Developer - Associate

lab

lab title

Programming AWS ElastiCache Redis using NodeJS V1.00

Course title

AWS Certified Developer Associate

Table of Contents

Contents

About the Lab

These lab notes are to support the instructional videos on Programming AWS ElastiCache Redis using NodeJS in the BackSpace AWS Certified Developer course.

In this lab we will:

- Create an ElastiCache Redis cluster using the console.
- Connect to an ElastiCache Redis cluster using the AWS NodeJS SDK.
- Read and Write to an ElastiCache Redis cluster using the AWS NodeJS SDK.

Please refer to the AWS JavaScript SDK documentation at:

http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/ElastiCache.html

Please refer to the Redis command documentation at:

http://redis.io/commands

Please refer to the NPM Redis documentation at:

https://github.com/NodeRedis/node_redis

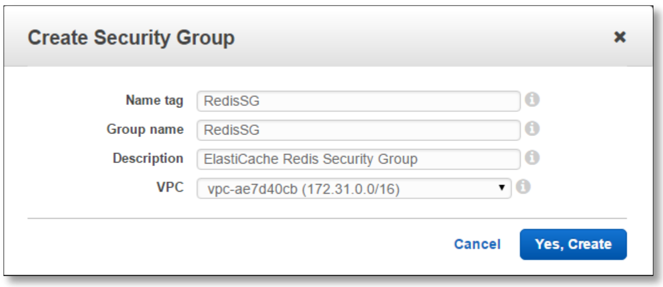
Please note that AWS services change on a weekly basis and it is extremely important you check the version number on this document to ensure you have the lastest version with any updates or corrections.

Launch an ElastiCache Redis Cluster

In this section we will create an ElastiCache Redis cluster using the console.

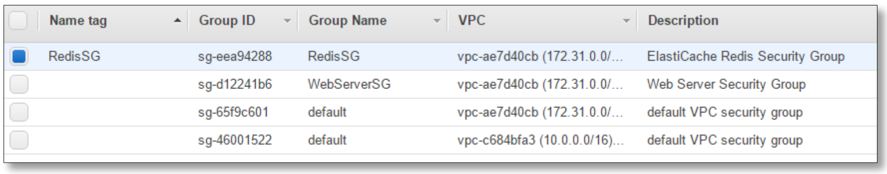
Go to the VPC console

Create a new security group in the default VPC and call it RedisSG



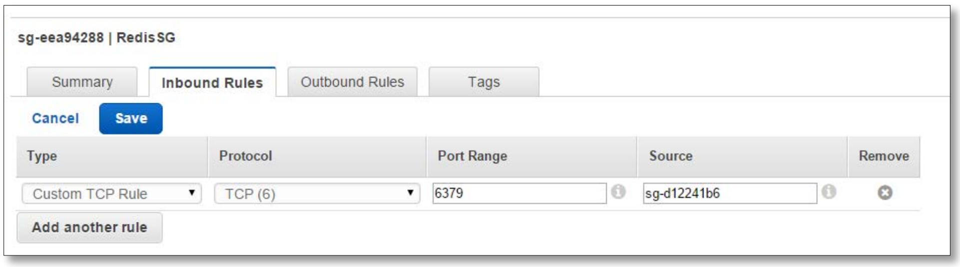
Create Security Group

Name tag	RedisSG
Group name	RedisSG
Description	ElastiCache Redis Security Group
VPC	vpc-ae7d40cb (172.31.0.0/16)



	Name tag	Group ID	Group Name	VPC	Description
■	RedisSG	sg-eea94288	RedisSG	vpc-ae7d40cb (172.31.0.0/...	ElastiCache Redis Security Group
☐		sg-d12241b6	WebServerSG	vpc-ae7d40cb (172.31.0.0/...	Web Server Security Group
☐		sg-65f9c601	default	vpc-ae7d40cb (172.31.0.0/...	default VPC security group
☐		sg-46001522	default	vpc-c684bfa3 (10.0.0.0/16)...	default VPC security group

Create a custom TCP rule for the ElastiCache Redis port 6379 and the WebServerSG security group.

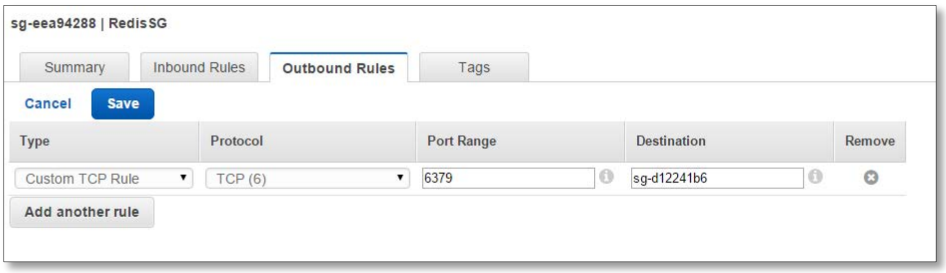


sg-eea94288 | RedisSG

Type	Protocol	Port Range	Source	Remove
Custom TCP Rule	TCP (6)	6379	sg-d12241b6	⊗

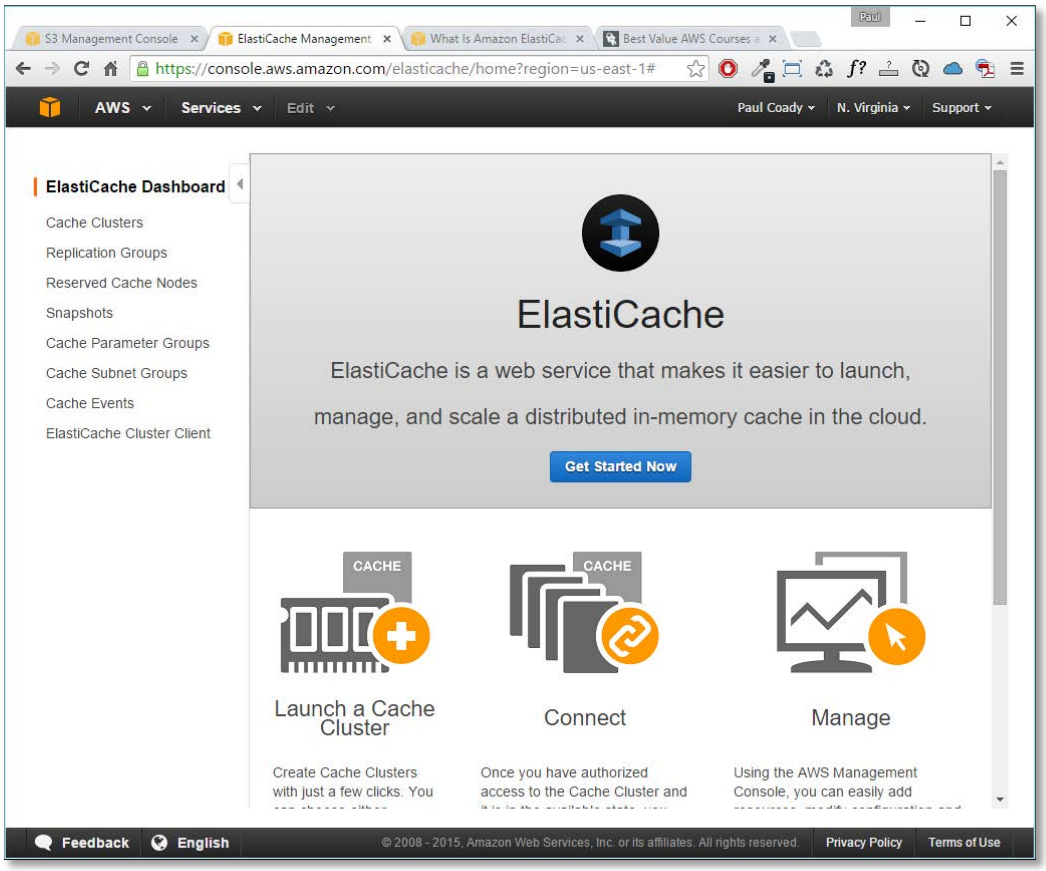
Click Save

Create a custom TCP outbound rule for the ElastiCache Redis port 6379 and the WebServerSG security group.


sg-eea94288 | Redis SG

Type	Protocol	Port Range	Destination	Remove
Custom TCP Rule	TCP (6)	6379	sg-d12241b6	

Go to the ElastiCache console.


https://console.aws.amazon.com/elasticache/home?region=us-east-1#

ElastiCache

ElastiCache is a web service that makes it easier to launch, manage, and scale a distributed in-memory cache in the cloud.

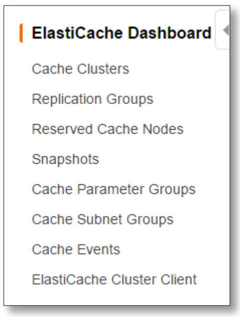
Create Cache Clusters with just a few clicks. You

Once you have authorized access to the Cache Cluster and

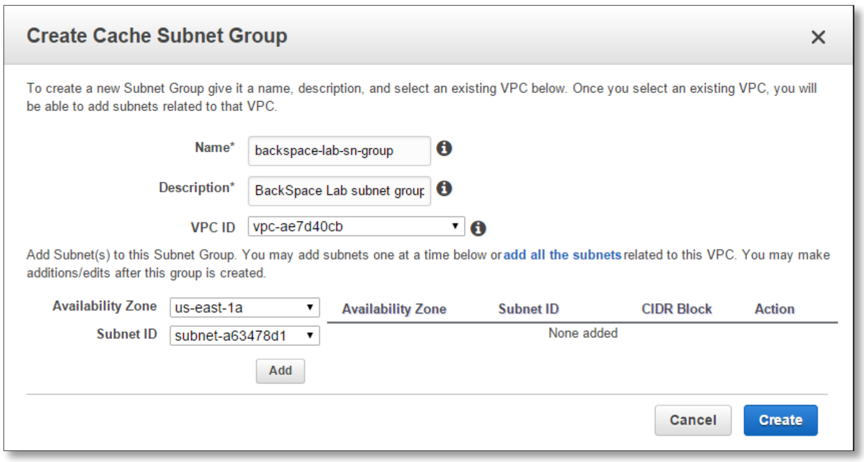
Using the AWS Management Console, you can easily add

© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Click on Cache Subnet Groups



Click Create Cache Subnet Group
Give it a name
Select the default VPC and an AZ and subnet.
Click Add
Click Create



Create Cache Subnet Group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Name*	backspace-lab-sn-group
Description*	BackSpace Lab subnet group
VPC ID	vpc-ae7d40cb

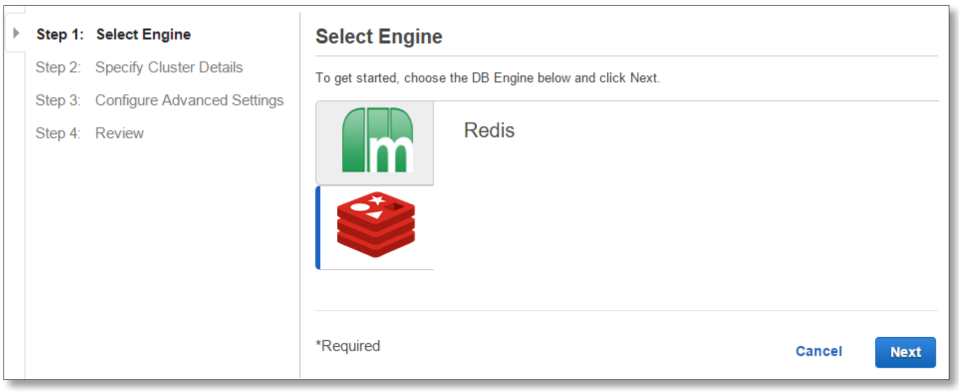
Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created.

Availability Zone	us-east-1a
Subnet ID	subnet-a63478d1

Availability Zone	Subnet ID	CIDR Block	Action
	None added		

Click ElastiCache Dashbboard
Click “Get Started Now”

Select Redis


Step 1: Select Engine
Step 2: Specify Cluster Details
Step 3: Configure Advanced Settings
Step 4: Review

Select Engine

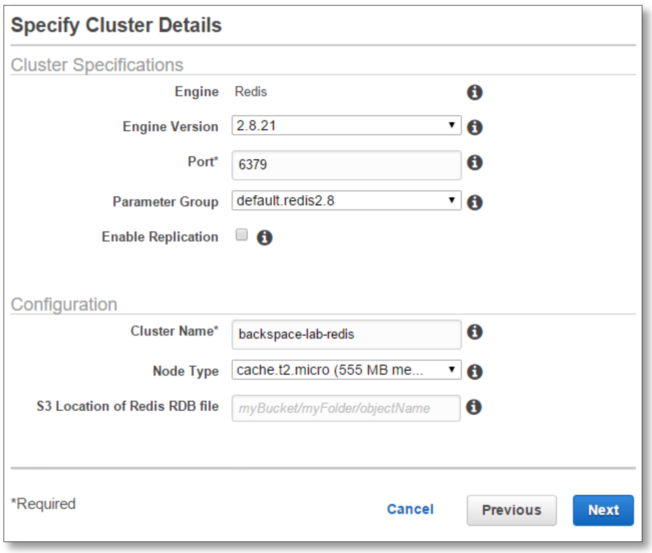
To get started, choose the DB Engine below and click Next.

Redis

*Required

Click Next

Call the cluster backspace-lab-redis
Uncheck Enable replication for the lab
Select the t2 micro node type



Specify Cluster Details

Cluster Specifications

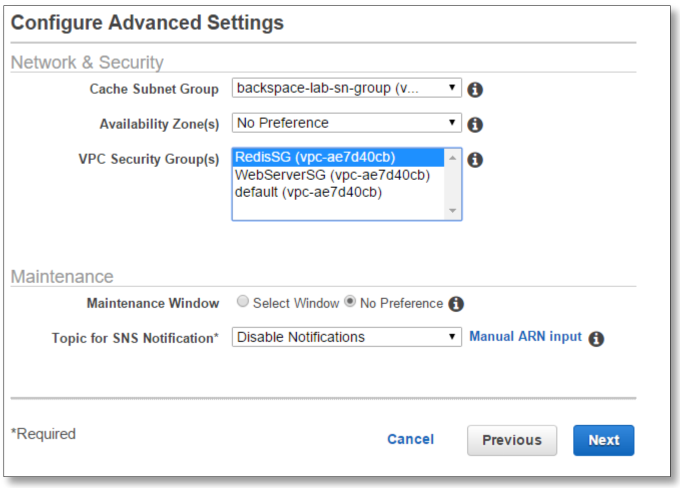
Engine	Redis
Engine Version	2.8.21
Port*	6379
Parameter Group	default.redis2.8
Enable Replication	☐

Configuration

Cluster Name*	backspace-lab-redis
Node Type	cache.t2.micro (555 MB me...
S3 Location of Redis RDB file	myBucket/myFolder/objectName

*Required

Click Next
Select your Subnet Group created previously
Select default VPC
Select your Security Group created previously



Configure Advanced Settings

Network & Security

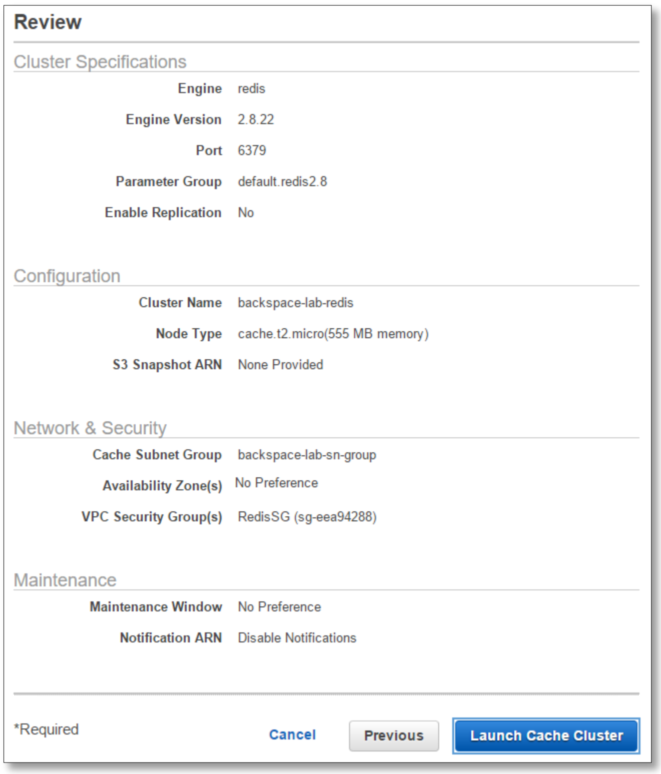
Cache Subnet Group	backspace-lab-sn-group (v...
Availability Zone(s)	No Preference
VPC Security Group(s)	RedisSG (vpc-ae7d40cb) [selected] WebServerSG (vpc-ae7d40cb) default (vpc-ae7d40cb)

Maintenance

Maintenance Window	Select Window [● No Preference]
Topic for SNS Notification*	Disable Notifications — Manual ARN input

*Required

Click Next



Review

Cluster Specifications

Engine	redis
Engine Version	2.8.22
Port	6379
Parameter Group	default.redis2.8
Enable Replication	No

Configuration

Cluster Name	backspace-lab-redis
Node Type	cache.t2.micro(555 MB memory)
S3 Snapshot ARN	None Provided

Network & Security

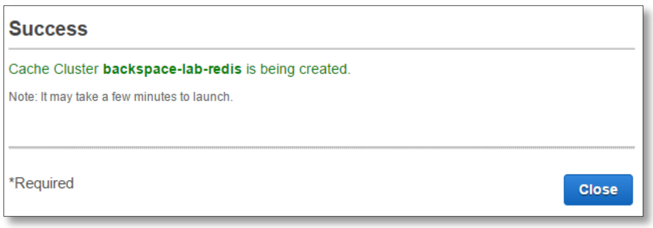
Cache Subnet Group	backspace-lab-sn-group
Availability Zone(s)	No Preference
VPC Security Group(s)	RedisSG (sg-eea94288)

Maintenance

Maintenance Window	No Preference
Notification ARN	Disable Notifications

*Required

Launch Cache Cluster



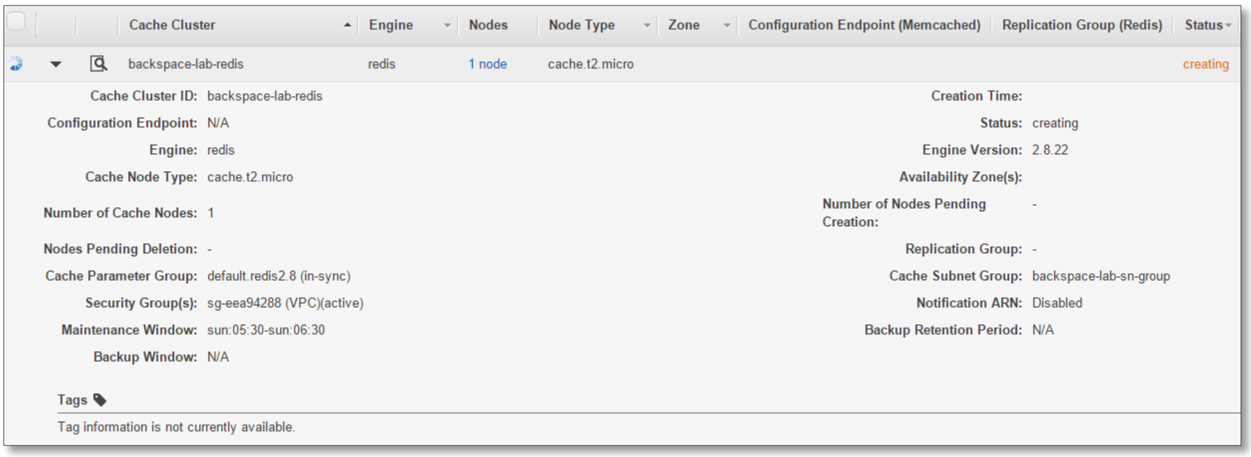
Success

Cache Cluster **backspace-lab-redis** is being created.

Note: It may take a few minutes to launch.

*Required

Click Close



Cache Cluster	Engine	Nodes	Node Type	Zone	Configuration Endpoint (Memcached)	Replication Group (Redis)	Status
backspace-lab-redis	redis	1 node	cache.t2.micro				creating

Cache Cluster ID:	backspace-lab-redis	Creation Time:	
Configuration Endpoint:	N/A	Status:	creating
Engine:	redis	Engine Version:	2.8.22
Cache Node Type:	cache.t2.micro	Availability Zone(s):	
Number of Cache Nodes:	1	Number of Nodes Pending Creation:	-
Nodes Pending Deletion:	-	Replication Group:	-
Cache Parameter Group:	default.redis2.8 (in-sync)	Cache Subnet Group:	backspace-lab-sn-group
Security Group(s):	sg-eea94288 (VPC)(active)	Notification ARN:	Disabled
Maintenance Window:	sun:05:30-sun:06:30	Backup Retention Period:	N/A
Backup Window:	N/A		

Tags

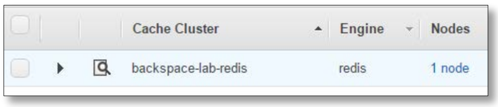
Tag information is not currently available.

Connect to an ElastiCache Redis Cluster using NodeJS

In this section we will connect to an ElastiCache Redis cluster using NodeJS.

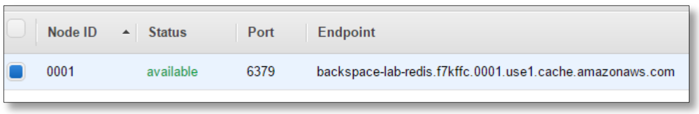
From the console go to Cache Clusters

Click on the Cluster Node in your Cache Cluster



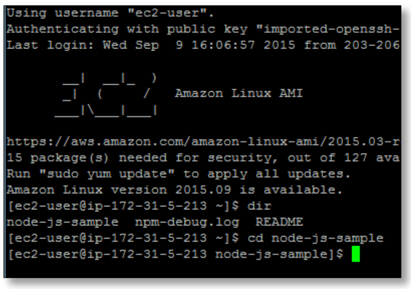
			Cache Cluster	Engine	Nodes
			backspace-lab-redis	redis	1 node

Copy the endpoint and the port, we will need this to connect to the node.



	Node ID	Status	Port	Endpoint
	0001	available	6379	backspace-lab-redis.f7kffc.0001.use1.cache.amazonaws.com

Open Putty and CD into your sample application



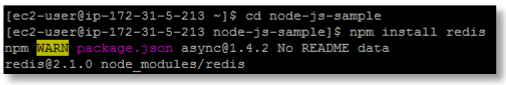
```
Using username "ec2-user".
Authenticating with public key "imported-openssh-
Last login: Wed Sep  9 16:06:57 2015 from 203-206

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.03-r
15 package(s) needed for security, out of 127 ava
Run "sudo yum update" to apply all updates.
Amazon Linux version 2015.09 is available.
[ec2-user@ip-172-31-5-213 ~]$ dir
node-js-sample  npm-debug.log  README
[ec2-user@ip-172-31-5-213 ~]$ cd node-js-sample
[ec2-user@ip-172-31-5-213 node-js-sample]$
```

Install the node Redis package

npm install redis



```
[ec2-user@ip-172-31-5-213 ~]$ cd node-js-sample
[ec2-user@ip-172-31-5-213 node-js-sample]$ npm install redis
npm WARN package.json async@1.4.2 No README data
redis@2.1.0 node_modules/redis
```

Open Atom IDE and Remote Edit into Index.js

Delete the existing application and replace with
*** Be sure to change for your endpoint and port if different.

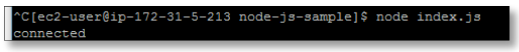
```
// Include the async package
// Make sure you add "async" to your package.json
var async = require('async');
// Include the redis package
// Be sure to npm install redis
var redis = require('redis');
var PORT = 6379;
var HOST = "YOUR_REDIS_ENDPOINT";
var client = redis.createClient(PORT, HOST); //creates a new Redis client

client.on('connect', function() {
    console.log('connected');
});
```

Click Ctrl + S to save to the EC2 instance.

Now run your application and it should connect to your Redis node.

It should now be connected to Redis



```
^C[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
```

Click Ctrl + C to stop application.

Using ElastiCache Redis with NodeJS

In this section we will read and write to an ElastiCache Redis cluster using NodeJS.

Open Atom IDE

Add a call to a function called writeRedisKey in the connect callback

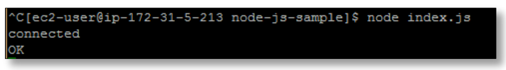
Create the new function which stores the high score for a game.

```
client.on('connect', function() {
    console.log('connected');
    writeRedisKey("myHighScore", "1000");
});

function writeRedisKey(keyRedis, value){
  client.set(keyRedis, value, function(err, response) {
    console.log(response);
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run your application and it should create and save the key to your Redis node.



```
^C[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
OK
```

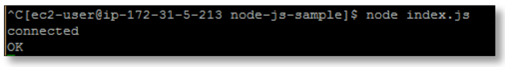
Click Ctrl + C to stop application.

Now set an expire time of 30 seconds for the key.

```
function writeRedisKey(keyRedis, value){
  client.set(keyRedis, value, function(err, response) {
    console.log(response);
    client.expire('keyRedis', 30);
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run your application and it should create and save the key with an expiry time to your Redis node.



```
^C[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
OK
```

Click Ctrl + C to stop application.

Add a call to a function called readRedisKey in the writeRedisKey callback

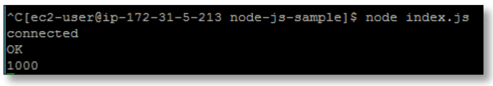
Create the new function which returns current the high score for a game.

```
function writeRedisKey(keyRedis, value){
  client.set(keyRedis, value, function(err, response) {
    console.log(response);
    client.expire(keyRedis, 30); // key expires in 30 s
    readRedisKey(keyRedis);
  });
}

function readRedisKey(keyRedis){
  client.get(keyRedis, function(err, response) {
      console.log(response);
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run your application.



```
^C[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
OK
1000
```

Click Ctrl + C to stop application.

Add a call to a function called writeRedisObject in the readRedisKey callback

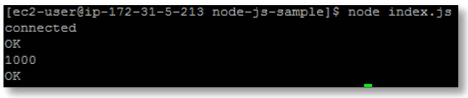
Create the new function which will create and save an object of keys.

```
function readRedisKey(keyRedis){
  client.get(keyRedis, function(err, response) {
      console.log(response);
      var objInfo ={
        info1: "This is info 1",
        info2: "This is info 2",
        info3: "This is info 3"
      };
      writeRedisObject("myInfo", objInfo);
  });
}

function writeRedisObject(objRedis, value){
  client.hmset(objRedis, value, function(err, response){
    console.log(response);
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run your application.



```
[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
OK
1000
OK
```

Click Ctrl + C to stop application.

Add a call to a function called readRedisObject in the writeRedisObject callback

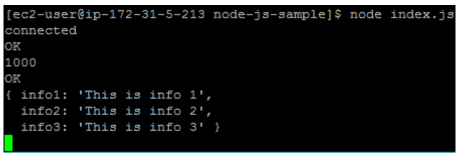
Create the new function which will read an object of keys.

```
function writeRedisObject(objRedis, value){
  client.hmset(objRedis, value, function(err, response){
    console.log(response);
    readRedisObject(objRedis);
  });
}

function readRedisObject(objRedis){
  client.hgetall(objRedis, function(err, response) {
      console.log(response);
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run your application.



```
[ec2-user@ip-172-31-5-213 node-js-sample]$ node index.js
connected
OK
1000
OK
{ info1: 'This is info 1',
  info2: 'This is info 2',
  info3: 'This is info 3' }
```

Click Ctrl + C to stop application.

Now clean up by deleting your cluster in the console.