AWS SETUP

1. What you need:
   a. An AWS account. Signup is at Amazon EC2 page
   b. Create a local directory aws: ~/aws or c:\aws.
      It is easier to have a sub directory to keep all aws stuff
   c. (optional) Have downloaded and installed aws tools. I install them in the aws sub directory
   d. Setup and store an SSH key. My key is at ~/aws/SSH-Key1.pem
   e. Create a security group (for example Test-1 as shown in Figure 1 – Open port 22,80
      and 8080. Remember this is for experimental purposes and not for production)

![Security Group Image]

2. Redeem AWS credit. Amazon has given us a credit voucher for $25. (Thanks to Jeff Barr and
Jenny Kohr Chynoweth)
   a. Visit the AWS Credit Redemption Page and enter the promotion code above. You
      must be signed in to view this page.
   b. Verify that the credit has been applied to your account by viewing your Account
      Activity page.
   c. These credits will be good for $25 worth of AWS services from June 10, 2010-December

3. I have created OSCON-NOSQL AMI that has the stuff needed for this tutorial.
   b. Reference Pack
      i. The NOSQL Slide set
      ii. The hand out (This document)
iii. Reference List
iv. The AMI
v. The NOSQL papers

4. To launch the AMI
   AWS Management Console-EC2-Launch Instance
   Search for OSCON-NOSQL (Figure 2)

   ![Figure 2]
   
   FIGURE 2

   - [Select]
   - [Instance Details] (small instance is fine)
   - [Advanced Instance Options] (Default is fine)
   - [Create Key pair] (Use existing if you already have one or create one and store in the ~/aws directory)
   - [Configure Firewall] – choose Test-1 or create 1 with 80,22 and 8080 open
   - [Review]-[Launch]-[Close]

   Instance should show up in the management console in a few minutes (Figure 3)

   ![Figure 3]
   
   FIGURE 3

5. Open up a console and ssh into the instance
   SSH-Key1.pem ubuntu@ec2-184-73-85-67.compute-1.amazonaws.com
   And type-in couple of commands (Figure 4)
Note:

If it says connection refused, reboot the instance. You can check the startup logs by typing:

```
ec2-get-console-output --private-key <private key file name> --cert <certificate file name> <instance id>
```

Also type-in commands like the following to get a feel for the instance:

```
ps -A
vmstat
cat /proc/cpuinfo
uname -a
sudo fdisk -l
sudo lshw
df -h -T
```

Note: Don’t forget to turn off the instance at the end of the hands-on session!
Close Encounters of the first kind …

- Start mongo shell: type `mongo`

```
$ mongo
MongoDB shell version: 1.5.5-pre-connection to: test
> show dbs
admin
local
locus
> use locus
switched to db locus
> db.stats()
{
  "collections" : 0,
  "objects" : 0,
  "avgObjSize" : NaN,
  "dataSize" : 0,
  "storageSize" : 0,
  "numExtents" : 0,
  "indexes" : 0,
  "indexSize" : 0,
  "fileSize" : 0,
  "ok" : 1
}
```

**FIGURE 5**

- The CLI is a full Javascript interpreter
- Type any javascript commands
  ```javascript
  function loopTest() {
    for (i=0; i<10; i++) {
      print(i);
    }
  }
  ```
  See how the shell understands editing, multi-line and end of function
  - loopTest will show the code
  - loopTest() will run the function
- Other commands to try
  - `show dbs`
  - `use <dbname>`
  - `show collections`
  - `db.help()`
  - `db.collections.help()`
  - `db.stats()`
  - `db.version()`

**Warm up**

- create a document & insert it
- person = {"name":"aname","phone":"555-1212"}
  - db.persons.insert(person)
  - db.persons.find()
- <discussion about id>
  - db.persons.help()
  - db.persons.count()
- Create 2 more records
- Embedded Nested documents
  - address = {"street":"123,some street","City":"Portland","state":"WA","tags":["home","WA"]}
  - db.persons.update({"name":"aname"},address) – will replace
- Need to use update modifier
  - Db.persons.update({"name":"aname"},{$set :{address}}) to make add address
- Discussion:
  - Embedded docs are very versatile
  - For example the collider could add more data from another source with nested docs and added metadata
  - As mongo understands json structure, it can index inside embedded docs. For example address.state to update keys in embedded docs
- db.persons.remove({})
- "$unset" to remove a key
- Upsert – interesting
  - db.persons.update({"name":"xyz"},{"$inc":{"checkins":1}}) < - won't add
  - db.persons.update({"name":"xyz"},{"$inc":{"checkins":1}},{"upsert":"true"}) < - will add
- Add a new field/update schema
  - db.persons.update({},{"set":{"insertDate":new Date()}})
  - db.runCommand({getLastError:1})
  - db.persons.update({},{"set":{"insertDate":new Date()}}),{"upsert":"false"},{"multi":"true"})
  - db.runCommand({getLastError:1})

**Main Feature – Locus**

- Goal is to develop location based/geospatial capability:
  - Allow users to 'check in' at places
  - Based on the location, recommend near-by attractions
  - Allow users to leave tips about places
  - Do a few aggregate map-reduce operations like 'How many people are checked in at a place?' and 'List who is there' ...
- But for this demo, we will implement a small slice : Ability to checkin and show nearby soccer attractions using a Java program
- A quick DataModel
  - Database : locus
  - Collections: Places – state, city, stadium, latlong
people – name, when, latlong

Hands on

ubuntudomU:12-31-39-00-40-D-72:~$ ls -l

```
total 228
drwxr-x-x 12 ubuntu ubuntu  4096 2010-07-14 00:17 cassandra
-rwxr-x-x  1 ubuntu ubuntu  4252 2010-07-14 00:14 CheckIn.java
-rwxr-x-x  1 ubuntu ubuntu 20119 2010-07-14 00:14 mongo-2.0.jar
-rwxr-x-x  1 ubuntu ubuntu  9063 2010-07-14 00:14 TestMongo1.java
ubuntudomU:12-31-39-00-40-D-72:~$ ls cassandra CheckIn.class CheckIn.java mongo-2.0.jar TestMongo1.class TestMongo1.java
ubuntudomU:12-31-39-00-40-D-72:~$ java -classpath ./mongo-2.0.jar:./CheckIn

Yep, got here!

locus

admin

db

local

ubuntudomU:12-31-39-00-40-D-72:~$ java -classpath ./mongo-2.0.jar:./CheckIn

Username,latitude,longitude for exit > K51, -23, 25

Excellent. K51 has checked in lat -23 long 25 at 2010/07/15 00:28:45

*** Query Object = ["latlong" : {"near" : ["0" : -23, "1" : 25]}]

--- Soccer Points Of Interest in the order of distance ---

Error: com.mongodb.MongoInternalException: couldn't get next element

[java.lang.StackTraceElement;@867e89

End Of FourSquareSoccer

ubuntudomU:12-31-39-00-40-D-72:~$ mongo

MongoDB shell version: 1.5.5-pre-

connecting to: test

> use locus

switched to db locus

> show collections

checks

places

system.indexes

> db.places.stats()

```
{
   "ns" : "locus.places",
   "count" : 40,
   "size" : 4332,
   "avgObjSize" : 108.3,
   "storageSize" : 6400,
   "numExtents" : 1,
   "nindexes" : 1,
   "lastExtentSize" : 6400,
   "paddingFactor" : 1,
   "flags" : 1,
   "totalIndexSize" : 8192,
   "indexSizes" : {
      "_id" : 8192
   },
   "ok" : 1
}
```

> 

FIGURE 6

FIGURE 7

- use <dbname> (use locus)
- Show collections
o db.stats()
o db.<collection>.find()
  o db.checkins.find()
o db.places.find()

Ubuntu@desktop-12-31-39-08-40-72:~$ mongo
MongoDB shell version: 1.5.5-pre-
connecting to: test
> show dbs
admin
local
locus
> use locus
switched to db locus
> show collections
checkins
places
system.indexes
> db.checkins.find()
{ "_id" : ObjectId("4e3e5613df7753b795a0d31"), "person" : "KS1", "when" : "2010/07/15 00:28:45", "latlong" : { "x" : -23, "y" : 25 } }
{ "_id" : ObjectId("4e3e56c75d553b78be8c95e"), "person" : "KS2", "when" : "2010/07/15 01:13:43", "latlong" : { "x" : -23, "y" : 25 } }
> db.places.find()
{ "_id" : ObjectId("4e3e5614600c53b3d3081c34"), "country" : "Afghanistan", "capital" : "Kabul", "latlong" : { "x" : 34.28, "y" : 69.11 } }
{ "_id" : ObjectId("4e3e5614600c53b3d3081c34"), "country" : "Algaria", "capital" : "Algiers", "latlong" : { "x" : 36.42, "y" : 3.88 } }
{ "_id" : ObjectId("4e3e5614600c53b3d3081c34"), "country" : "Germany", "capital" : "Berlin", "latlong" : { "x" : 52.3, "y" : 13.25 } }
{ "_id" : ObjectId("4e3e5614600c53b3d3081c34"), "country" : "Nigeria", "capital" : "Abuja", "latlong" : { "x" : 9.05, "y" : 7.32 } }

FIGURE 8

> db.places.ensureIndex({latlong:"2d"})
> exit
bye
Ubuntu@desktop-12-31-39-08-40-72:--$ java -classpath ./mongo-2.0.jar:/CheckIn
Username,latitude,longitude on 999 for exit > K52,-23,25
K52,-23,25

Excellent. KS2 has checked in lat -23 long 25 at 2010/07/15 01:13:43
*** Query Object = { "latlong" : { "$near" : [ { "0" : -23, "1" : 25 }] }}

--- Soccer Points Of Interest in the order of distance ---

Royal Bafokeng Stadium | Rustenburg | { "x" : 25.579, "y" : 27.161 } | 38646
Lofthus Versfeld Stadium | Pretoria | { "x" : 25.44, "y" : 28.12 } | 42588
Soccer City | Johannesburg | { "x" : 26.235, "y" : 27.982 } | 84490
Ellis Park Stadium | Johannesburg | { "x" : 26.197, "y" : 28.06 } | 55886
Peter Mokaba Stadium | Polokwane | { "x" : 23.924, "y" : 29.468 } | 41733
Free State Stadium | Bloemfontein | { "x" : 29.117, "y" : 26.209 } | 40911
Mbombela Stadium | Nelspruit | { "x" : 25.462, "y" : 30.929 } | 62760
Moses Mabhida Stadium | Durban | { "x" : 29.829, "y" : 31.03 } | 62760
Nelson Mandela Bay Stadium | Port Elizabeth | { "x" : 33.937, "y" : 25.599 } | 42486
Cape Town Stadium | Cape Town | { "x" : 33.983, "y" : 18.411 } | 64100
Nigeria | Abuja | { "x" : 9.85, "y" : 7.32 } | 1100
Argentinea | Buenos Aires | { "x" : -36.3, "y" : 60.0 } | 1000
China | Accra | { "x" : 5.35, "y" : -0.06 } | 1000

FIGURE 9
**db.places.find({latlong: {$near: [40,70]}})**

**Mapreduce**

Mapreduce is best for batch type operations, especially aggregations - while the standard queries are well suited for lookup operations/matching operations on documents. In our example, we could run mapreduce operation every few hours to calculate which locations are 'trending' i.e. have a many check ins in the last few hours.

First we will add one more element place and capture the place where the person checks in. Then we could use the following simple map-reduce operation which operates on the checkin collection which aggregates counts of all historical checkins per location:

```
m = function() {
  emit(this.place, 1);
}
r = function(key, values) {
  total = 0;
  for (var i in values) total += values[i];
  return total;
}
res = db.checkins.mapReduce(m,r)
db[res.result].find()
```

You could run this (this particular code should run in the javascript shell) and it would create a temporary collection with totals for checkins for each location, for all historical checkins.

If you wanted to just run this on a subset of recent data, you could just run the
mapreduce, but filter on checkins over the last 3 hours e.g.
res = db.checkins.mapReduce(m,r, {when: {$gte:nowminus3hours}}), where
nowminus3hours is a variable containing a datetime corresponding to the
current time minus 3 hours

This would give you a collection with documents for each location that was
checked into for the last 3 hours with the totals number of checkins for each
location contained in the document.

We could use this to develop a faceted search – popular places - 3 hrs, 1 day, 1
week, 1 month and max.

(Thanks to Nosh Petigara/10Gen for the scenario and the code)

The wind-up

*** First terminate the instance ! ***

Discussions

The Afterhours (NOSQL + Beer)

Homework – add social graph based inference mechanisms.
   a) MyNearby Friends
      When you check-in show if your friends are here (either add friends to this site or based
         on their facebook page using the OpenGraph API)
   b) We were here
      Show if their friends were here or nearby places the friends visited
   c) Show pictures taken by friends when they were here