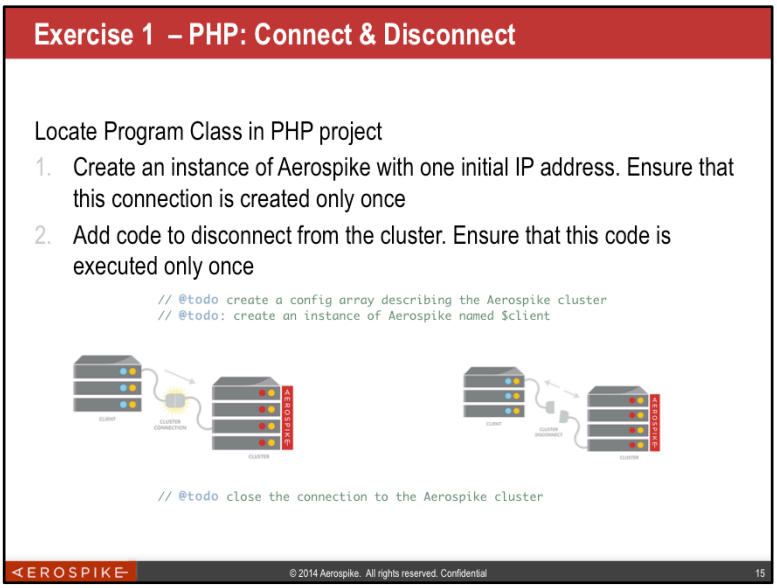
2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

3. In TweetService.updateUserUsingOperate() , add code similar to this:

```
Record record = client.operate(policy, userKey,
     Operation.add(new Bin("tweetcount", 1)),
     Operation.put(new Bin("lasttweeted", ts)),
     Operation.get());
```


PHP Exercises
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
14



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.

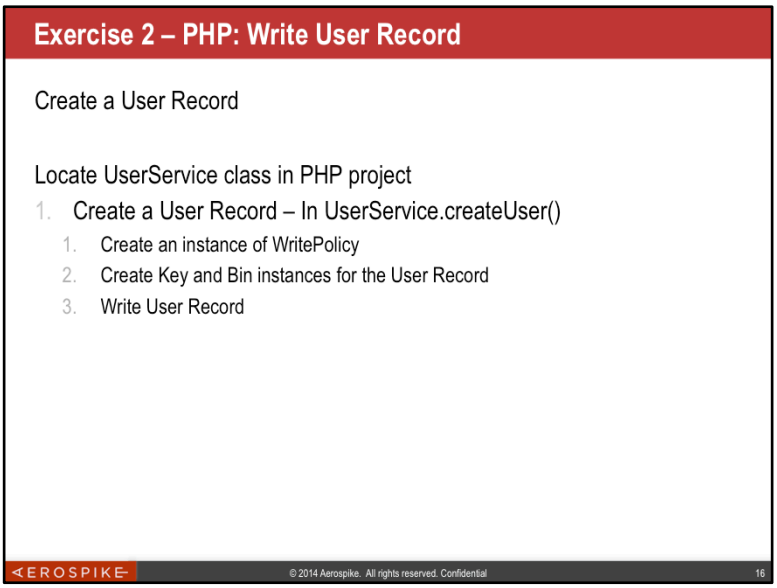
1. In the constructor for the class Program, add code similar to this;

```
$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.

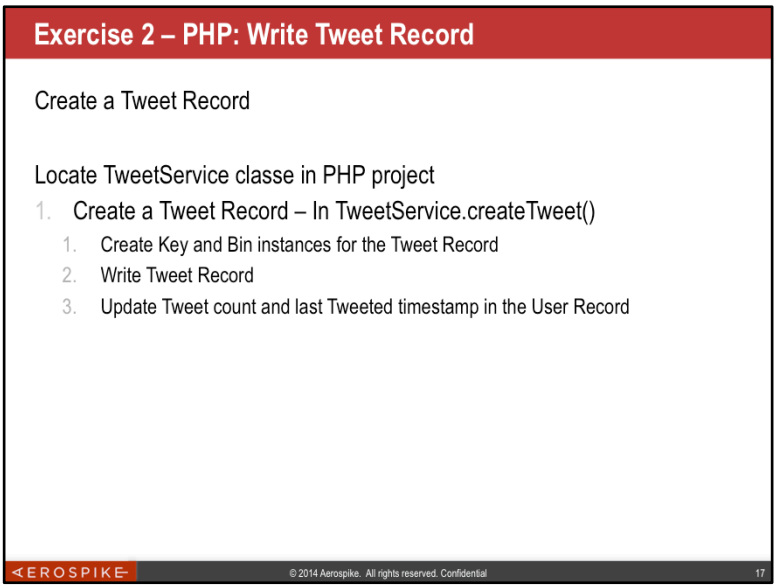
```
$client->close();
```



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");
}
```



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

1. Read user record

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

2. Create Key and Bin instances

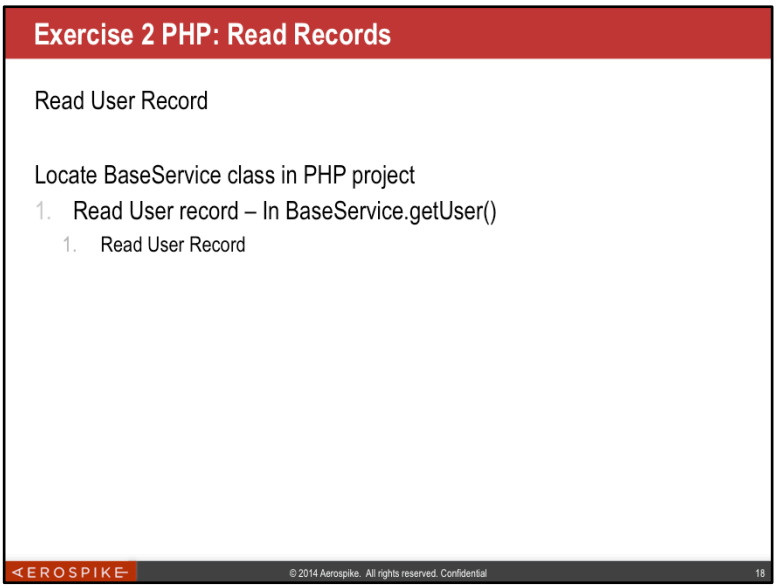
```
$bins = array();
// Get a tweet
echo colorize("Enter tweet for $username: ");
$bins['tweet'] = trim(readline());
$bins['ts'] = time() * 1000;
$bins['username'] = $username;
```

3. Write a tweet record using the Key and Bins

```
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");
}
```

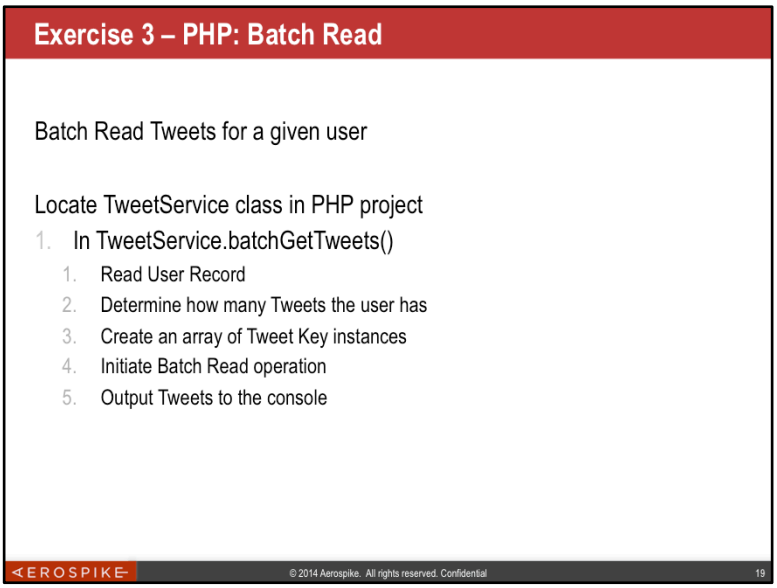
4. Update the user record with tweet count

```
echo success();
return $this->updateUser($username, $bins['ts'], $tweet_count);
```



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()");
}
```



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

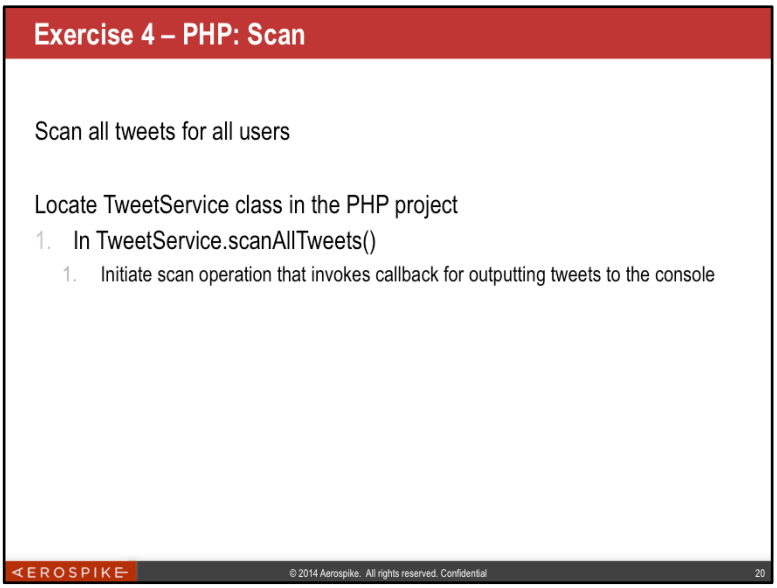
```
$record = $this->getUser($username, array('tweetcount'));
$tweet_count = intval($record['bins']['tweetcount']);
if ($this->annotate) display_code(__FILE__, __LINE__, 4);
$keys = array();
for ($i = 1; $i <= $tweet_count; $i++) {
    $keys[] = $this->getTweetKey($username, $i);
}
```

4. Perform a Batch operation to read all the tweets

```
echo colorize("Batch-reading the user's tweets >", 'black',
true);
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");
}
```

5. Then print out the tweets

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

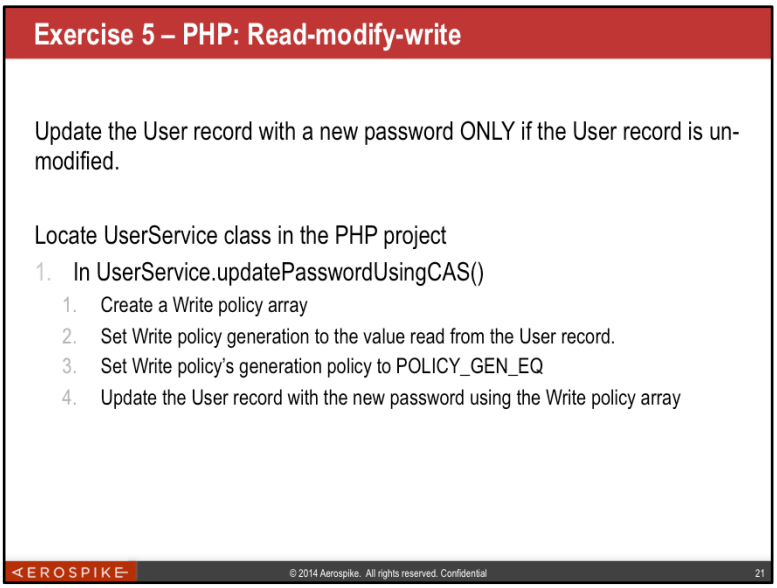


Scan all the tweets for all users – warning – there could be a large result set.

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");
}
```

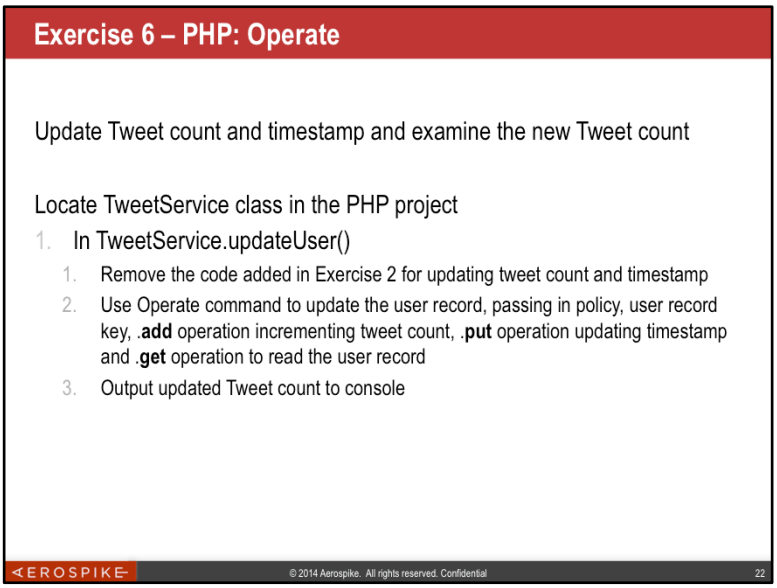


Update the User record with a new password ONLY if the User record is un-modified

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");
}
```

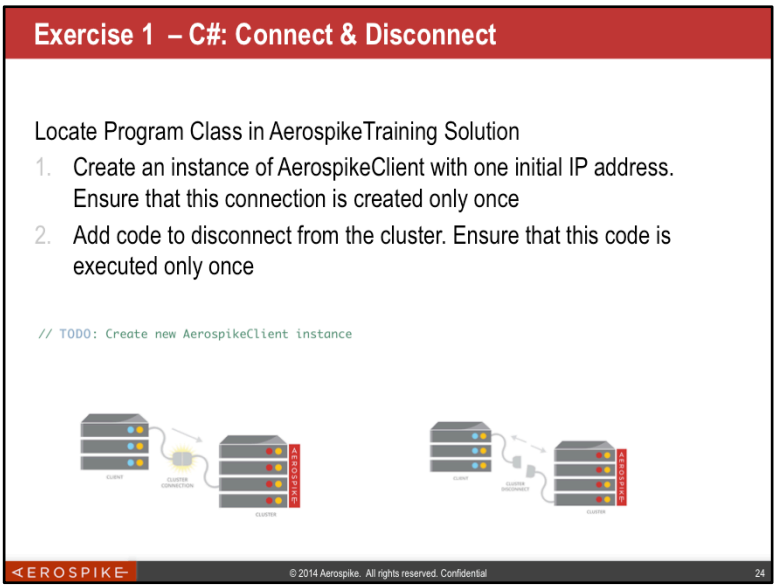


Aerospike can perform multiple operations on a record in one transaction. Update the tweet count and timestamp in a user record and read the new tweet count.

In TweetService.updateUser()

1. Delete the code added in Exercise 2


C# Exercises
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
23



In this exercise you will connect to a Cluster by creating an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

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

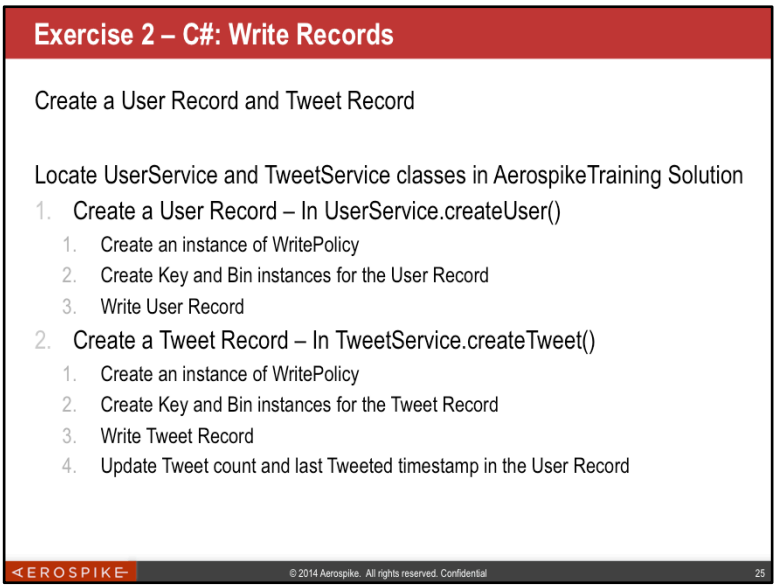
1. In the Main() method for the class Program, add code similar to this;

```
// Specify IP of one of the nodes in the cluster
string asServerIP = "172.16.159.206";
// Specity Port that the node is listening on
int asServerPort = 3000;
// Establish connection
client = new AerospikeClient(asServerIP, asServerPort);
```

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

2. At the end of Main() method, 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.

```
if (client != null && client.Connected)
{
      // Close Aerospike server connection
      client.Close();
}
```



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

1. Create a WritePolicy

```
// Write record
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

2. Create Key and Bin instances

```
Key key = new Key("test", "users", username);
Bin bin1 = new Bin("username", username);
Bin bin2 = new Bin("password", password);
Bin bin3 = new Bin("gender", gender);
Bin bin4 = new Bin("region", region);
Bin bin5 = new Bin("lasttweeted", 0);
Bin bin6 = new Bin("tweetcount", 0);
Bin bin7 = Bin.AsList("interests",
           interests.Split(',').ToList<object>());
```

3. Write a user record using the Key and Bins

```
client.Put(wPolicy, key, bin1, bin2, bin3,
                         bin4, bin5, bin6, bin7);
```

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

1. Read User record

```
userRecord = client.Get(null, new Key("test", "tweets", username));
```

2. Create a WritePolicy

```
WritePolicy wPolicy = new WritePolicy();
wPolicy.recordExistsAction = RecordExistsAction.UPDATE;
```

3. Create Key and Bin instances

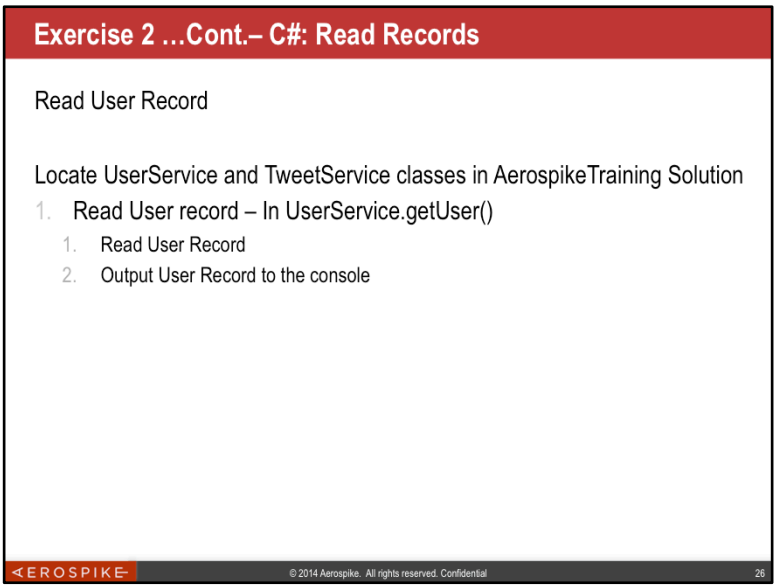
```
long ts = getTimeStamp();
tweetKey = new Key("test", "tweets", username + ":"
                                     + nextTweetCount);
Bin bin1 = new Bin("tweet", tweet);
Bin bin2 = new Bin("ts", ts);
Bin bin3 = new Bin("username", username);
```

4. Write a tweet record using the Key and Bins

```
client.Put(wPolicy, tweetKey, bin1, bin2, bin3);
Console.WriteLine("\nINFO: Tweet record created!");
```

5. Update the user record with tweet count

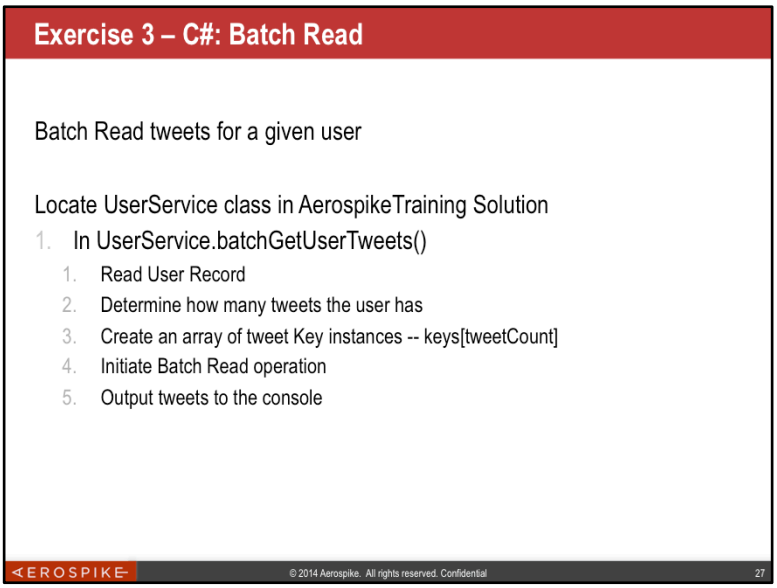
```
// Update tweet count and last tweet'd timestamp
// in the user record
updateUser(client, userKey, wPolicy, ts, nextTweetCount);
```



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

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
if (userRecord != null)
{
    Console.WriteLine("\nINFO: User record read successfully! Here are the details:\n");
    Console.WriteLine("username:     " + userRecord.GetValue("username"));
    Console.WriteLine("password:     " + userRecord.GetValue("password"));
    Console.WriteLine("gender:       " + userRecord.GetValue("gender"));
    Console.WriteLine("region:       " + userRecord.GetValue("region"));
    Console.WriteLine("tweetcount:   " + userRecord.GetValue("tweetcount"));
    List<object> interests = (List<object>) userRecord.GetValue("interests");
    Console.WriteLine("interests:    " + interests.Aggregate((x, y) => x + "," + y));
}
else
{
    Console.WriteLine("ERROR: User record not found!");
}
```



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

1. Read a user record

```
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
```

2. Get the tweet count

```
// Get how many tweets the user has
int tweetCount = int.Parse(userRecord.GetValue("tweetcount").ToString());
```

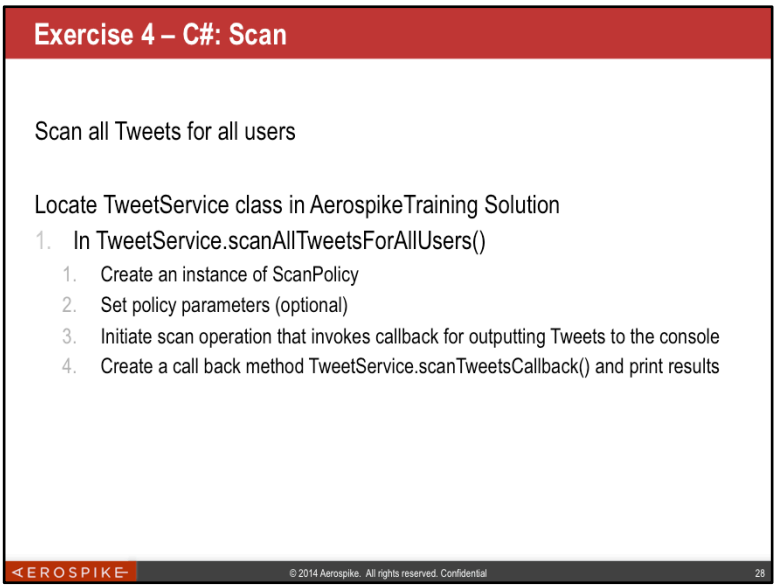
3. Create a "list" of tweet keys

```
// Create an array of keys so we can initiate batch read operation
Key[] keys = new Key[tweetCount];
for (int i = 0; i < keys.Length; i++)
{
    keys[i] = new Key("test", "tweets", (username + ":" + (i + 1)));
}
Console.WriteLine("\nHere's " + username + "'s tweet(s):\n");
```

4. Perform a Batch operation to read all the tweets

```
// Initiate batch read operation
Record[] records = client.Get(null, keys);
for (int j = 0; j < records.Length; j++)
{
    Console.WriteLine(records[j].GetValue("tweet"));
}
```

5. Then print out the tweets



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

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

1. Create a ScanPolicy
2. Set policy parameters

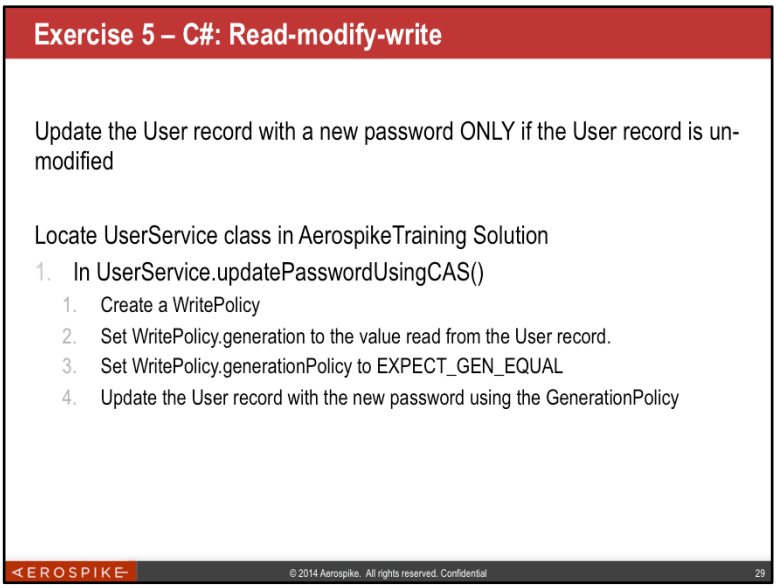
```
ScanPolicy policy = new ScanPolicy();
policy.concurrentNodes = true;
policy.priority = Priority.LOW;
policy.includeBinData = true;
```

3. Perform a Scan operation

```
client.ScanAll(policy, "test", "tweets", scanTweetsCallback, "tweet");
```

4. Create a call back method TweetService.scanTweetsCallback() and print the results

```
public void scanTweetsCallback(Key key, Record record)
{
     Console.WriteLine(record.GetValue("tweet"));
} //scanTweetsCallback
```



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 WritePolicy
2. Set WritePolicy.generation to the value read from the User record.
3. Set WritePolicy.generationPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the GenerationPolicy

```
// Check if username exists
userKey = new Key("test", "users", username);
userRecord = client.Get(null, userKey);
if (userRecord != null)
{
    // Get new password
    string password;
    Console.WriteLine("Enter new password for " + username + ":");
    password = Console.ReadLine();

    WritePolicy writePolicy = new WritePolicy();
    // record generation
    writePolicy.generation = userRecord.generation;
    writePolicy.generationPolicy = GenerationPolicy.EXPECT_GEN_EQUAL;
    // password Bin
    passwordBin = new Bin("password", password);
    client.Put(writePolicy, userKey, passwordBin);

    Console.WriteLine("\nINFO: The password has been set to: " + password);
}
```

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()

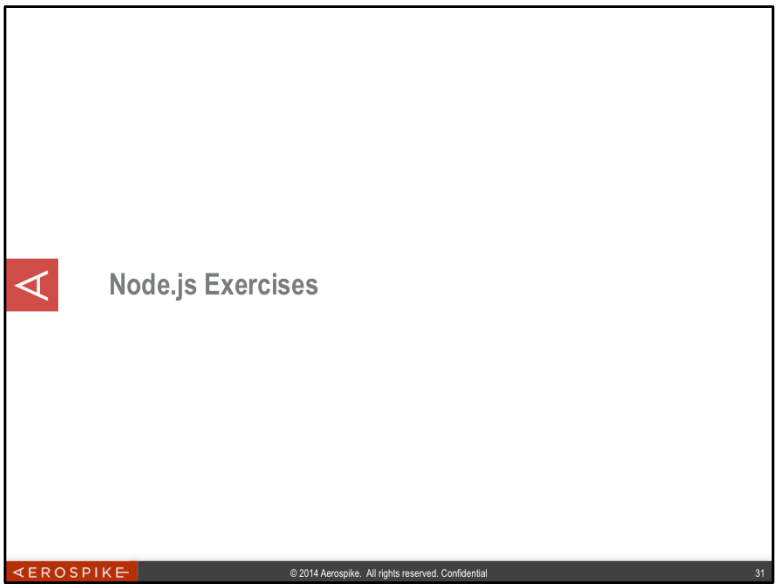
1. Comment out the code added in exercise 2

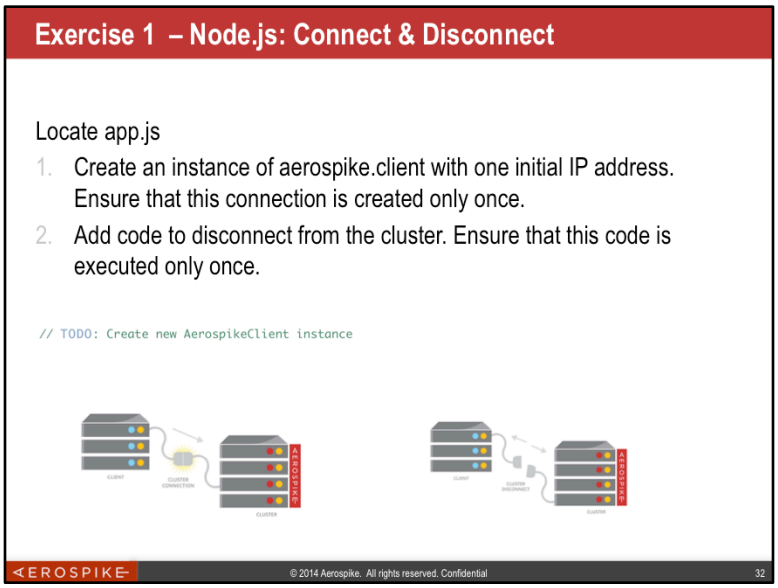
2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

3. In TweetService.updateUserUsingOperate() , add code similar to this:

```
Record record = client.Operate(policy, userKey,
     Operation.Add(new Bin("tweetcount", 1)),
     Operation.Put(new Bin("lasttweeted", ts)),
     Operation.Get());
Console.WriteLine("INFO: The tweet count now is: " + record.GetValue("tweetcount"));
```


Node.js Exercises
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
31



In this exercise you will connect to a Cluster by creating an Aerospike client instance, passing a single IP address and port to the constructor. The IP address and port 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 client is thread safe and creates a pool of worker threads, this means you DO NOT need to create your own connection or thread pool.

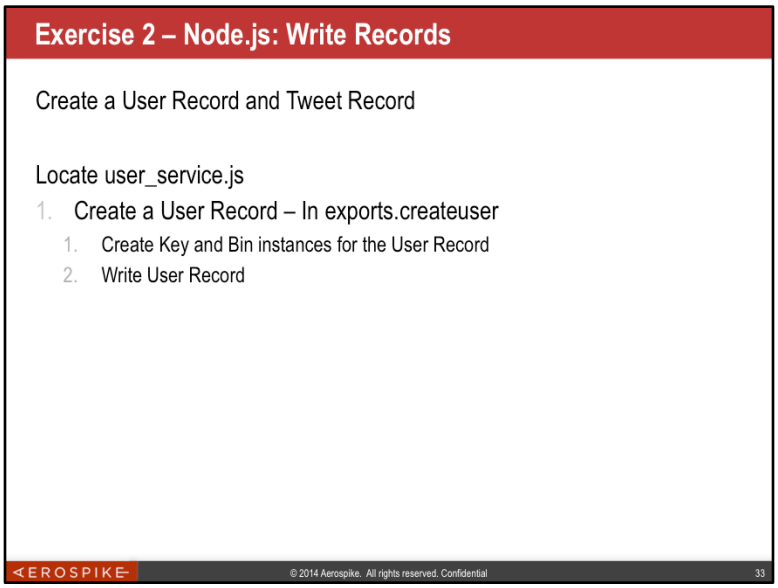
1. In app.js add code similar to this;

```
// Connect to the Aerospike Cluster
var client = aerospike.client({
    hosts: [ { addr: '172.16.159.172', port: 3000 } ]
}).connect(function(response) {
    // Check for errors
    if ( response.code == aerospike.status.AEROSPIKE_OK ) {
      // Connection succeeded
      console.log("Connection to the Aerospike cluster succeeded!");
    }
    else {
      // Connection failed
      console.log("Connection to the Aerospike cluster failed. Please check cluster IP and Port
settings and try again.");
      process.exit(0);
    }
});
```

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

2. Add a *process.on* exit function and call close() to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

```
// Setup tear down
process.on('exit', function() {
  if (client != null) {
    client.close();
    // console.log("Connection to Aerospike cluster closed!");
  }
});
```

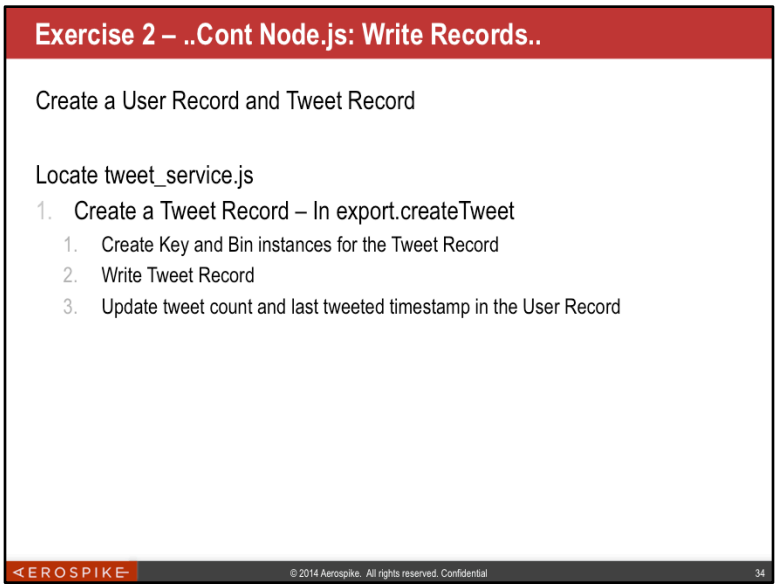


Create a User Record. In exports.createUser, add code similar to this:

1. Create Key and Bin instances

2. Write a user record using the Key and Bins

```
var key = {
  ns:  "test",
  set: "users",
  key: answers.username
};

var bins = {
  username: answers.username,
  password: answers.password,
  gender: answers.gender,
  region: answers.region,
  lasttweeted: 0,
  tweetcount: 0,
  interests: answers.interests.split(",")
};

client.put(key, bins, function(err, rec, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User record created!");

    // Create tweet record
    tweet_service.createTweet(client);
  }
  else {
    console.log("ERROR: User record not created!");
    console.log(err);
  }
});
```



Create a Tweet Record. In exports.createTweet add code similar to this:

1. Create Key and Bin instances

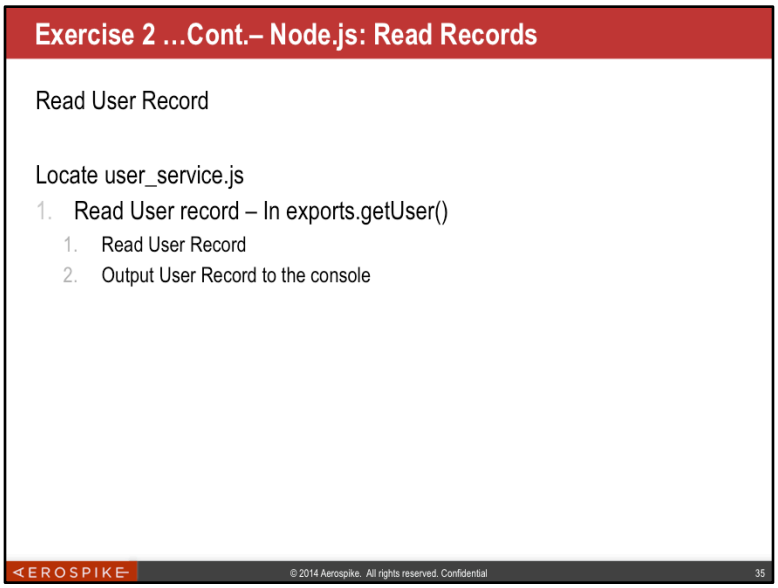
```
// Write Tweet record
var tweet_key = {
  ns:  "test",
  set: "tweets",
  key: userrecord.username + ":" + tweet_count
};

var bins = {
  username: userrecord.username,
  tweet: answer.tweet,
  ts: ts
};
```

2. Write a tweet record using the Key and Bins

```
client.put(tweet_key, bins, function(err, tweetrecord, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: Tweet record created!");
```

3. Update the user record with tweet count

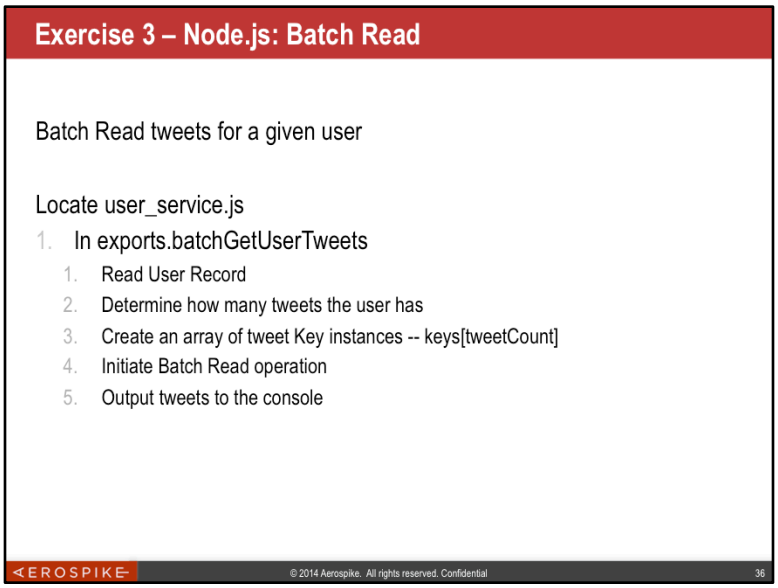
```
    // Update tweetcount and last tweet'd timestamp in the user record
    updateUser(client, user_key, ts, tweet_count);
  }
  else {
    console.log("ERROR: Tweet record not created!");
    console.log("",err);
  }
```



Read a User Record. In exports.getUser, add code, similar to this, to:

1. Read a User record
2. Output the User record to the console

```
// Read User record
var key = {
  ns:  "test",
  set: "users",
  key: answer.username
};

client.get(key, function(err, rec, meta) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User record read successfully! Here are the details:");
    console.log("username:   " + rec.username);
    console.log("password:   " + rec.password);
    console.log("gender:     " + rec.gender);
    console.log("region:     " + rec.region);
    console.log("tweetcount: " + rec.tweetcount);
    console.log("lasttweeted: " + rec.lasttweeted);
    console.log("interests:  " + rec.interests);
  }
  else {
    console.log("ERROR: User record not found!");
  }
});
```



Read all the tweets for a given user. In the function exports.batchGetUserTweets, add code similar to this:

1. Read a user record

```
// Read User record
var key = {
  ns:  "test",
  set: "users",
  key: answer.username
};

client.get(key, function(err, userrecord, meta) {
  // Check for errors
  if ( err.code === 0 ) {
```

2. Get the tweet count

```
    var tweet_count = userrecord.tweetcount;
    var tweet_keys = [];
```

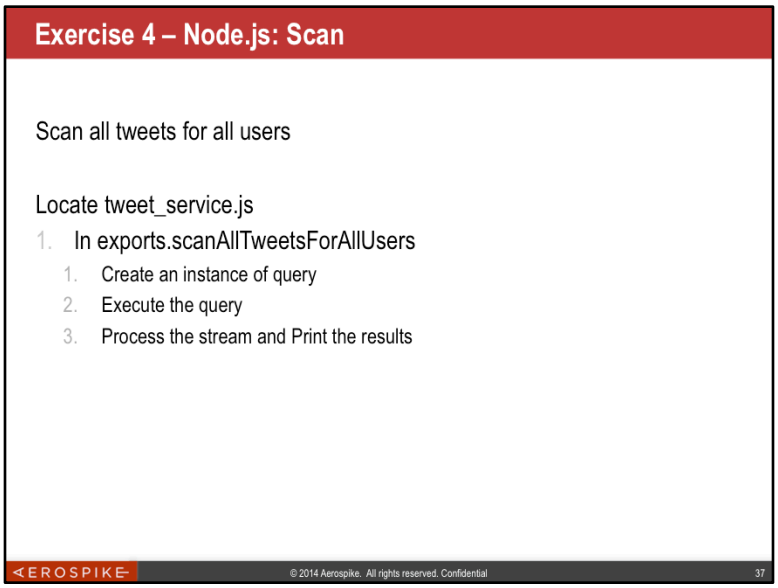
3. Create a “list” of tweet keys

```
    for(var i=1;i<=tweet_count;i++)  {
      tweet_keys.push({ns: "test", set: "tweets", key: answer.username + "
    }
```

4. Perform a Batch operation to read all the tweets

5. Then print out the tweets

```
    client.batchGet(tweet_keys, function (err, results) {
      // Check for errors
      if ( err.code === 0 ) {
        for(var j=0;j<results.length;j++)  {
          console.log(results[j].record.tweet);
        }
      }
      else {
        console.log("ERROR: Batch Read Tweets For User failed\n", err);
      }
    });

  }
  else {
    console.log("ERROR: User record not found!");
  }
});

  });
```



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

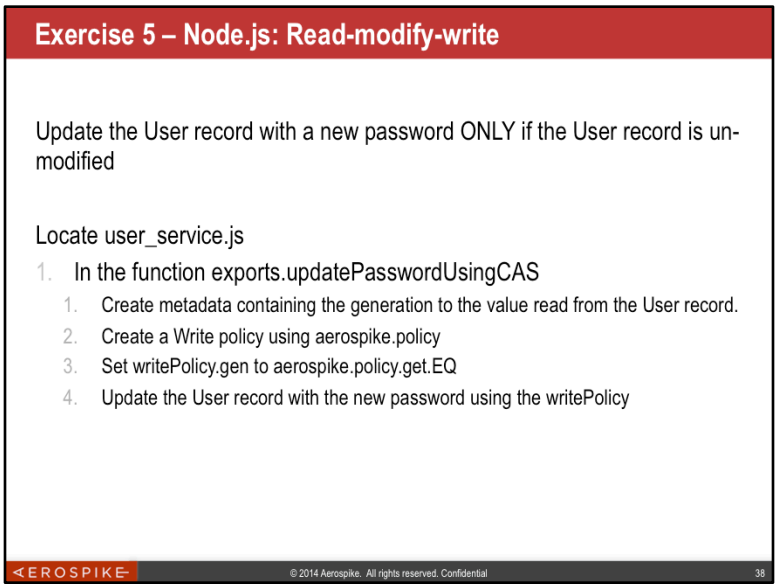
In the function exports.scanAllTweetsForAllUsers, add code similar to this:

1. Create a instance of a query
2. Execute the query

```
var query = client.query('test', 'tweets');
var stream = query.execute();
```

3. Process the stream and print the results

```
stream.on('data', function(record){
     console.log(record.bins.tweet);
});
stream.on('error', function(err){
     console.log('ERROR: Scan All Tweets For All Users failed: ',err);
});
stream.on('end', function(){
     // console.log('INFO: Scan All Tweets For All Users completed!');
});
```

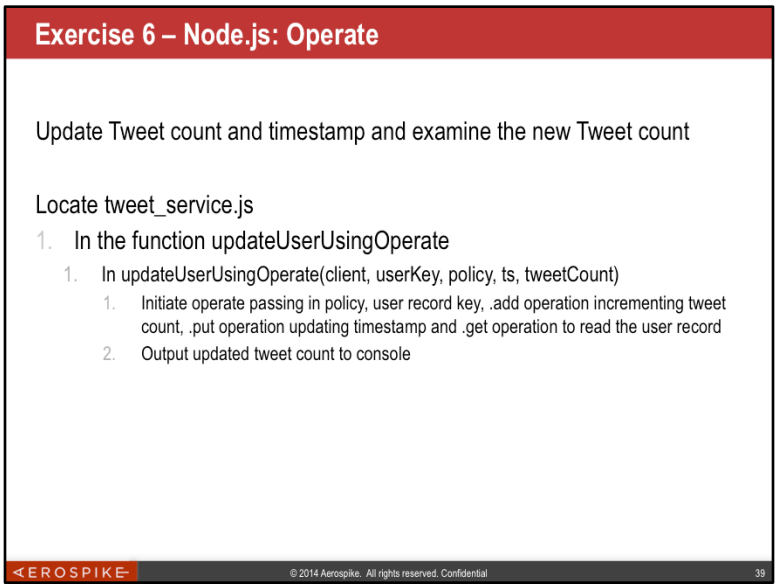


Update the User record with a new password ONLY if the User record is un-modified

In exports.updatePasswordUsingCAS, add code similar to this:

1. Create metadata containing the generation to the value read from the User record.
2. Create a Write policy using aerospike.policy
3. Set writePolicy.gen to aerospike.policy.get.EQ
4. Update the User record with the new password using the writePolicy

```
// Set the generation count to the current one from the user record.
// Then, setting writePolicy.gen to aerospike.policy.gen.EQ will ensure we don't have a 'dirty-
read' when updating user's password
var metadata = {
  gen: meta.gen
}

var writePolicy = aerospike.policy;
writePolicy.key = aerospike.policy.key.SEND;
writePolicy.retry = aerospike.policy.retry.NONE;
writePolicy.exists = aerospike.policy.exists.IGNORE;
writePolicy.commitLevel = aerospike.policy.commitLevel.ALL;
// Setting writePolicy.gen to aerospike.policy.gen.EQ will ensure we don't have a 'dirty-read'
when updating user's password
writePolicy.gen = aerospike.policy.gen.EQ;

var bin = {
  password: answer2.password
};

client.put(key, bin, metadata, writePolicy, function(err, rec) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: User password updated successfully!");
  }
  else {
    console.log("ERROR: User password update failed:\n", err);
  }
});
```



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 updateUser

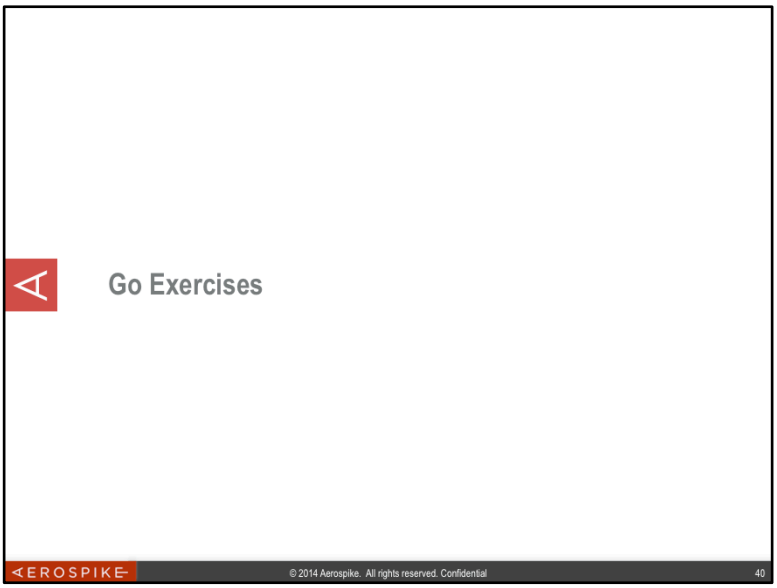
1. Comment out the code added in exercise 2

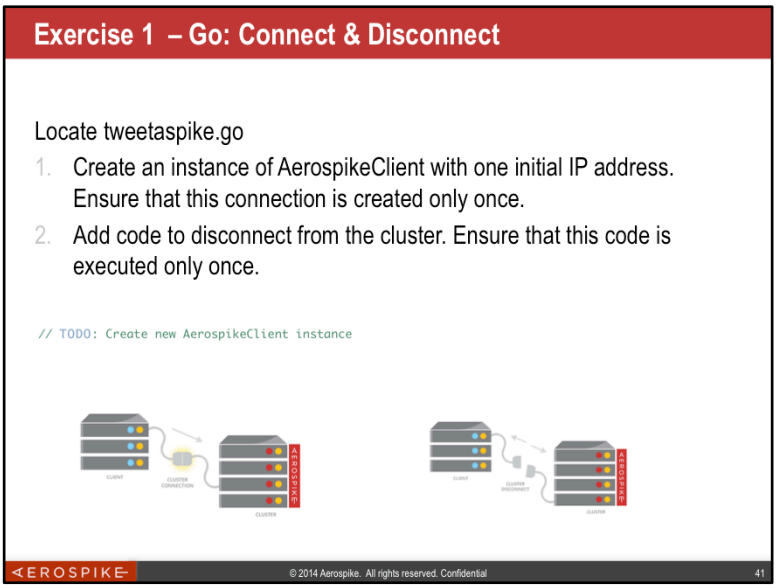
2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using operate
// Exercise 6
// console.log("TODO: Update tweet count and last tweeted timestamp in the user record using
operate");
// updateUserUsingOperate(client, user_key, ts);
```

3. In updateUserUsingOperate , add code similar to this:

```
// Update User record
var operator = aerospike.operator;
var operations = [operator.incr('tweetcount', 1),operator.write('lasttweeted',
ts),operator.read('tweetcount')];
client.operate(user_key, operations, function(err, bins, metadata, key) {
  // Check for errors
  if ( err.code === 0 ) {
    console.log("INFO: The tweet count now is: " + bins.tweetcount);
  }
  else {
    console.log("ERROR: User record not updated!");
    console.log(err);
  }
});
```


Go Exercises
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
40



In this exercise you will connect to a Cluster by creating an AerospikeClient instance, passing a single IP address and port to the constructor. The IP address and port should be to a node in your own cluster.

Ensure that you only create one client instance at the start of the program. The AerospikeClient 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 main() function add code similar to this;

```
fmt.Println("INFO: Connecting to Aerospike cluster...")
// Establish connection to Aerospike server
client, err := NewClient("54.90.203.181", 3000)
panicOnError(err)
```

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

2. Add a “defer” and call Close() to disconnect from the cluster. This should only be done once. After close() is called, the client instance cannot be used.

```
defer client.Close()
```

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

1. Create a WritePolicy

```
// Write record
wPolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
wPolicy.RecordExistsAction = UPDATE
```

2. Create Key and Bin instances

```
key, _ := NewKey("test", "users", username)
bin1 := NewBin("username", username)
bin2 := NewBin("password", password)
bin3 := NewBin("gender", gender)
bin4 := NewBin("region", region)
bin5 := NewBin("lasttweeted", 0)
bin6 := NewBin("tweetcount", 0)
arr := strings.Split(interests, ",")
bin7 := NewBin("interests", arr)
```

3. Write a user record using the Key and Bins

```
err := client.PutBins(wPolicy, key, bin1, bin2, bin3,
                              bin4, bin5, bin6, bin7)
panicOnError(err)
```

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

1. Create a WritePolicy

```
// Write record
wPolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
wPolicy.RecordExistsAction = UPDATE
// Create timestamp to store along with the tweet so we can
// query, index and report on it
timestamp := getTimeStamp()
```

2. Create Key and Bin instances

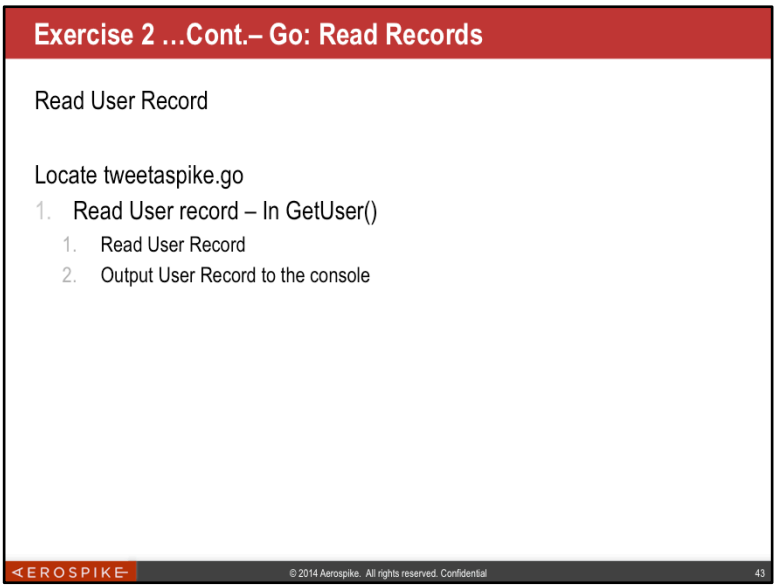
```
keyString := fmt.Sprintf("%s:%d", username, tweetCount)
tweetKey, _ := NewKey("test", "tweets", keyString)
bin1 := NewBin("tweet", tweet)
bin2 := NewBin("ts", timestamp)
bin3 := NewBin("username", username)
```

3. Write a tweet record using the Key and Bins

```
err := client.PutBins(wPolicy, tweetKey, bin1, bin2, bin3)
panicOnError(err)
fmt.Printf("\nINFO: Tweet record created! with key: %s, %v, %v, %v\n",
                        keyString, bin1, bin2, bin3)
```

4. Update the user record with tweet count

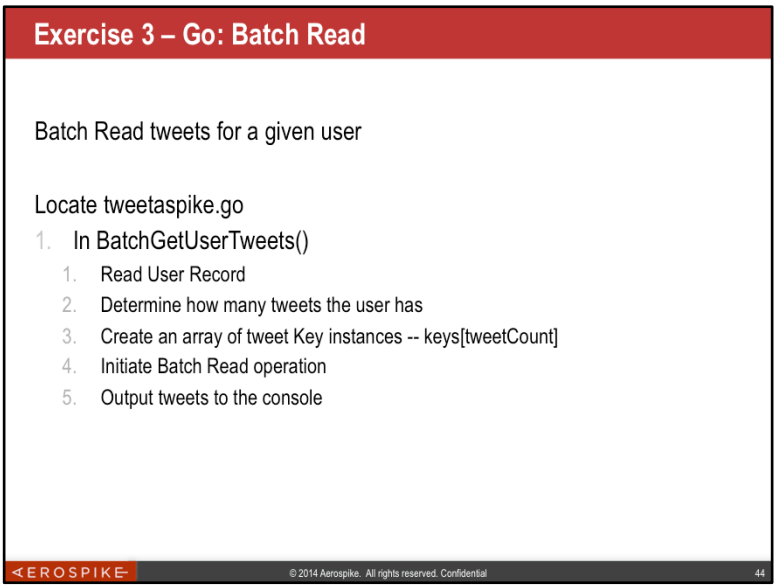
```
// Update tweet count and last tweet'd timestamp in the user record
updateUser(client, userKey, nil, timestamp, tweetCount)
```



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

1. Read a User record
2. Output the User record to the console

```
// Check if username exists
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)
if userRecord != nil {
      fmt.Printf("\nINFO: User record read successfully! Here are the details:\n")
      fmt.Printf("username:   %s\n", userRecord.Bins["username"].(string))
      fmt.Printf("password:   %s\n", userRecord.Bins["password"].(string))
      fmt.Printf("gender:     %s\n", userRecord.Bins["gender"].(string))
      fmt.Printf("region:     %s\n", userRecord.Bins["region"].(string))
      fmt.Printf("tweetcount: %d\n", userRecord.Bins["tweetcount"].(int))
      fmt.Printf("interests:  %v\n", userRecord.Bins["interests"])
} else {
      fmt.Printf("ERROR: User record not found!\n")
}
```



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

1. Read a user record

```
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)

if userRecord != nil {
```

2. Get the tweet count

```
    // Get how many tweets the user has
    tweetCount := userRecord.Bins["tweetcount"].(int)

    // Create an array of keys so we can initiate batch read
    // operation
    keys := make([]*Key, tweetCount)
```

3. Create a “list” of tweet keys

```
    for i := 0; i < len(keys); i++ {
        keyString, _ := fmt.Scanf("%s:%d", username, i+1)
        key, _ := NewKey("test", "tweets", keyString)
        keys[i] = key
    }

    fmt.Printf("\nHere's %s's tweet(s):\n", username)
```

4. Perform a Batch operation to read all the tweets

```
    // Initiate batch read operation
    if len(keys) > 0 {
        records, err := client.BatchGet(NewPolicy(), keys)
        panicOnError(err)
```

5. Then print out the tweets

```
        for _, element := range records {
            fmt.Println(element.Bins["tweet"])
        }
    }
}
```

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

In the ScanAllTweetsForAllUsers(), add code similar to this:

1. Create a ScanPolicy
2. Set policy parameters

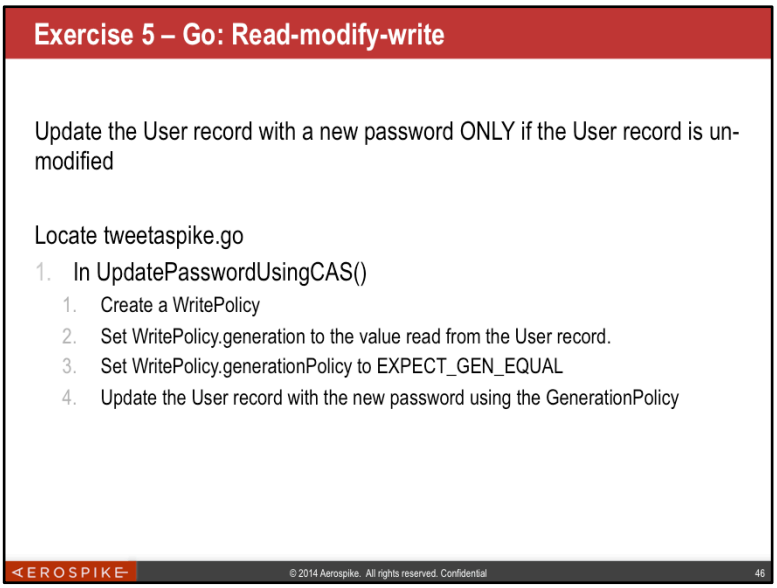
```
policy := NewScanPolicy()
policy.ConcurrentNodes = true
policy.Priority = LOW
policy.IncludeBinData = true
```

3. Perform a Scan operation

```
records, err := client.ScanAll(policy, "test", "tweets", "tweet")
panicOnError(err)
```

4. Print the results

```
for element := range records.Records {
     fmt.Println(element.Bins["tweet"])
}
```

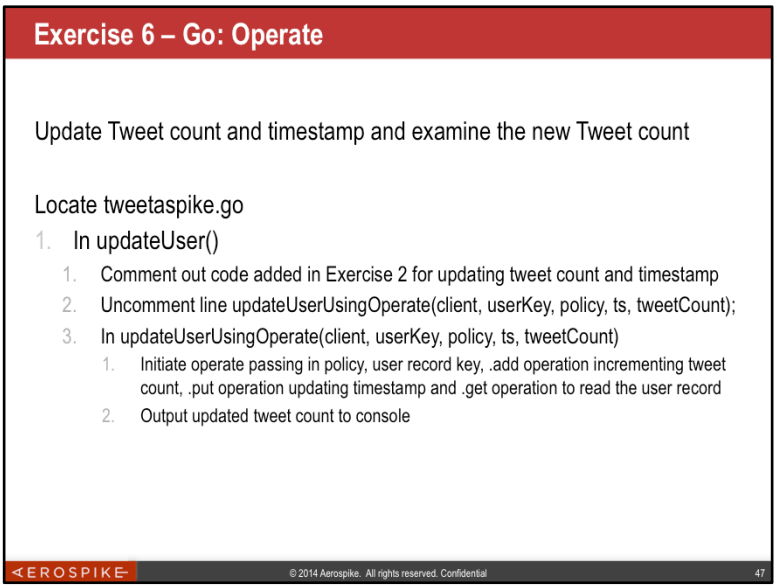


Update the User record with a new password ONLY if the User record is un-modified

In UpdatePasswordUsingCAS(), add code similar to this:

1. Create a WritePolicy
2. Set WritePolicy.generation to the value read from the User record.
3. Set WritePolicy.generationPolicy to EXPECT_GEN_EQUAL
4. Update the User record with the new password using the GenerationPolicy

```
// Check if username exists
userKey, _ := NewKey("test", "users", username)
userRecord, err := client.Get(nil, userKey)
panicOnError(err)
if err == nil {
      // Get new password
      var password string
      fmt.Print("Enter new password for %s:", username)
      fmt.Scanf("%s", &password)

      writePolicy := NewWritePolicy(0, 0) // generation = 0, expiration = 0
      // record generation
      writePolicy.Generation = userRecord.Generation
      writePolicy.GenerationPolicy = EXPECT_GEN_EQUAL
      // password Bin
      passwordBin := NewBin("password", password)
      err = client.PutBins(writePolicy, userKey, passwordBin)
      panicOnError(err)
      fmt.Printf("\nINFO: The password has been set to: %s", password)
} else {
      fmt.Printf("ERROR: User record not found!")
}
```



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 updateUser()

1. Comment out the code added in exercise 2

2. Uncomment the line:

```
// TODO: Update tweet count and last tweeted timestamp in the user record using Operate
// Exercise 6
// updateUserUsingOperate(client, userKey, policy, ts);
```

3. In updateUserUsingOperate() , add code similar to this:

```
record, err := client.Operate(policy, userKey,
     AddOp(NewBin("tweetcount", 1)),
     PutOp(NewBin("lasttweeted", timestamp)),
     GetOp())
panicOnError(err)

fmt.Printf("\nINFO: The tweet count now is: %d\n",
                      record.Bins["tweetcount"])
```

Summary

You have learned how to:

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

Summary

You have learned how to:

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


AEROSPIKE
AEROSPIKE
© 2014 Aerospike. All rights reserved. Confidential
50