# MrJob, Unsupervised Learning at Scale: Clustering, Canopy-based Kmeans, Expectation Maximization


**James G. Shanahan[2]**

**Assistants: Liang Dai[1, 3]**

*[1]NativeX, [2]iSchool UC Berkeley, CA, [3]UC Santa Cruz*



*EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com*


**Large Scale Machine Learning**

**MIDS, UC Berkeley**

**Live Session #4**

**February 2, 2016**

# Important Links for Week 4

- **Slides for Lives Session #4**
- **Instructions for Peer grading of homework HW4**
  - https://www.dropbox.com/s/97m31frthj4ac28/HOMEWORK%20GRADING%20INSTRUCTIONS%20for%20MIDS%20MLS?dl=0
- **Homework HW4 Folder Questions + Data**
  - HW4 is a group oriented homework
  - https://www.dropbox.com/sh/m0nxsf4vs5cyrp2/AABh8Rp-kZNKPnWzs4Ksx-i5a?dl=0
- **Team assignments**
  - https://docs.google.com/spreadsheets/d/1ncFQl5Tovn-16slD8mYjP_nzMTPSfiGeLLzW8v_sMjg/edit?usp=sharing
- **Please submit your homeworks (one per team) going forward via this form (and not thru the ISVC):**
  - https://docs.google.com/forms/d/1ZOr9Rnle_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form

# Live Session Outline

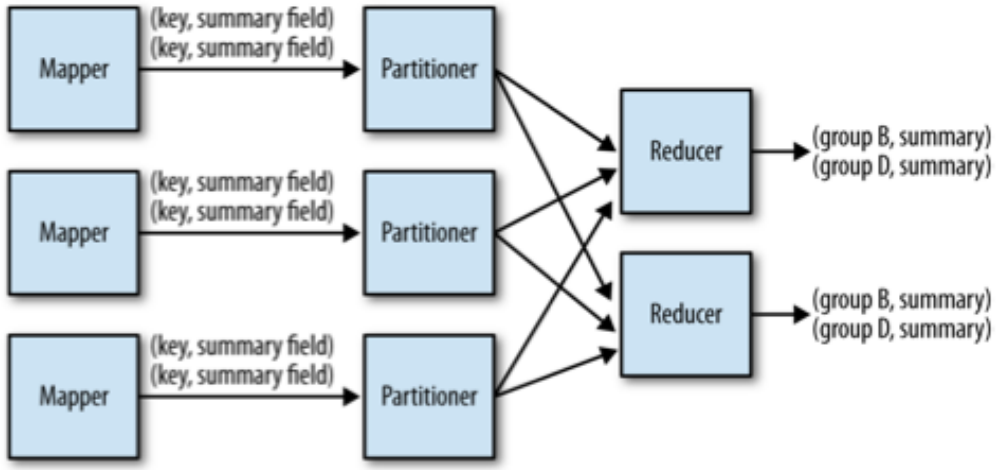
- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# MapReduce→ Hadoop → MRJob

- **Analyzes *raw* data in HDFS *where the data is***
- **Jobs are split into Mappers and Reducers**

**Mappers (you code this)**
- Loads data from HDFS
- Filter, transform, parse
- Outputs (key, value) pairs

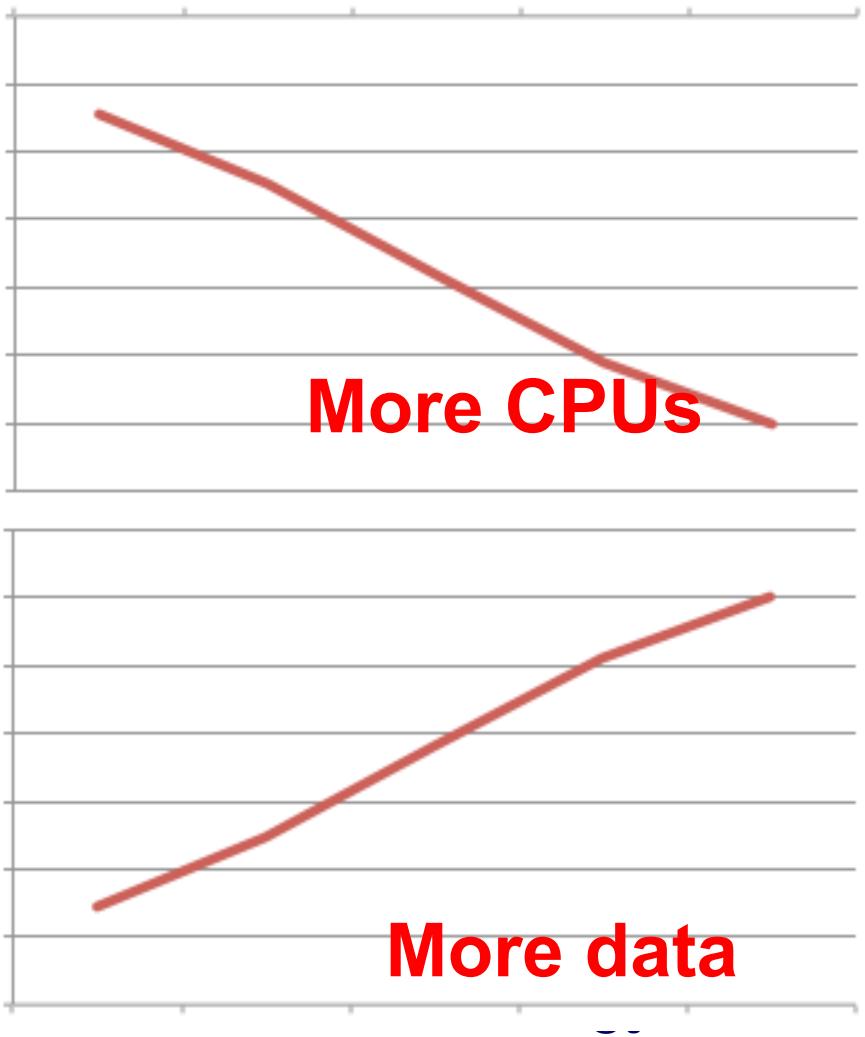
**Reducers (you code this, too)**
- Automatically Groups by the mapper's output key
- Aggregate, count, statistics
- Outputs to HDFS



# Cool Thing #1: Linear Scalability +Ship code to workers

- HDFS and MapReduce scale linearly
- If you have twice as many computers, jobs run twice as fast
- If you have twice as much data, jobs run twice as slow
- If you have twice as many computers, you can store twice as much data

**DATA LOCALITY!!**



# Cool Thing #2: Schema on Read

## BEFORE:

ETL, SCHEMA DESIGN UPFRONT,
TOSSING OUT ORIGINAL DATA,
COMPREHENSIVE DATA STUDY

## WITH HADOOP:

### LOAD DATA FIRST, ASK QUESTIONS LATER

**Data is parsed/interpreted as it is loaded out of HDFS**

***What implications does this have?***

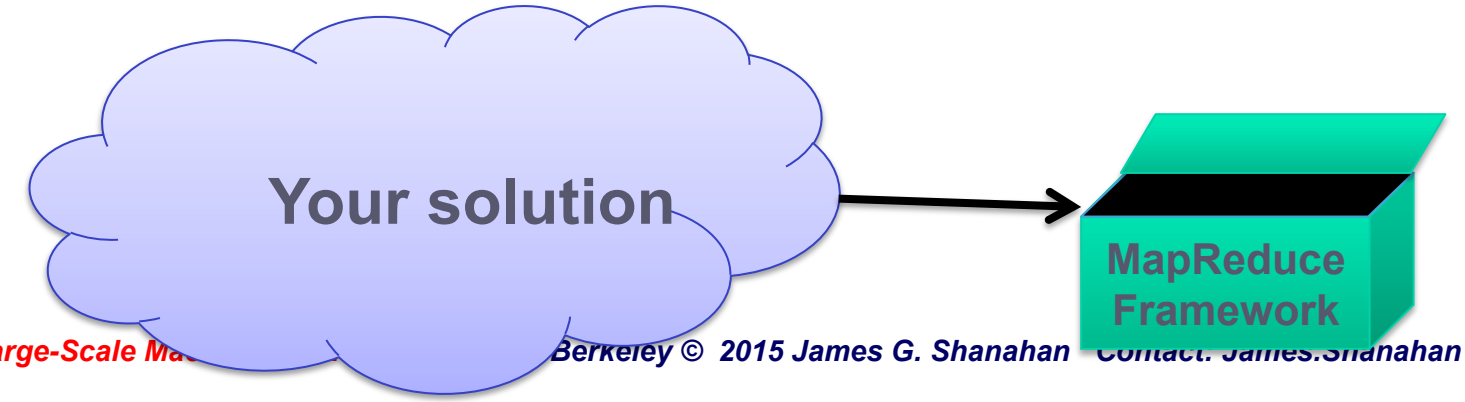
**Keep original data around!**
**Have multiple views of the same data!**
**Work with unstructured data sooner!**
**Store first, figure out what to do with it later!**

# Cool Thing #3: Transparent Parallelism

Code deployment?
Scalability?
Threading?
RPC?
Network programming?
Message passing?
Data storage?
Distributed stuff?
Locking?
Inter-process communication?
Fault tolerance?
Data center fires?

**With MapReduce, I DON'T CARE**

***… I just have to be sure my solution fits into this tiny box***



# Cool Thing #4: Unstructured Data

- **Unstructured data:**
  **media, text,**
  **forms, log data**
  **lumped structured data**

- **Query languages like SQL and Pig assume some sort of "structure"**
- **MapReduce is just Java:**
  **You can do anything Java can do in a Mapper or Reducer**

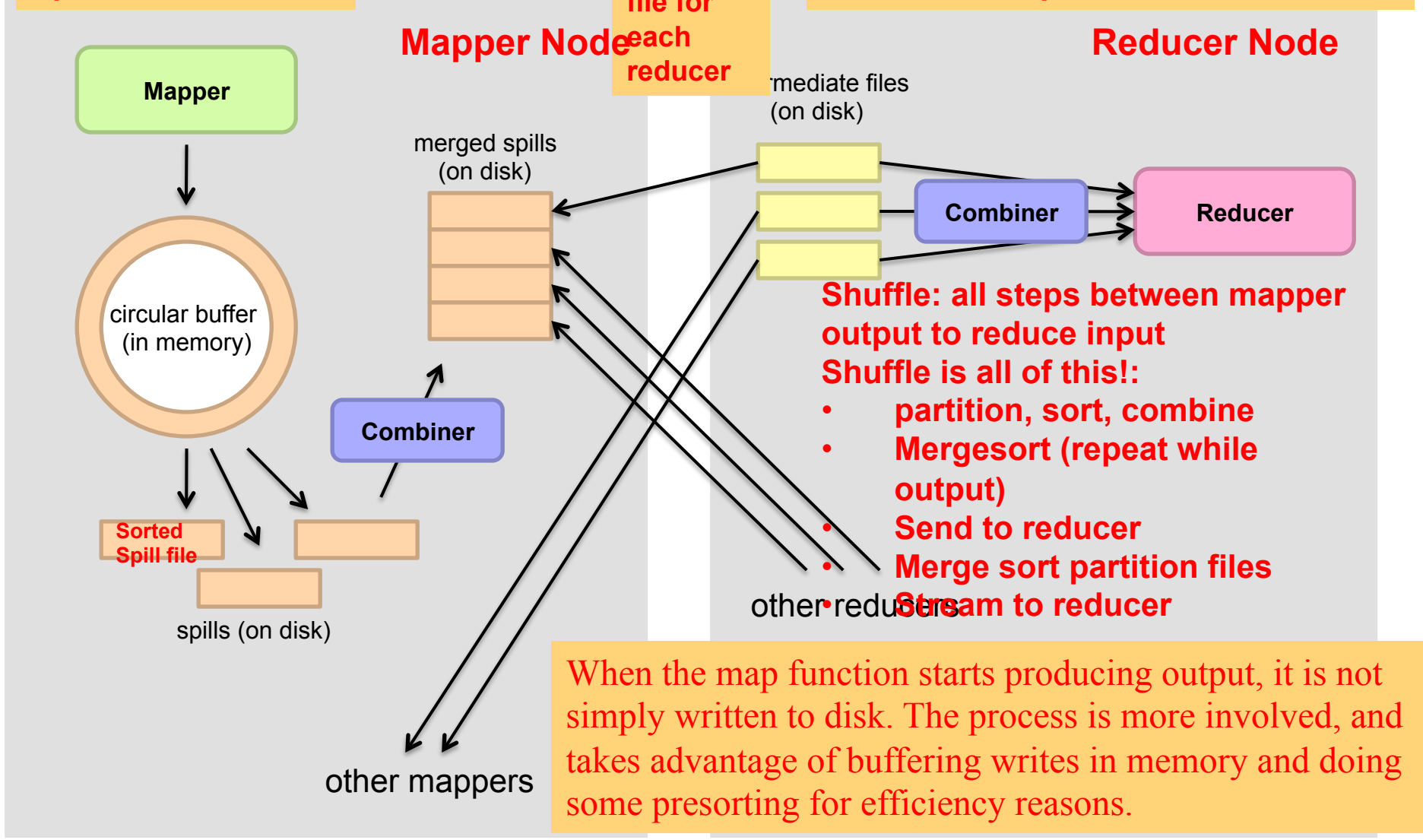
First of all shuffling is the process of transferring data from the mappers to the reducers

# Summary: Hadoop Shuffle

**Shuffle: partition, sort, combine Spill to disk**

**Ship a single partition file for each reducer**

**Merge the partition file from each mapper, Combine, and generate a stream of inputs for the reducerd**




**Shuffle: all steps between mapper output to reduce input**

**Shuffle is all of this!:**

- **partition, sort, combine**
- **Mergesort (repeat while output)**
- **Send to reducer**
- **Merge sort partition files**
- **Stream to reducer**

When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons.

# What is MRJob

- **Mrjob is a Python package that helps you write and run Hadoop streaming jobs.**
  - assists you in submitting your job to the Hadoop job tracker and in running each individual step under Hadoop Streaming.
- **Developed at Yelp.com**
- **It's offered under the Apache 2.0 license.**
- **The software was written to power a feature called "People Who Viewed this Also Viewed."**
  - (Regular users will see it in the lower right-hand corner of the screen when they look at popular content.);
  - used for advertising etc..

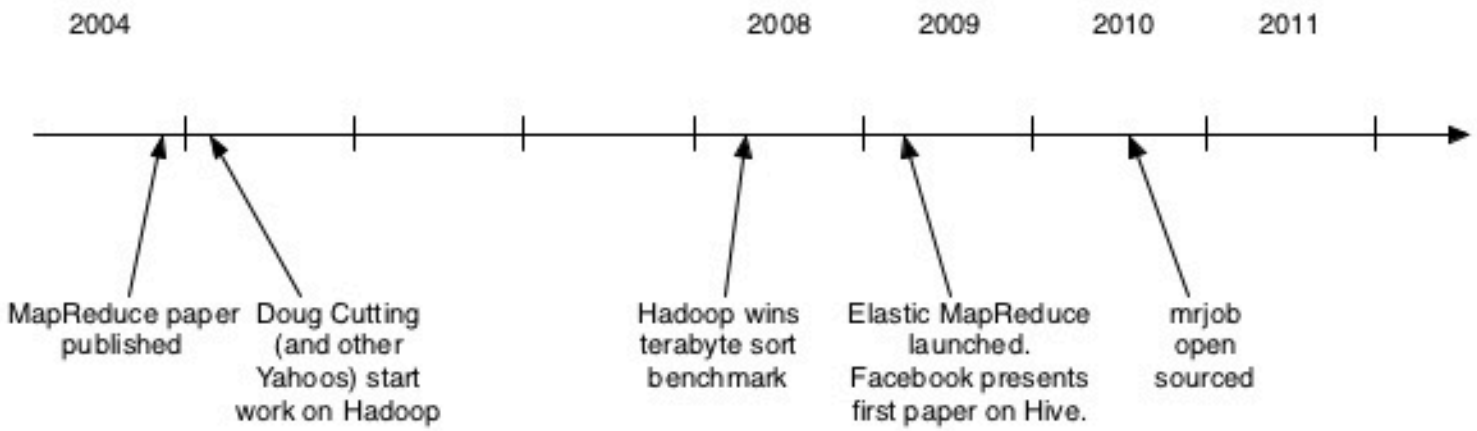
# Why MrJob

- **mrjob lets you write MapReduce jobs in Python 2.5+ and run them on several platforms.**
- **You can:**
  - Write multi-step MapReduce jobs in pure Python
  - Test on your local machine
  - Run on a Hadoop cluster
  - Run in the cloud using Amazon Elastic MapReduce (EMR)
  - Keep all MapReduce code for one job in a single class
  - Switch input and output formats with a single line of code
  - Simple and easy to learn!!!

- **mrjob is licensed under the Apache License, Version 2.0.**

# MapReduce→ Hadoop → MRJob

Timeline



# WordCount Mapper: MrJob vs. Hadoop Streaming

```python
#!/usr/bin/env python

from mrjob.job import MRJob

class MRwordcount(MRJob):

    def mapper(self, _, line):
        line = line.strip()
        keys = line.split()
        for key in keys:
            value = 1
            yield key, value

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRwordcount.run()
```

**Hadoop Streaming requires managing I/O etc.**

```python
for line in sys.stdin:
    line = line.strip()
    keys = line.split()
    for key in keys:
        value = 1
        print('%s\t%d' % (key, value))
```

## Unit 4 | MRJob, Unsupervised Learning at Scale: Clustering, Canopy-Based K-Means, and Expectation Maximization

Hide Contents

- 4.1 Weekly Introduction (2 mins )
- 4.2 Assigned Readings
- 4.3 Background and Motivation (5 mins )
- 4.4 mrjob Installation (4 mins )
  - 4.4.1 Quiz: Install mrjob and Verify
- 4.5 mrjob Fundamentals and Concepts (7 mins )
- 4.6 Writing mrjob Code (10 mins )
  - 4.6.1 Quiz: Word Frequency Challenge
- 4.7 Log File Processing (8 mins )
  - 4.7.1 Quiz: Log File Processing Challenge
- 4.8 Thought Experiment Bad Logs (5 mins )
- 4.9 Serializable JSON (15 mins )
- 4.10 mrjob Benchmarks (7 mins )
- 4.11 Clustering Overview (7 mins )
- 4.12 K-Means Algorithm (7 mins )

# Multistep MR Jobs

## Multi-Step

- Not all computations can be done in a single MapReduce step
- Map Input: <key, value>
- Reducer Output: <key, value>
- Compose MapReduce steps!

## Output as Input

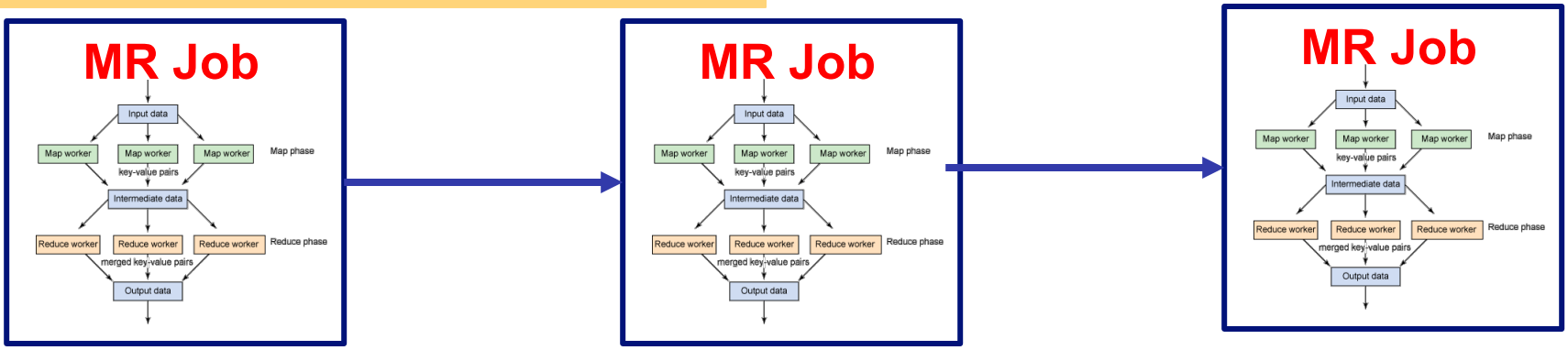
- The output of one MapReduce job can be used as the input to another

## Examples

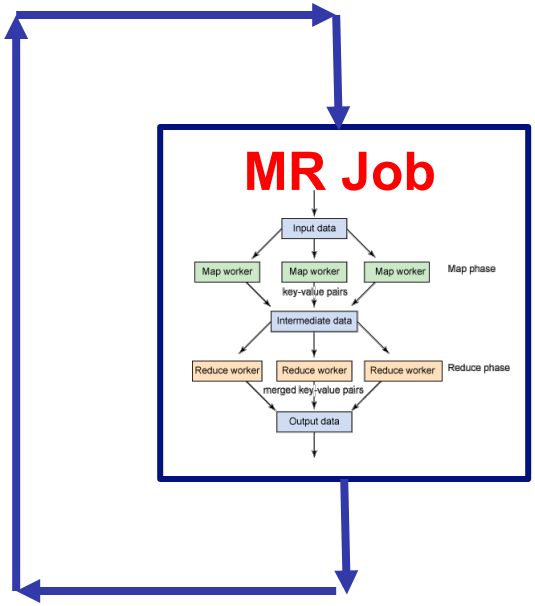
- PageRank: Multiple steps till solution converges
- Multi-level summaries

# Pipeline of jobs

## Pipeline of jobs in sequence




## Iterative Pipeline



# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# Canonical Word Count

**Object oriented framework**

**MrJob Class**
- **Methods**
- **Variables**

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))


if __name__ == '__main__':
    MRWordFreqCount.run()
```

**MRWordFreqCount Cla**

**Key, value record**

# Canonical Word Count

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")
```

The quick brown fox jumps over the lazy dog

Single record at a time

```
def mapper(self, _, line):
    for word in WORD_RE.findall(line):
        yield (word.lower(), 1)

def reducer(self, word, counts):
    yield (word, sum(counts))


if __name__ == '__main__':
    MRWordFreqCount.run()
```

the, 1
quick, 1
brown, 1
fox, 1
jumps, 1
over, 1
the, 1
lazy, 1
dog, 1

# Canonical Word Count

Mapper output

The 1
X 1
Y 1
The 1
The 1

Reducer input

The, [1, 1,1]
X, 1
Y, 1

Less book-keeping in MrJo

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))


if __name__ == '__main__':
    MRWordFreqCount.run()
```

I like this Hadoop thing

i, 1
like, 1
this, 1
hadoop, 1
thing, 1

# Canonical Word Count

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))


if __name__ == '__main__':
    MRWordFreqCount.run()
```

dog, [1, 1, 1, 1, 1, 1]

dog, 6

# Canonical Word Count

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))


if __name__ == '__main__':
    MRWordFreqCount.run()
```

**No combiner in this example**

cat, [1, 1, 1, 1, 1, 1, 1, 1] → cat, 8

# MRJOB DEMO!

# Example: find most frequent word

## MrJob Pipeline

- **Example 2: find most frequent word**
  - 1st MapReduce Job **WORDCOUNT**
    - Mapper
      - Read data from InputFile
      - Split it into words
      - Output <word, 1 > tuples
    - Reducer
      - Read the results of Mapper
      - Sum the occurrences of each word to a final count
      - Output its results <word, Sum > tuples to
  - 2nd MapReduce Job **SORT in descreasing order of frequency**
    - Mapper
      - Nothing to do
    - Reducer
      - Select the maximum count tuples

# Writing Mrjob code

- **Example 2: MostUsedWord (Cont.)**

  Command Line:

  $python MostUsedWord.py inputfile.txt

# MrJob class for wordcount

https://www.dropbox.com/s/swlp55eqqvdvq1x/MrjobWordCount.ipynb?dl=0

```
]:
```

```
%%writefile WordCount.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def combiner(self, word, counts):
        yield word, sum(counts)

    #hello, (1,1,1,1,1,1): using a combiner? NO and YEs
    def reducer(self, word, counts):
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
Writing WordCount.py
```

After much dabbling I realized how wrong I was. Python's generators are indeed cool. They give us the use of infinite lists and they're useful for conserving memory usage, but lazy evaluation they can't quite do.

Let's make a generator for natural numbers:

```
def generator():
    i = 1
    while True:
        yield i
        i += 1
```

A simple function with a loop counting from one to infinity. The *yield* operator is what saves us from looping into infinity by turning the function into a generator. We can now take any number of natural numbers:

```
from itertools import islice

def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))

print take(5, generator())
# [1, 2, 3, 4, 5]
```

Cool, we've implemented python's native *range* function. Handy, bot nothing special.

**yield in python: a form of lazy evaluation**

# Example: find most frequent word

### Wrtie some words to MostUsedWord file

Generate data

```
!echo foo foo quux labs foo bar quux hell > MostUsedWord.txt
!echo labs labs hello fool labs >>MostUsedWord.txt
!echo california new york LA San Francisco >>MostUsedWord.txt
```

### Write MostusedWord MrJob class

Need to override steps() function
It has two MapReduce Jobs.

```
%%writefile Mostused.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re
WORD_RE = re.compile(r"[\w']+")
class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRJobStep(mapper=self.mapper_get_words,
                      combiner=self.combiner_count_words,
                      reducer=self.reducer_count_words),
            MRJobStep(reducer=self.reducer_find_max_word),
        ]
    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)
    def combiner_count_words(self, word, counts):
        # optimization: sum the words we have seen so far
        yield (word, sum(counts))
    def reducer_count_words(self, word, counts):
        # send all (num_occurrences,work) paris to the same reducer.
        # num_occurrences is the key, then we can easily use max funciton to get most used word
        yield None, (sum(counts), word)
        # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)
if __name__ == '__main__':
    MRMostUsedWord.run()
```

```
class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRJobStep(mapper=self.mapper_get_words,
                      combiner=self.combiner_count_words,
                      reducer=self.reducer_count_words),
            MRJobStep(reducer=self.reducer_find_max_word),
        ]
```

Step1:Job1

Step2: Job2

Word Count job

Reducer 2: Max

```
Overwriting Mostused.py
```

It has two mapreduce. The first Mapreduce Mapper outputs (word, 1) key value pairs, and then conbiner combines the sum locally. At last, Reducer sums them up. The second mapreduce output the word having max number of count.

### Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4	"labs"
```

https://www.dropbox.com/s/nd2wow1t3y77jqk/MrjobMostUsedWord.ipynb?dl=0

- ...

```
%%writefile Mostused.py

from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRMostUsedWord(MRJob):

#     OUTPUT_PROTOCOL = JSONValueProtocol

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)

    def steps(self):
        return [
            self.mr(mapper=self.mapper_get_words,
                    combiner=self.combiner_count_words,
                    reducer=self.reducer_count_words),
            self.mr(reducer=self.reducer_find_max_word)
        ]

if __name__ == '__main__':
    MRMostUsedWord.run()
```

```
Overwriting Mostused.py
```

```
!python WordCount.py --jobconf -numReduceTasks=3 WordCount.txt --output-dir mrJobOutput


# mr_your_job.py --jobconf mapred.map.tasks=23 --jobconf
#> mapred.reduce.tasks=42
```

```
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner'

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --
strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-
protocols

creating tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mappe
r_part-00000
Counters from step 1:
  (no counters found)
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mapper-
sorted
> sort /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mapper_par
t-00000
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-reduce
r_part-00000
Counters from step 1:
  (no counters found)
Moving /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-reducer_par
t-00000 -> mrJobOutput/part-00000
Streaming final output from mrJobOutput
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
"mining"        1
"quux"  2
"science"       1
removing tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289
```

```
!ls mrJobOutput/*
!cat mrJobOutput/part-00000
```

```
mrJobOutput/part-00000
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
```

# Example: find most used word

## Wrtie some words to MostUsedWord file

```
!echo foo foo quux labs foo bar quux hell > MostUsedWord.txt
!echo labs labs hello fool labs >>MostUsedWord.txt
!echo california new york LA San Francisco >>MostUsedWord.txt
```

## Write MostusedWord MrJob class

```
%%writefile Mostused.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re
WORD_RE = re.compile(r"[\w']+")
class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRJobStep(mapper=self.mapper_get_words,
               combiner=self.combiner_count_words,
               reducer=self.reducer_count_words),
            MRJobStep(reducer=self.reducer_find_max_word),
        ]
    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)
    def combiner_count_words(self, word, counts):
        # optimization: sum the words we have seen so far
        yield (word, sum(counts))
    def reducer_count_words(self, word, counts):
        # send all (num_occurrences,work) paris to the same reducer.
        # num_occurrences is the key, then we can easily use max funciton to get most used word
        yield None, (sum(counts), word)
        # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)
if __name__ == '__main__':
    MRMostUsedWord.run()
```

```
Overwriting Mostused.py
```

It has two mapreduce. The first Mapreduce Mapper outputs (word, 1) key value pairs, second mapreduce output the word having max number of count.

**Run this program from the command line**
**The most frequent word is “labs”**

## Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4	"labs"
```


## Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4	"labs"
```

# Running the Example: find most used word

Mostused.py

**Run the code in command line**

```
1 !python Mostused.py MostUsedWord.txt
```

```
4       "labs"
```

**Versus**

- **Or run the driver from the Notebook and get the result**

**Run the code though python driver**

```
1 from Mostused import MRMostUsedWord
2 mr_job = MRMostUsedWord(args=['MostUsedWord.txt'])
3 with mr_job.make_runner() as runner:
4     runner.run()
5     # stream_output: get access of the output
6     for line in runner.stream_output():
7         print mr_job.parse_output_line(line)
```

```
(4, u'labs')
```

- **Counters in MrJob**

# Example of counter customization in Mrjob

- **Count how many times mapper and reducer function are called**

**MrJob class for Counter**

https://www.dropbox.com/s/5thl14n4pqvhzt5/Counter.ipynb?dl=0

```
%%writefile mr_wc_counter.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        self.increment_counter('group', 'Num mapper calls', 1)
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def reducer(self, word, counts):
        self.increment_counter('group', 'Num_reducer_calls', 1)
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

**Note: no explicit declaration required**

# Example of counter customization in Mrjob(Cont.)

- **Input:** foo foo quux labs foo bar quux
- **Driver:**

```
1 from mr_wc_counter import MRWordFreqCount
2 mr_job = MRWordFreqCount(args=['WordCount.txt'])
3 with mr_job.make_runner() as runner:
4     runner.run()
5     # stream_output: get access of the output
6     print runner.counters()
```

- **Output:**

```
[{'group': {'Num_mapper_calls': 1, 'Num_reducer_calls': 4}}]
```

https://www.dropbox.com/s/5thl14n4pqvhzt5/Counter.ipynb?dl=0

# Accessing a component of a component

```
{
    "impression": {
        "extra": {
            "candidate_scores": {
                "category": 0.29999999999999999,
                "score_final": 0.46801499999999996,
                "score_mixed": 0.46801499999999996,
                "freshness": 0.5,
                "place": 0.0,
                "ecpm": 1.0,
                "location": 0.0001
            },
            "is_billable": true
        },
        "salesperson_id": 559,
        "ad_type": "announcement",
        "impresssion_id": "e26ad2b87823e437",
        "program": "CPC",
        "candidate_id": 200587,
        "scorer": "remnant",
        "position": 0,
        "content_id": 453333,
        "max_price": [
            "USD",
            200
        ],
        "win_criteria": {
```

## JSON Objects

**{“impression:” {**
**“extra:”{…**
**}**
**“Candidate_id:” 200587**
**….}**
**In python access object attributes as follows:**
**ad_campaign = ad_event[‘impression’][‘candidate_id’]**
**200587**

# Running MrJobs on the cloud (e.g., EMR)

**mrjob v0.4.2 documentation**

Home » Guides » Elastic MapReduce ← Elastic MapReduce | EMR runner options →

**Table Of Contents**



**Need help?**

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe @googlegroups.com.

type to search

## Elastic MapReduce Quickstart

### Configuring AWS credentials

Configuring your AWS credentials allows mrjob to run your jobs on Elastic MapReduce and use S3.

- Create an Amazon Web Services account
- Sign up for Elastic MapReduce
- Get your access and secret keys (click "Security Credentials" on your account page)

Now you can either set the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or set **aws_access_key_id** and **aws_secret_access_key** in your `mrjob.conf` file like this:

```
runners:
  emr:
    aws_access_key_id: <your key ID>
    aws_secret_access_key: <your secret>
```

### Configuring SSH credentials

Configuring your SSH credentials lets mrjob open an SSH tunnel to your jobs' master nodes to view live progress, see the job tracker in your browser, and fetch error logs quickly.

- Go to https://console.aws.amazon.com/ec2/home
- Make sure the **Region** dropdown (upper left) matches the region you want to run jobs in (usually "US East").
- Click on **Key Pairs** (lower left)
- Click on **Create Key Pair** (center).
- Name your key pair `EMR` (any name will work but that's what we're using in this example)
- Save `EMR.pem` wherever you like (`~/.ssh` is a good place)
- Run `chmod og-rwx /path/to/EMR.pem` so that `ssh` will be happy
- Add the following entries to your `mrjob.conf`:

```
runners:
  emr:
```

# EMR and MrJob: Homework Challenge

## How To

```
python code/unique_review.py -r emr -c mrjob.conf s3://i290-jblomo/data/selected_reviews.json.gz
```

- Simply specify `-r emr`

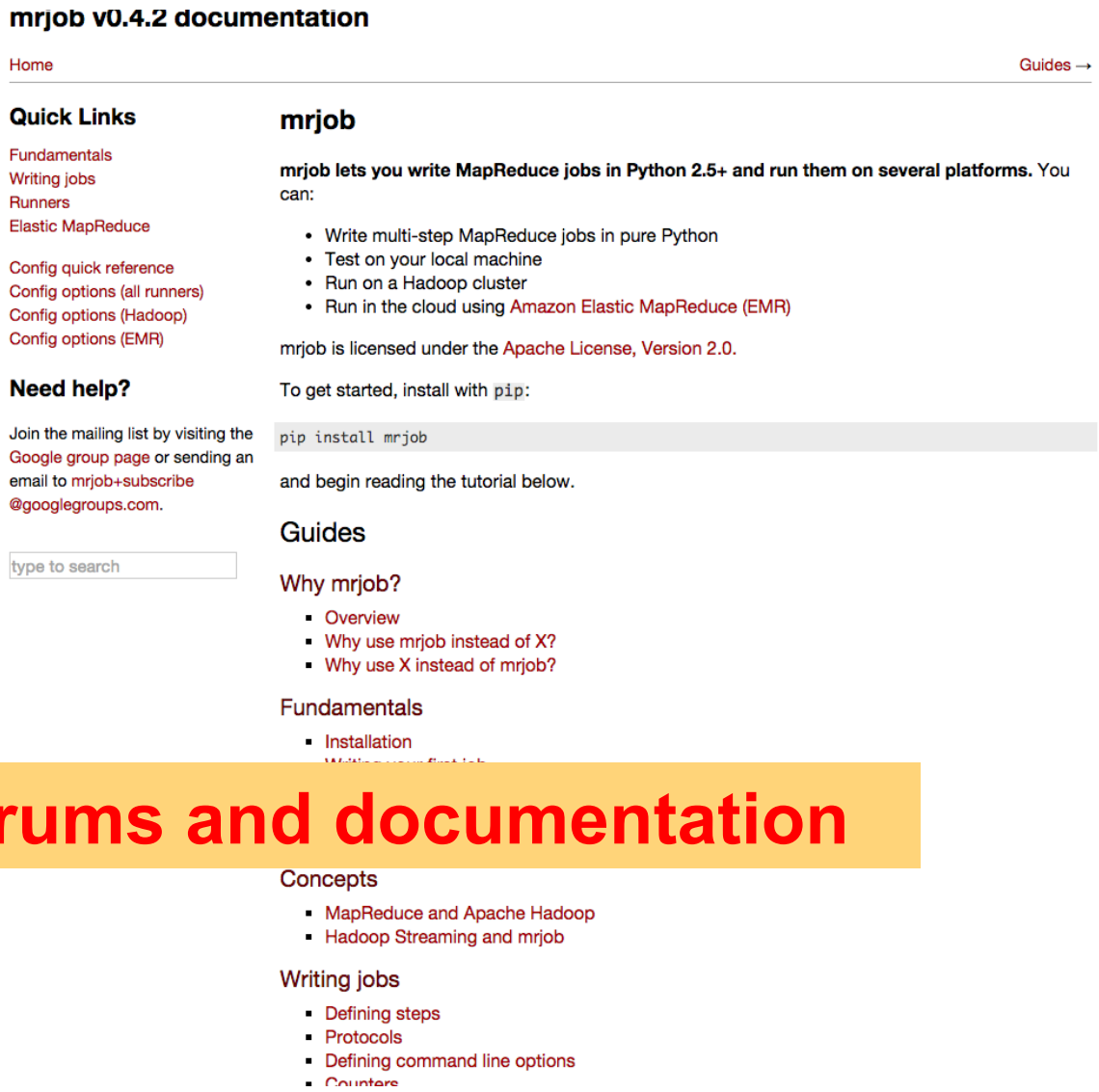
## mrjob Workflow

- Develop locally on subset of data
- Run on EMR on all data located in S3 bucket
- Either save output in S3, or stream locally

**ToDo get code and yelp/selected_reviews.json**

# MrJob Online forums and documentations

- https://pythonhosted.org/mrjob/



mrjob v0.4.2 documentation

Home Guides →

**Quick Links**

Fundamentals
Writing jobs
Runners
Elastic MapReduce

Config quick reference
Config options (all runners)
Config options (Hadoop)
Config options (EMR)

**Need help?**

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe @googlegroups.com.

type to search

## mrjob

**mrjob lets you write MapReduce jobs in Python 2.5+ and run them on several platforms.** You can:

- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)

mrjob is licensed under the Apache License, Version 2.0.

To get started, install with `pip`:

```
pip install mrjob
```

and begin reading the tutorial below.

### Guides

#### Why mrjob?

- Overview
- Why use mrjob instead of X?
- Why use X instead of mrjob?

#### Fundamentals

- Installation

#### Concepts

- MapReduce and Apache Hadoop
- Hadoop Streaming and mrjob

#### Writing jobs

- Defining steps
- Protocols
- Defining command line options

**Online forums and documentation**

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

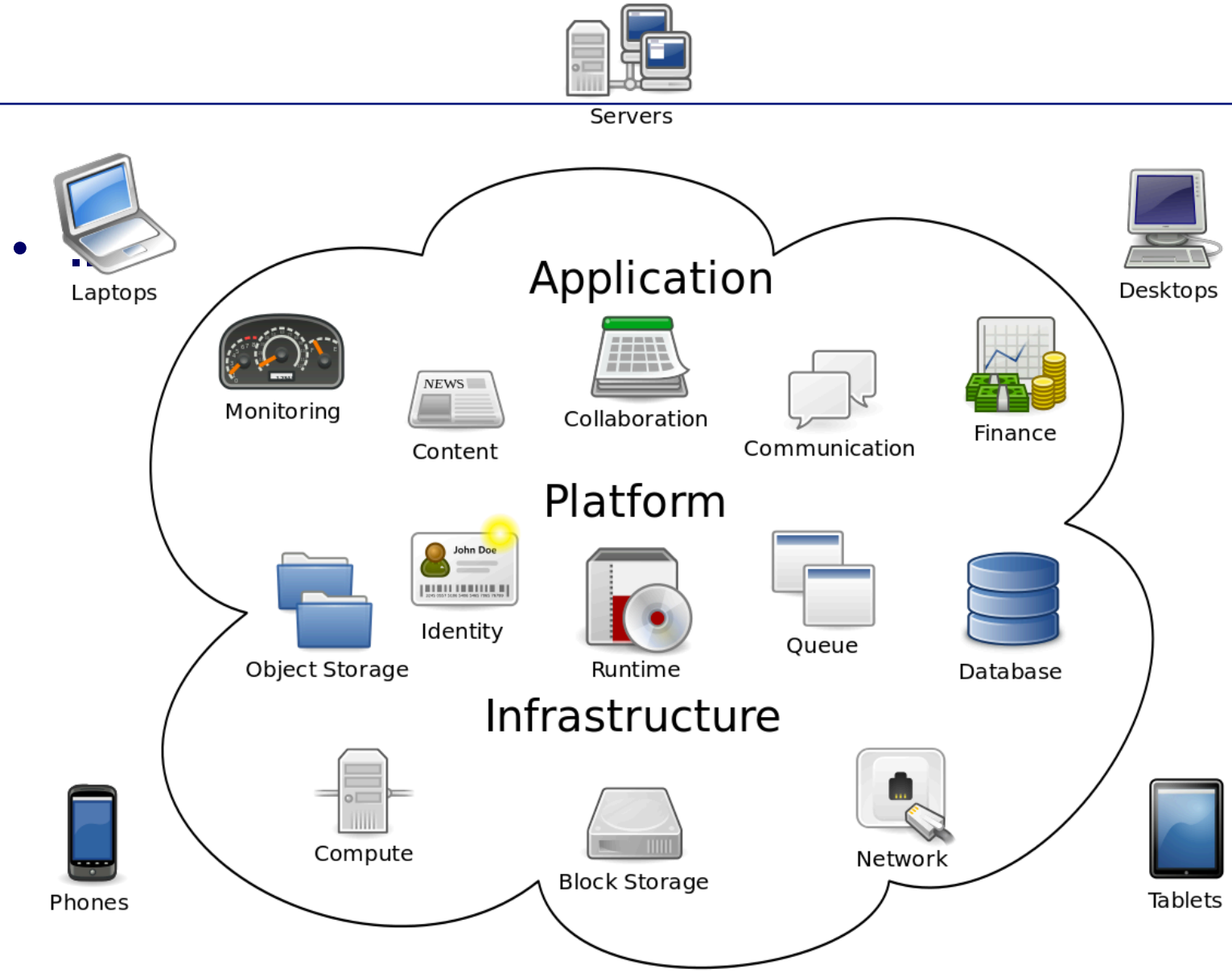
- **MrJob on the cloud**

- **Iaas**
  - (Amazon, Microsoft, Google, etc..)
- **PaaS**
  - Platform as a service (PaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage web applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.[1][2][3] PaaS can be delivered in two ways: as a public cloud service from a provider, where the consumer controls software deployment and configuration settings, and the provider provides the networks, servers, storage and other services to host the consumer's application; or as software installed in private data centers or public infrastructure as a service and managed by internal IT departments

# Cloud Computing

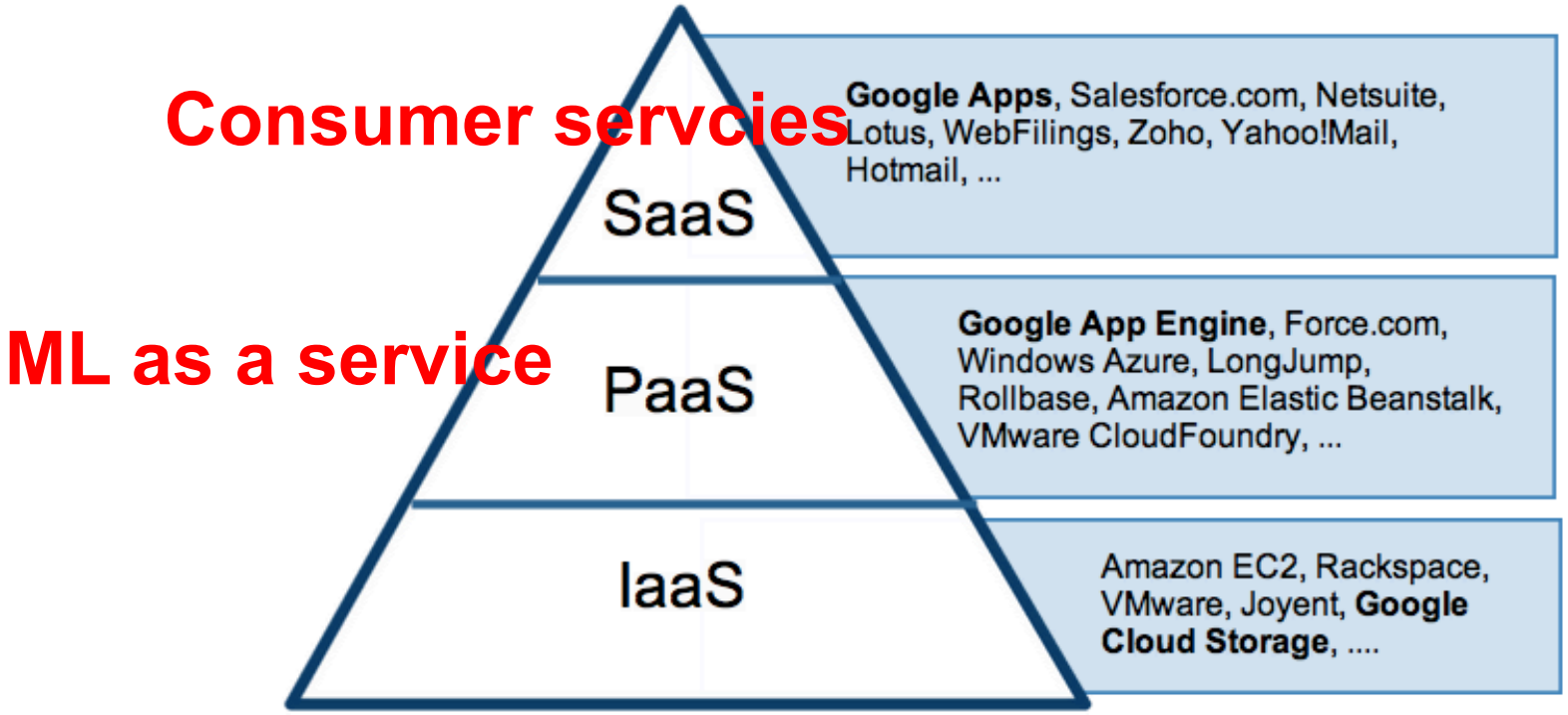
- **Cloud computing is a model for enabling ubiquitous, convenient, on-demand access to a shared pool of configurable computing resources.**
  - Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in third-party data centers.
  - It relies on sharing of resources to achieve coherence and economies of scale, similar to a utility (like the electricity grid) over a network.
  - At the foundation of cloud computing is the broader concept of converged infrastructure and shared services.
- **Cloud computing, or in simpler shorthand just "the cloud", als**

https://en.wikipedia.org/wiki/Cloud_computing




# Cloud Computing

# Cloud Computing as Gartner Sees It



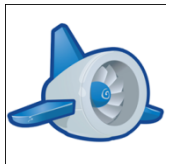

Source: Gartner AADI Summit Dec 2009

## IaaS

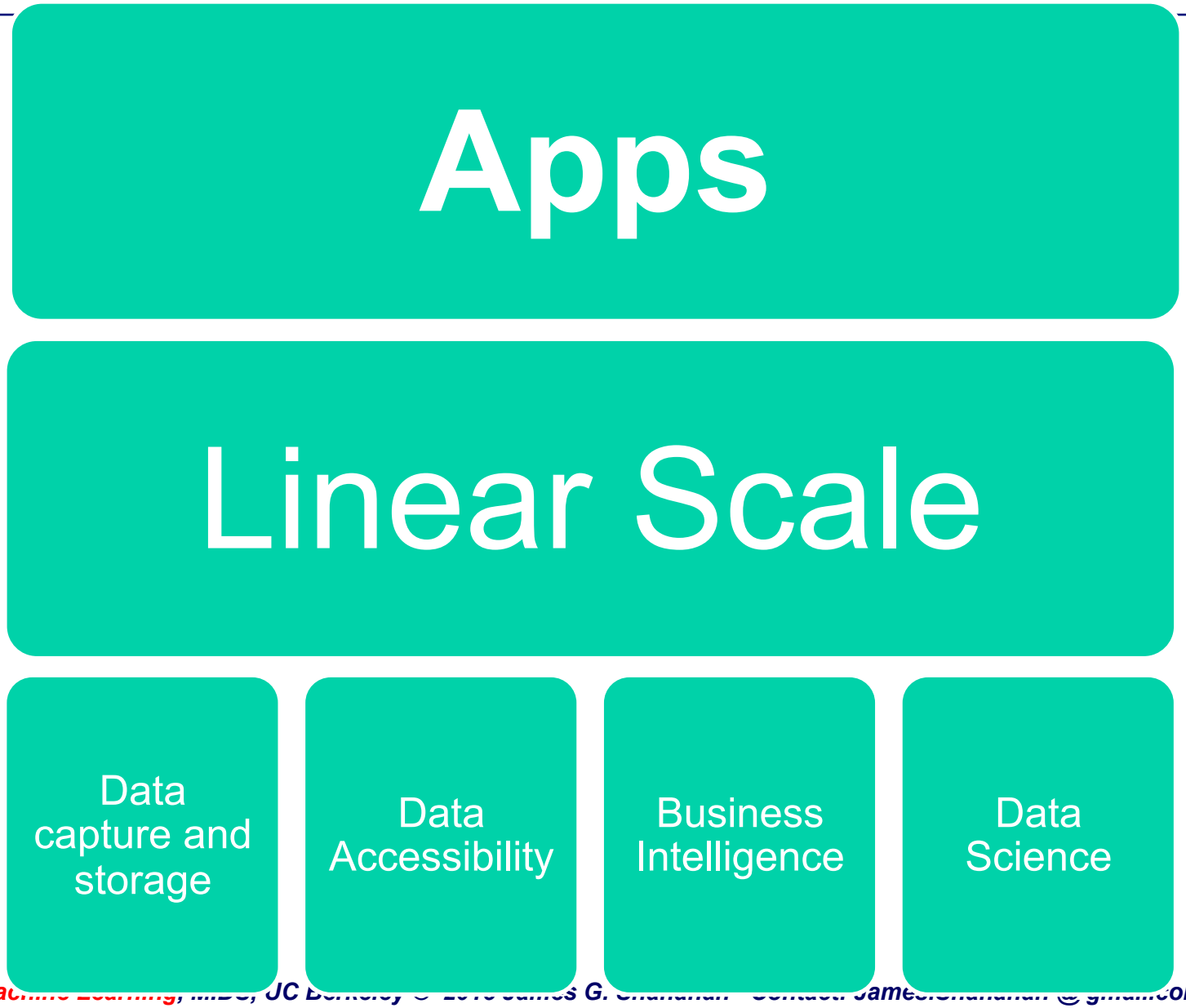
Services such as Amazon EC2 are primarily positioned as **IaaS** (*Infrastructure as a Service*). In this model, the user is given low-level resource management mechanisms (VM provisioning, distributed storage, etc.) and it is up to them to develop their application stack (operating system, HTTP server, MVC framework, database, etc.). While this provides extreme flexibility for the user, it also means that they must manage scaling and fault tolerance on their own.

# PaaS

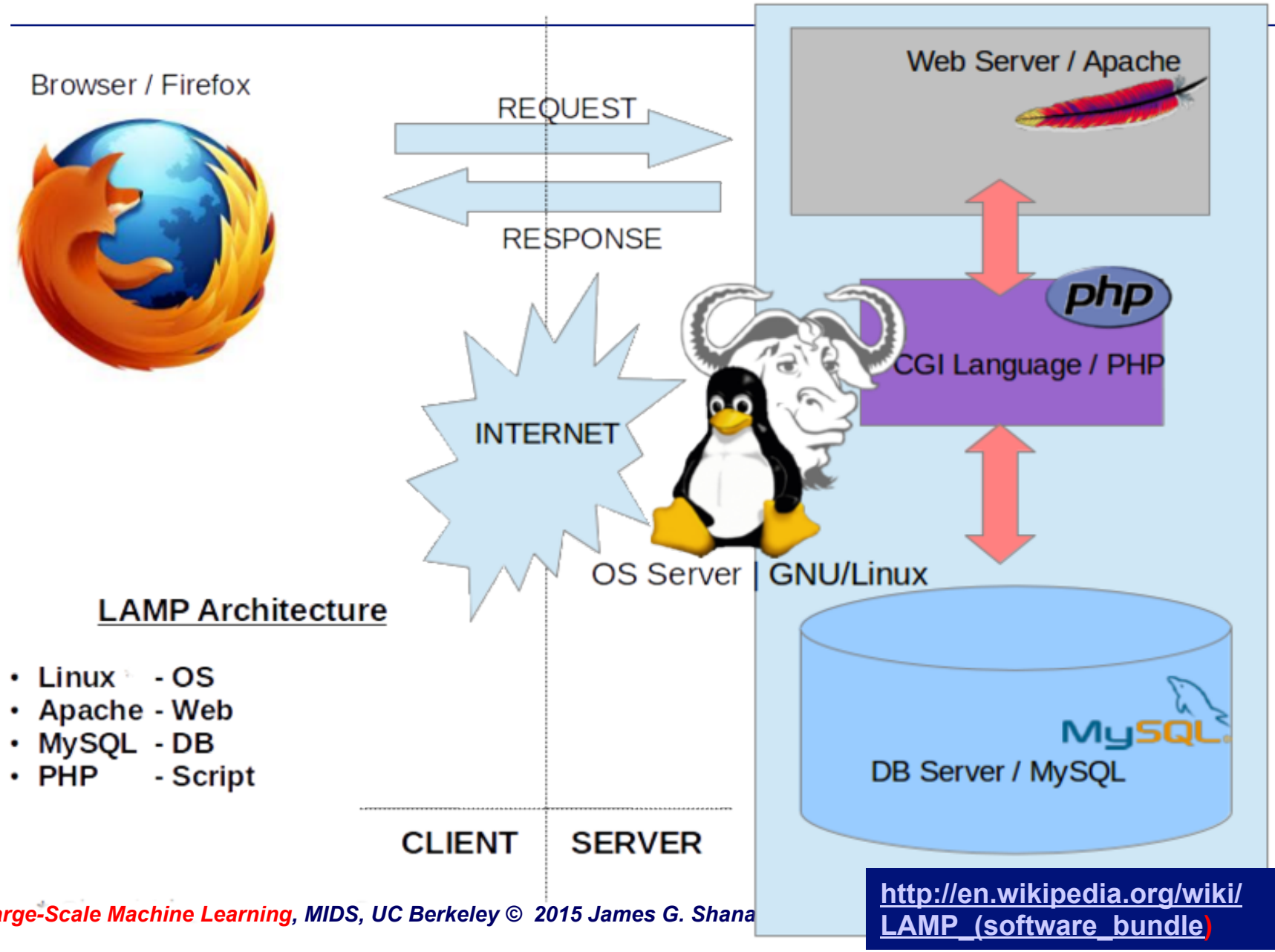
- **Amazon AWS: EC2 & S3 (among the ma infrastructure services)**
  - Linux machine
  - Windows machine
  - A three-tier enterprise application
- **Google app Engine**
  - Eclipse plug-in for GAE
  - Development and deployment of an application
- **Windows Azure**
  - Storage: blob store/container
  - MS Visual Studio Azure development and production environment



# Scale is everything (almost)



# LAMP Stack

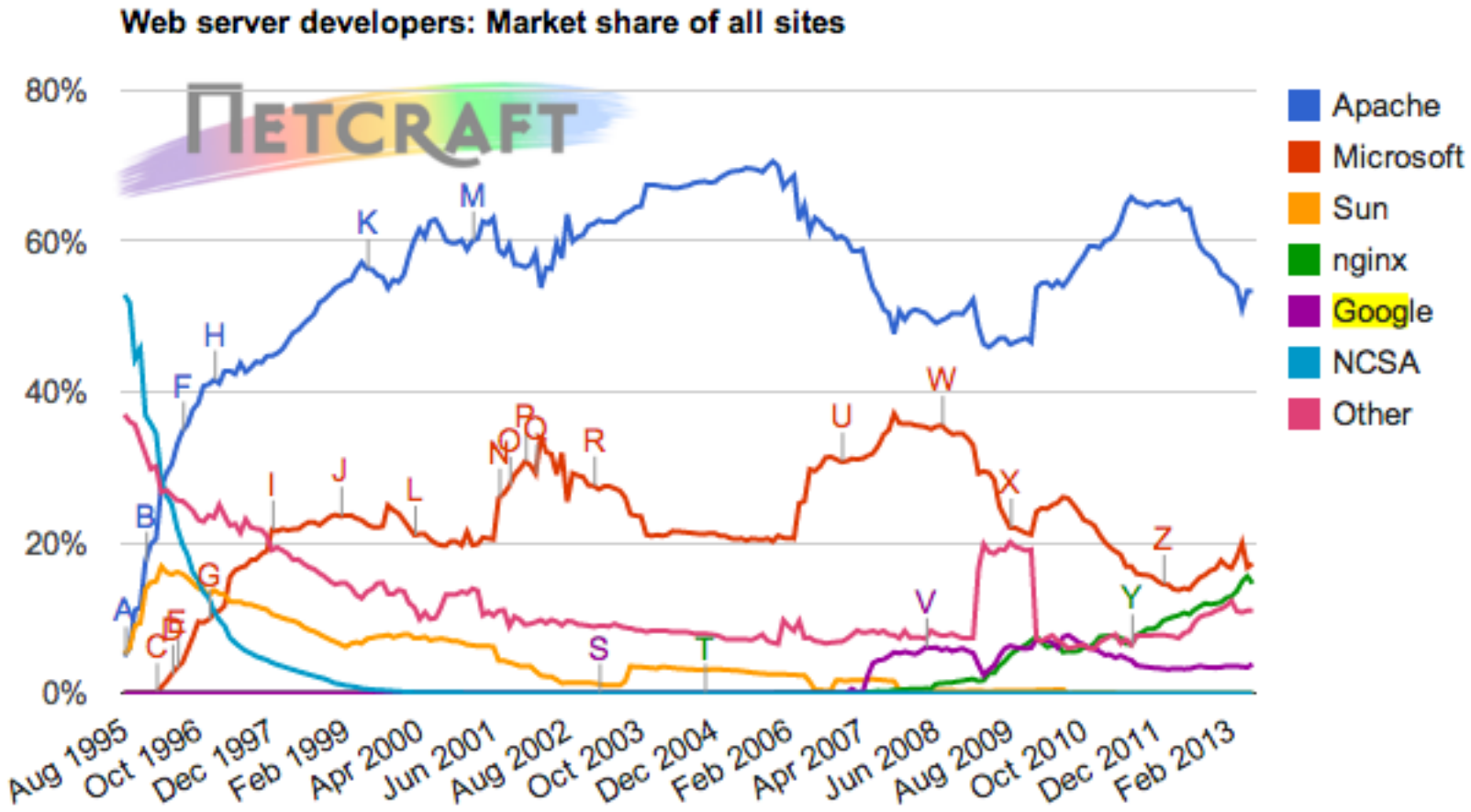



LAMP Architecture

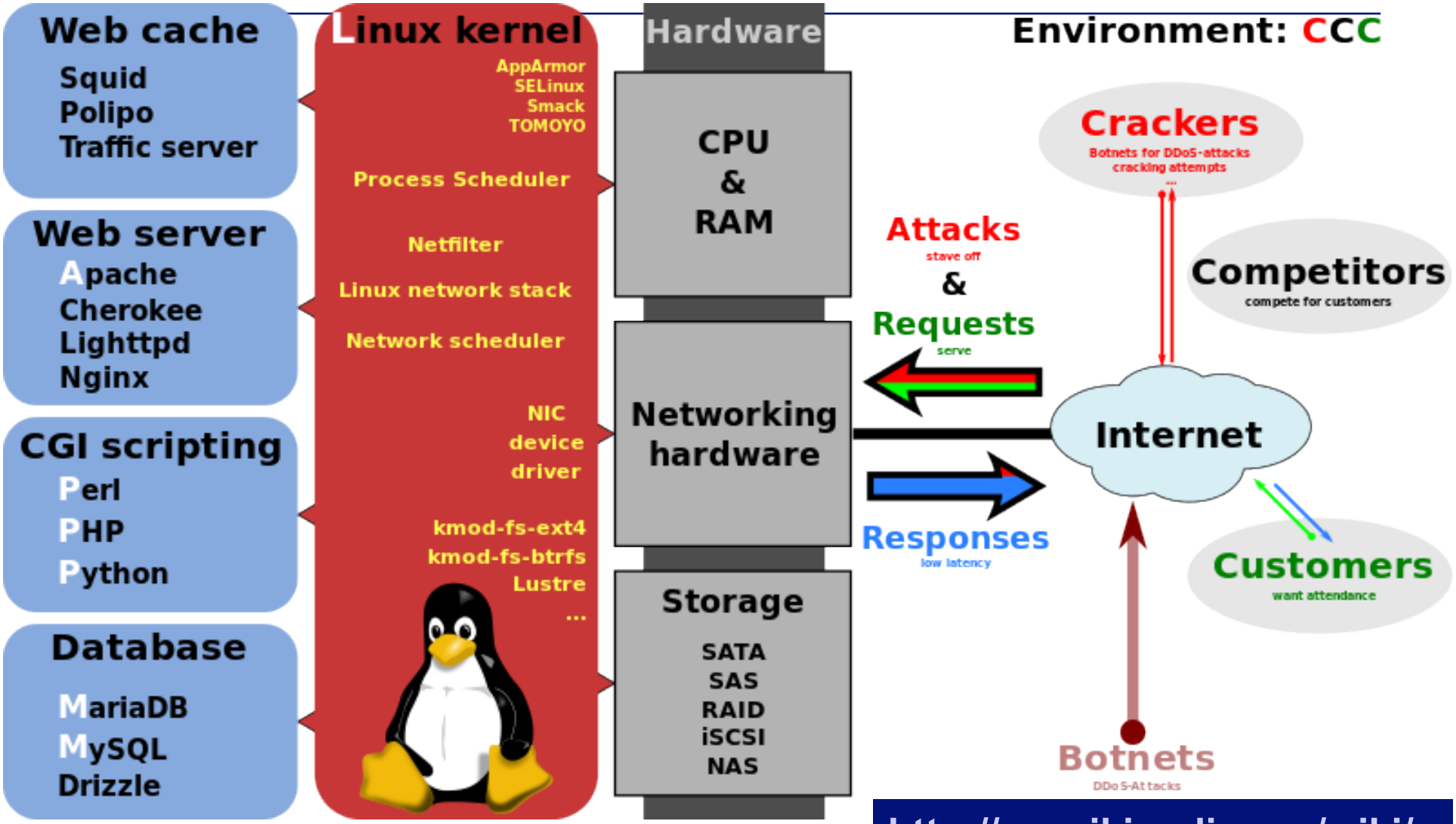
- Linux - OS
- Apache - Web
- MySQL - DB
- PHP - Script

http://en.wikipedia.org/wiki/LAMP_(software_bundle)

# Web servers


Web server developers: Market share of all sites
NETCRAFT
80%
60%
40%
20%
0%
Aug 1995
Oct 1996
Dec 1997
Feb 1999
Apr 2000
Jun 2001
Aug 2002
Oct 2003
Dec 2004
Feb 2006
Apr 2007
Jun 2008
Aug 2009
Oct 2010
Dec 2011
Feb 2013
Apache
Microsoft
Sun
nginx
Google
NCSA
Other

# LAMP Stack




http://en.wikipedia.org/wiki/LAMP_(software_bundle)

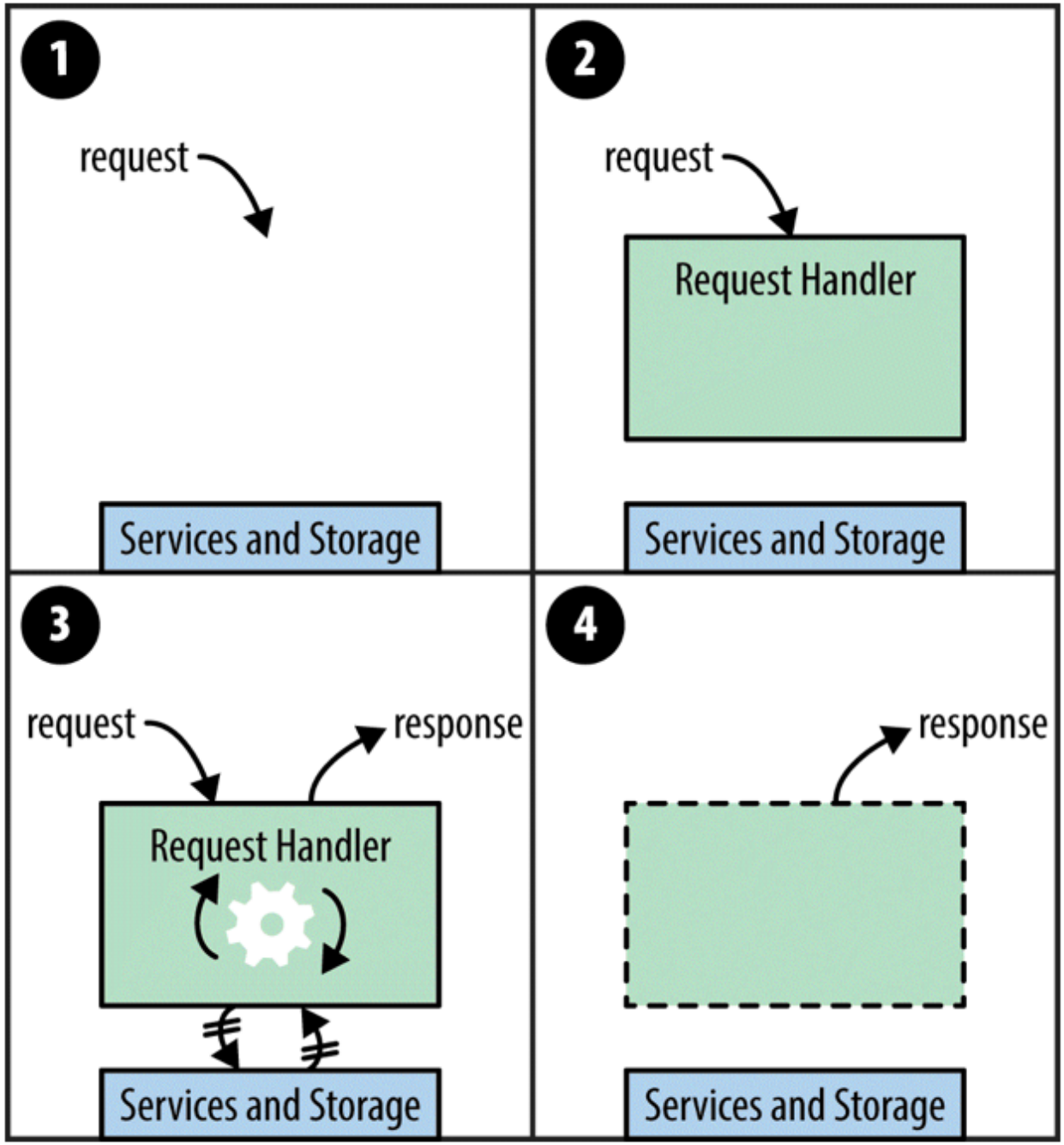
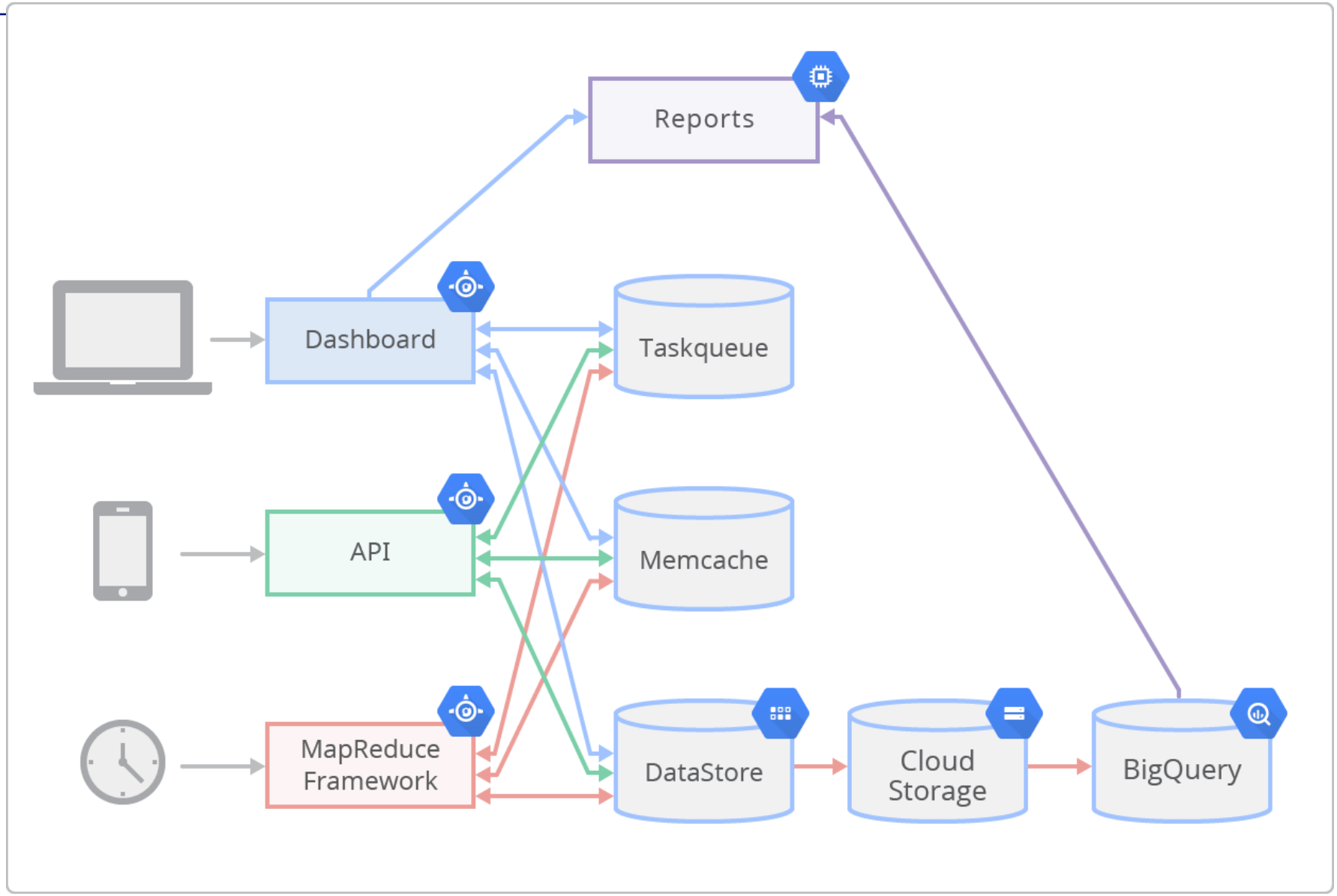


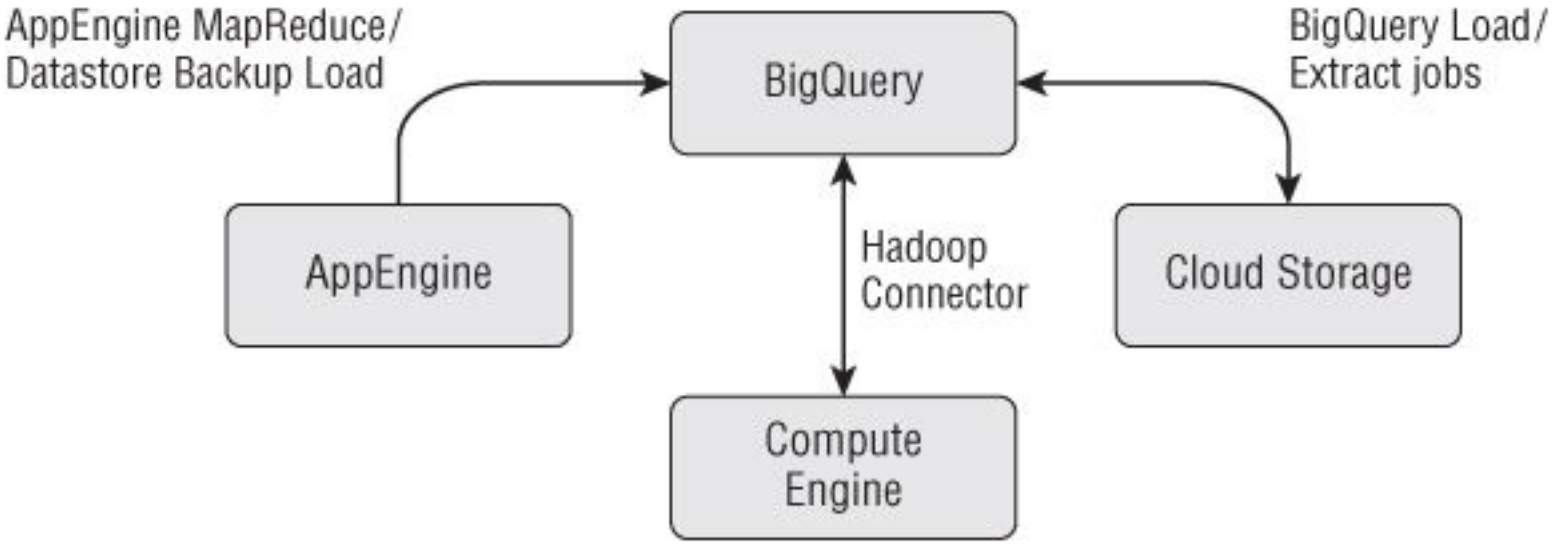

*Figure 4-1. Request handlers in the abstract: 1. A request arrives; 2. A request handler is created; 3. The request handler calls services and computes the response; 4. The request handler terminates, the*

# Sample Application on Google App Engine



# Google Compute Engine

- ..

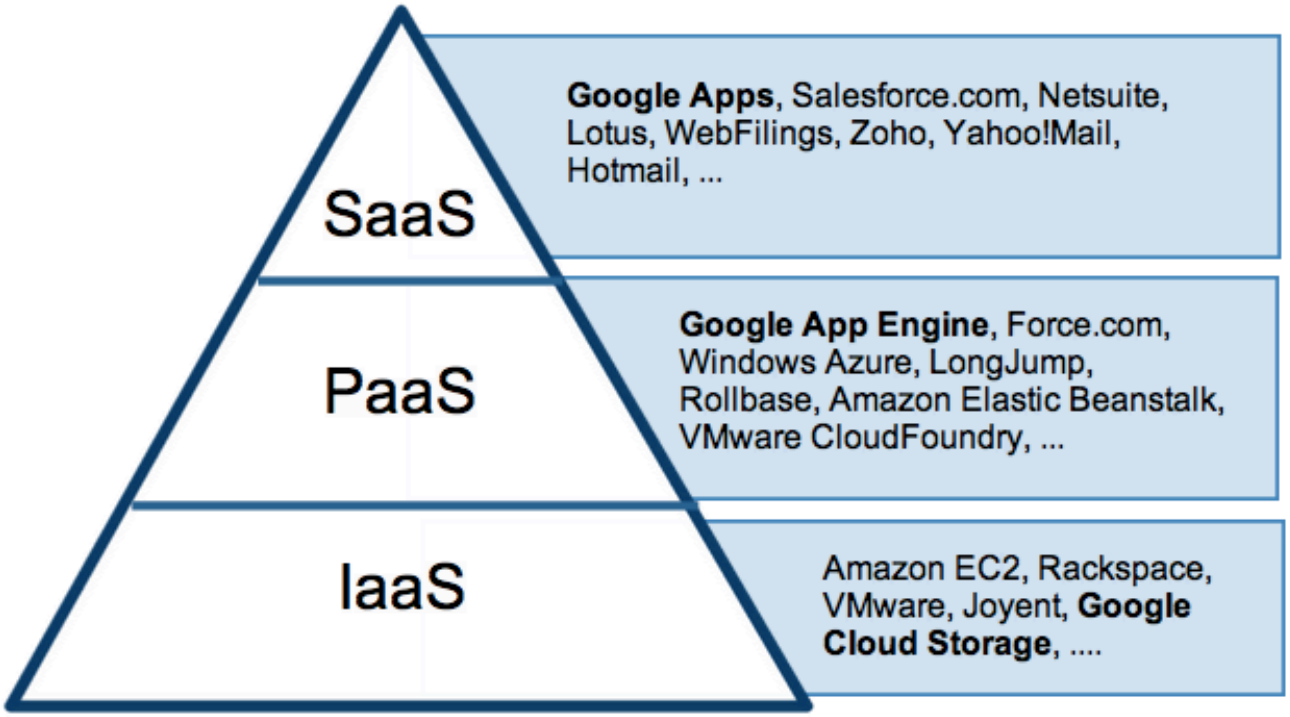


# Books

- ..


Build & Run Scalable Web Apps on Google's Infrastructure
2nd Edition
Programming
Google App Engine
O'REILLY | Google Press
Dan Sanderson

# Cloud Computing as Gartner Sees It



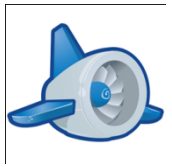

Source: Gartner AADI Summit Dec 2009

## IaaS

Services such as Amazon EC2 are primarily positioned as **IaaS** (*Infrastructure as a Service*). In this model, the user is given low-level resource management mechanisms (VM provisioning, distributed storage, etc.) and it is up to them to develop their application stack (operating system, HTTP server, MVC framework, database, etc.). While this provides extreme flexibility for the user, it also means that they must manage scaling and fault tolerance on their own.

# PaaS

- **Amazon AWS: EC2 & S3 (among the ma infrastructure services)**
  - Linux machine
  - Windows machine
  - A three-tier enterprise application
- **Google app Engine**
  - Eclipse plug-in for GAE
  - Development and deployment of an application
- **Windows Azure**
  - Storage: blob store/container
  - MS Visual Studio Azure development and production environment

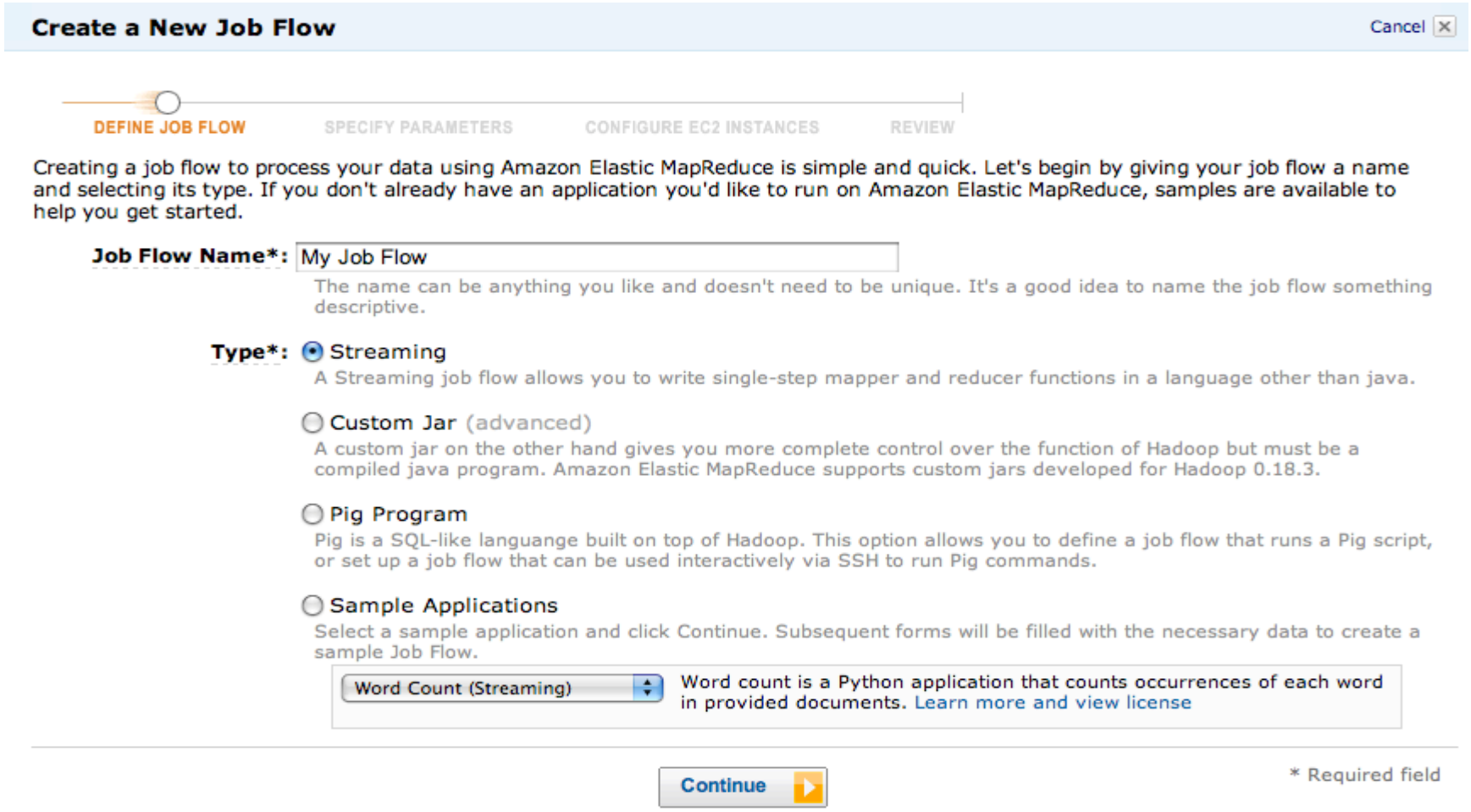


# Amazon Elastic MapReduce (IaaS)

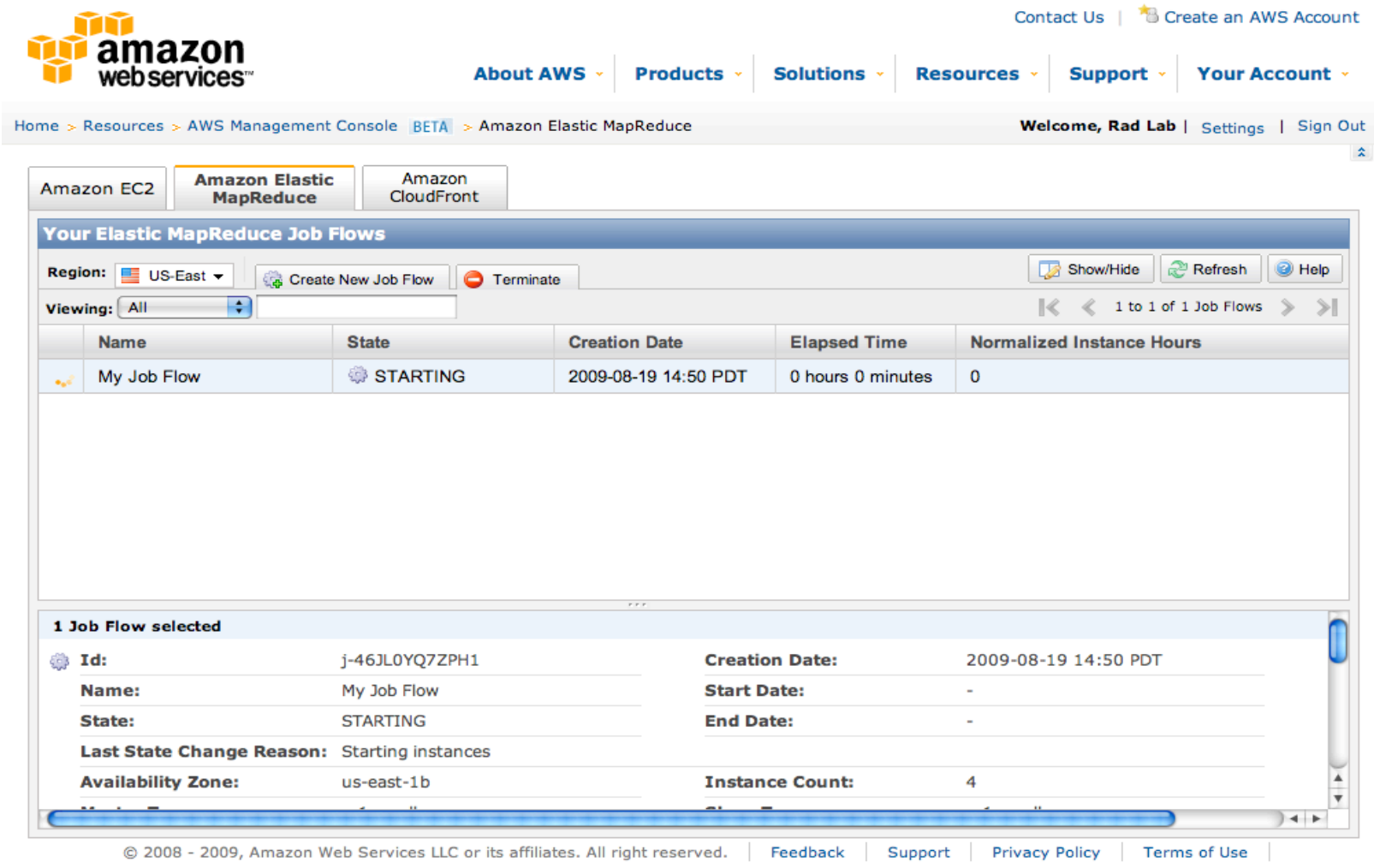
- Web interface and command-line tools for running Hadoop jobs on EC2
- Data stored in Amazon S3
- Monitors job and shuts machines after use

# Elastic MapReduce UI

## Cluster of computers



# Elastic MapReduce UI



# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

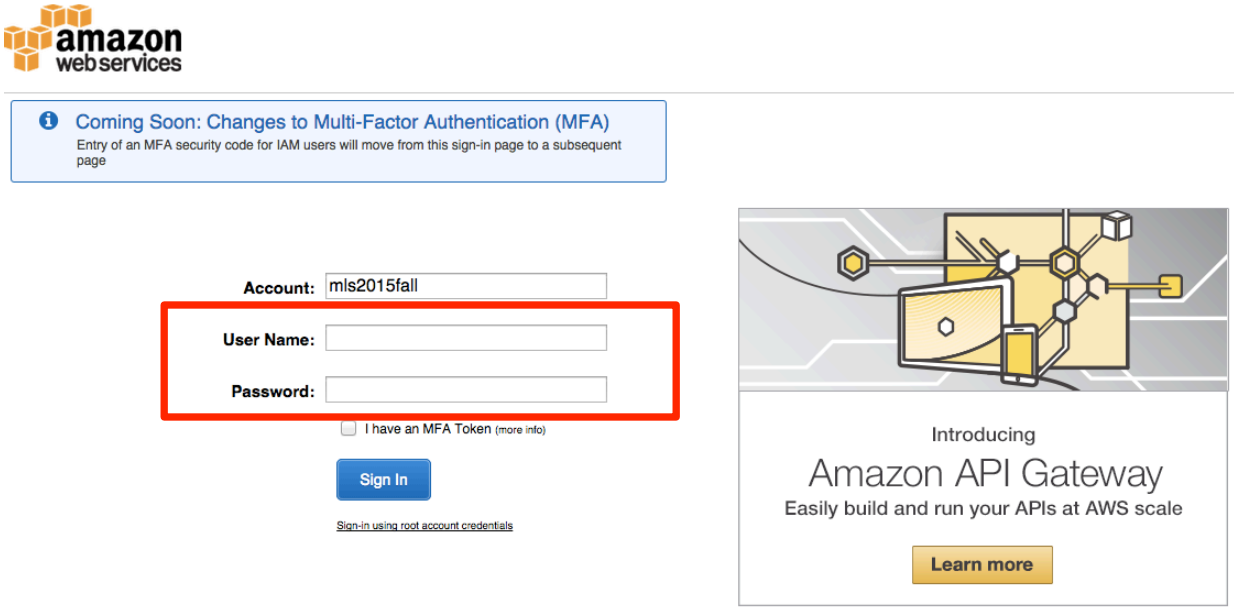
- **Getting access to the cloud**

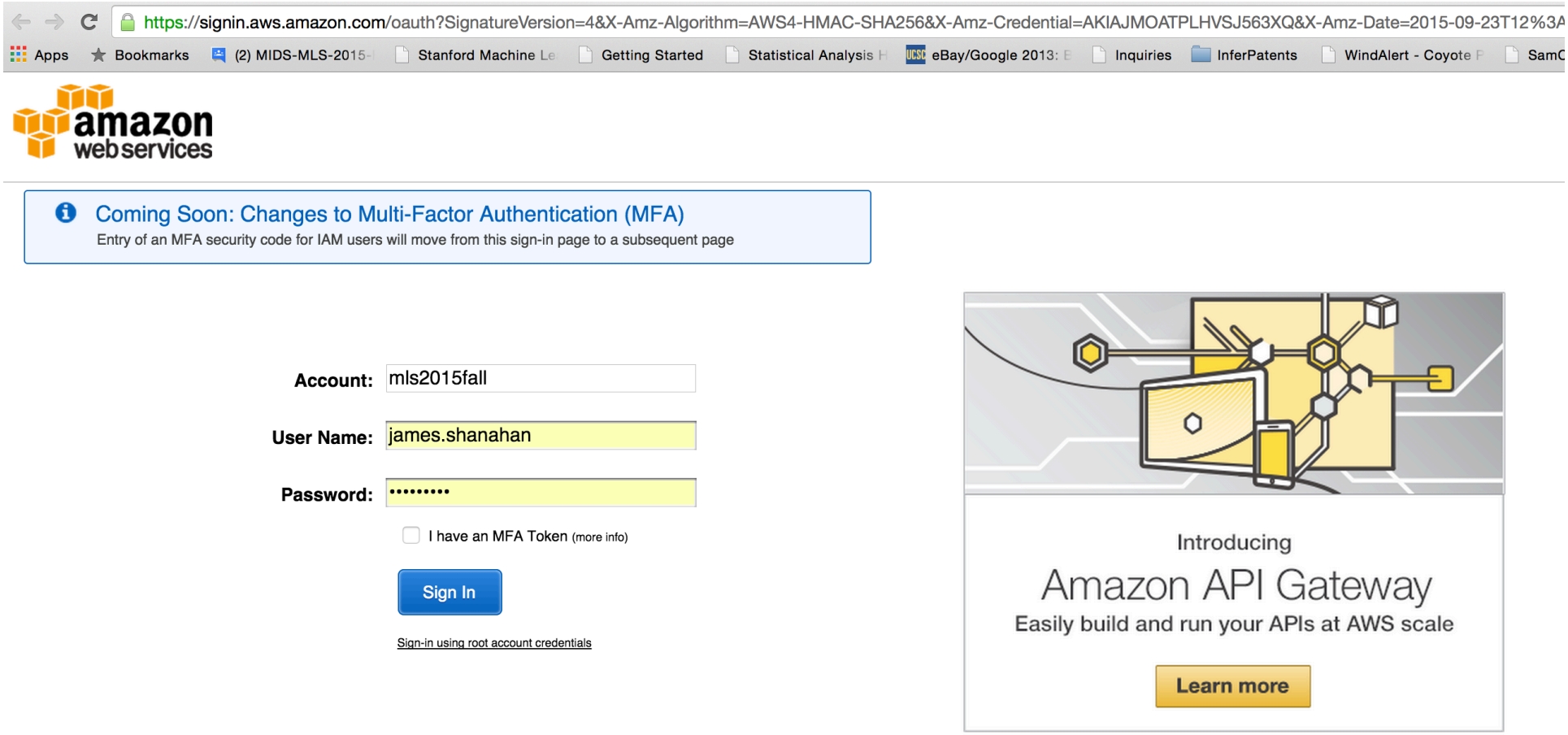
# Login

- **Link:**

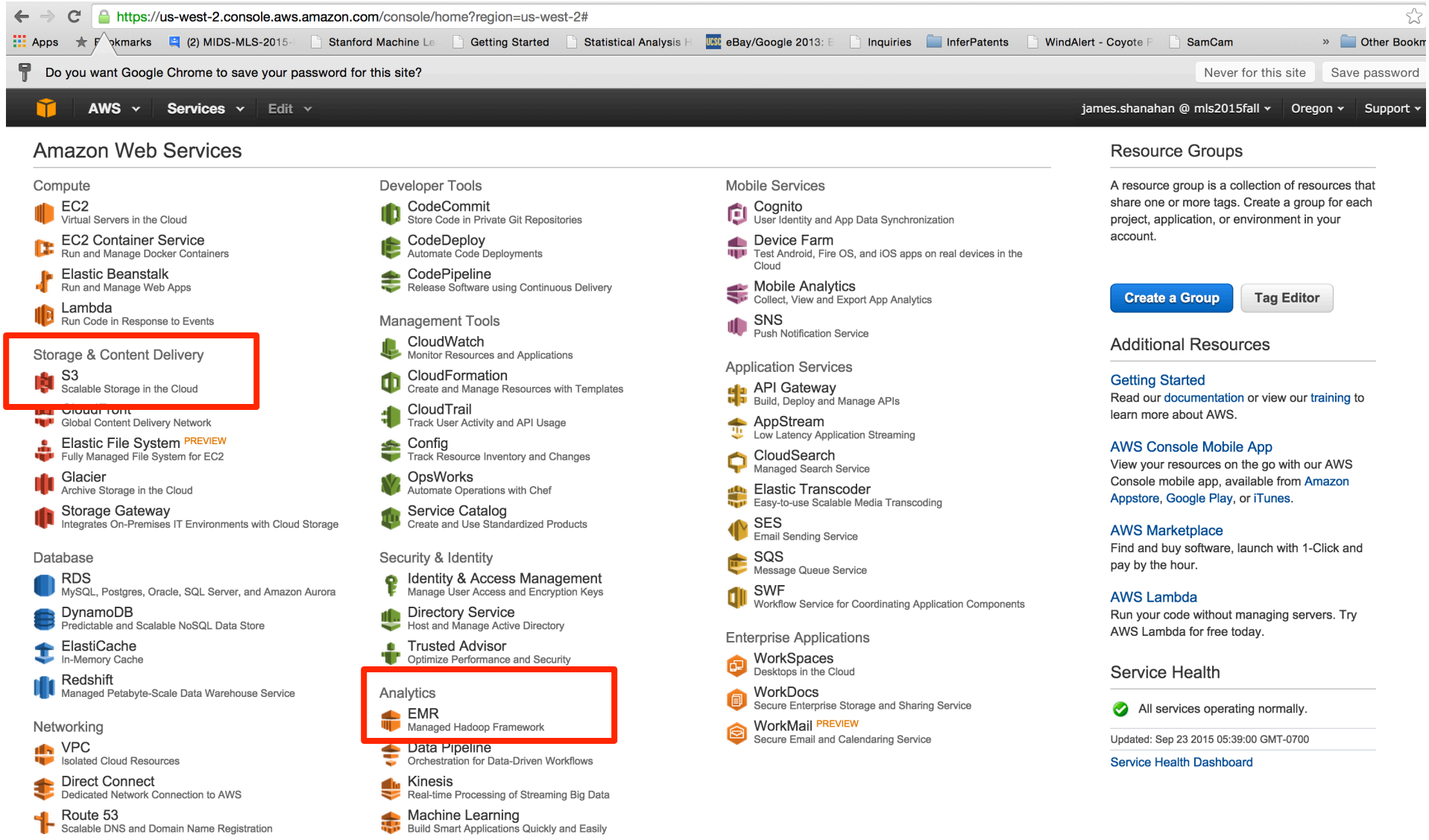
https://mls2015fall.signin.aws.amazon.com/console

- **Username: FirstName.LastName e.g. Liang.Dai**
- **Password: FirstName.LastName e.g. Liang.Dai**




https://signin.aws.amazon.com/oauth?SignatureVersion=4&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAJMOATPLHVSJ563XQ&X-Amz-Date=2015-09-23T12%3A
Apps
Bookmarks
(2) MIDS-MLS-2015-
Stanford Machine Le
Getting Started
Statistical Analysis H
eBay/Google 2013: E
Inquiries
InferPatents
WindAlert - Coyote F
SamC
amazon web services
Coming Soon: Changes to Multi-Factor Authentication (MFA)
Entry of an MFA security code for IAM users will move from this sign-in page to a subsequent page
Account:
mls2015fall
User Name:
james.shanahan
Password:
I have an MFA Token (more info)
Sign In
Sign-in using root account credentials
Introducing
Amazon API Gateway
Easily build and run your APIs at AWS scale
Learn more

# AWS: For now: S3 storage and EMR clusters



# Pricing by the hour of compute time

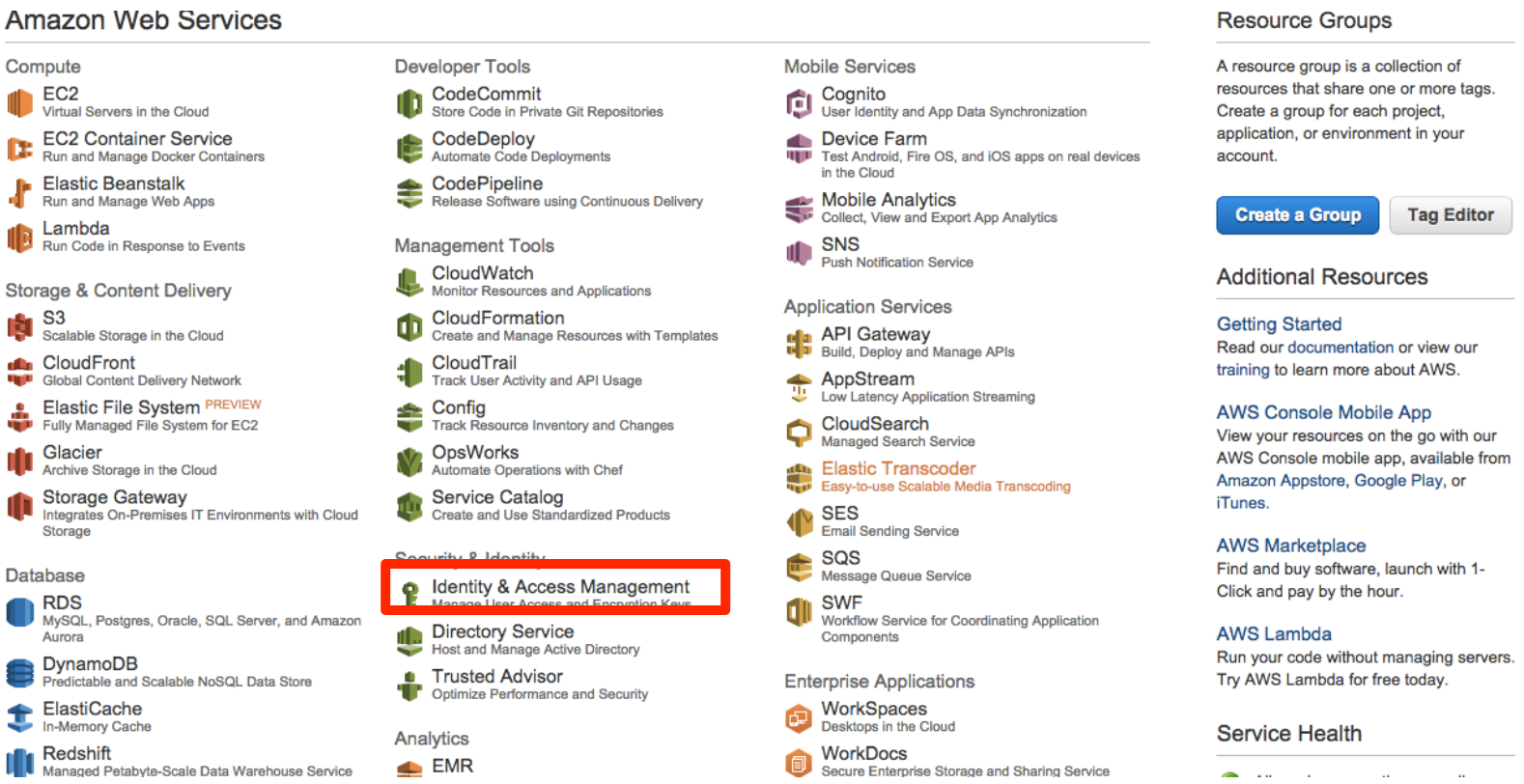
- **Pricing by the hour of compute time**
- **Pricing by the**
  - Megabyte of storage
  - Network transfer

- **So please be respectful and shutdown clusters when not in use!**

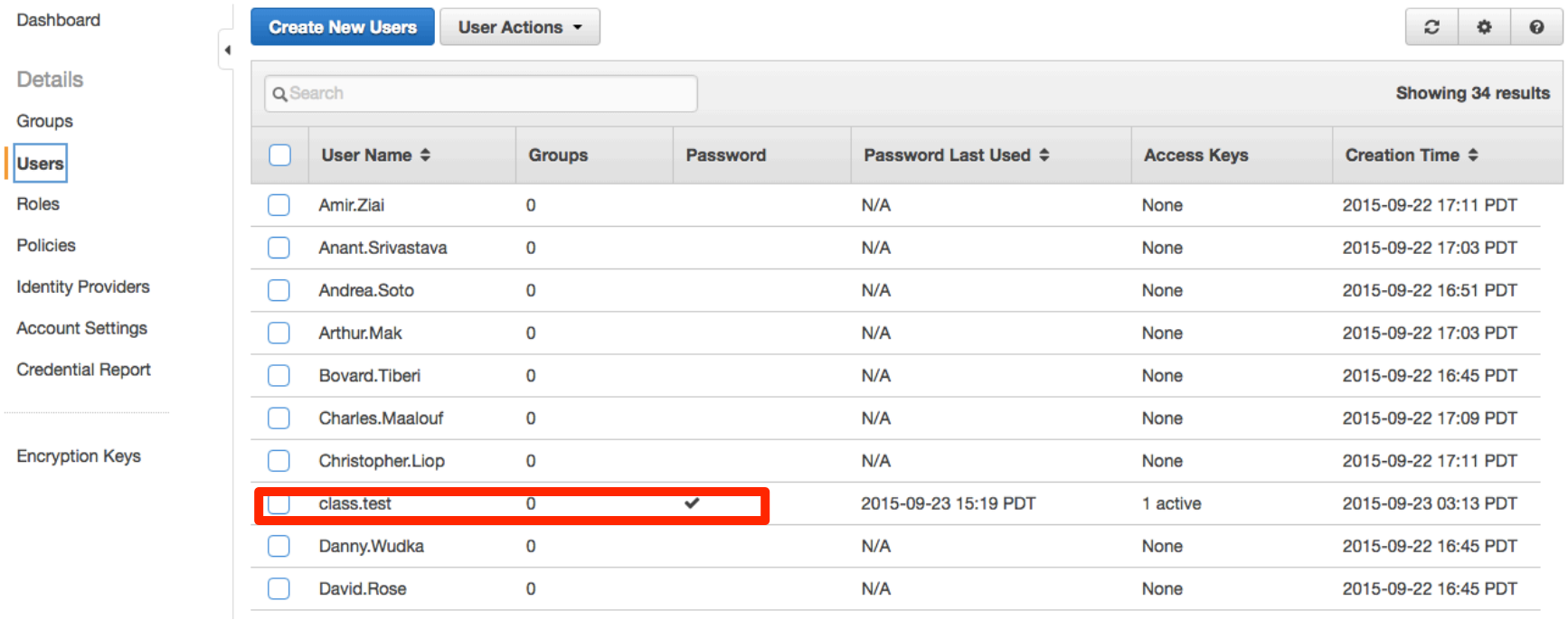
# Access Keys: programmatic calls to AWS

## *Access Key will be sent to you*

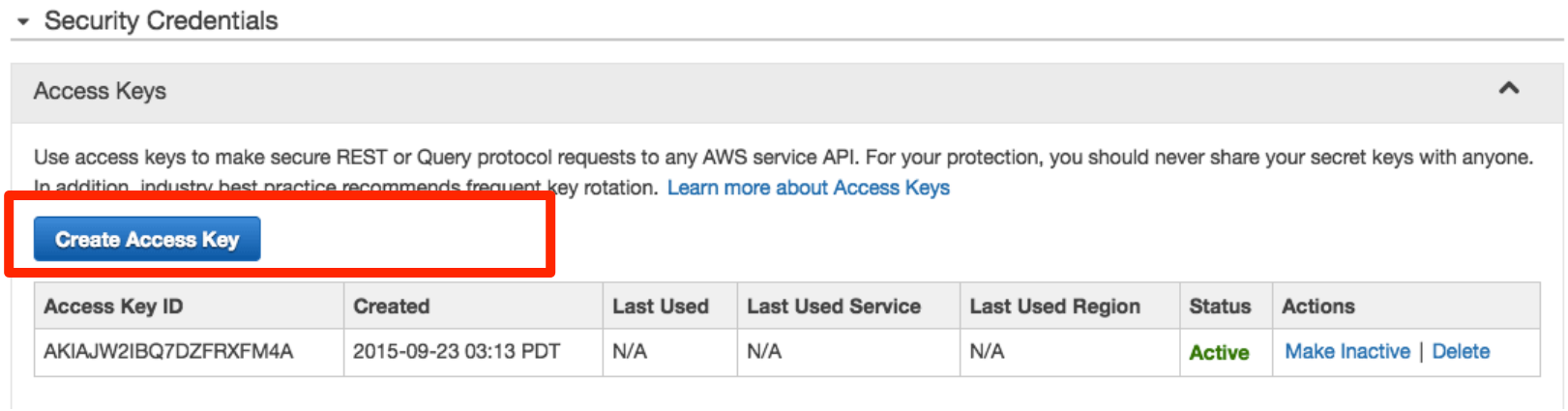
- **Users need their own access keys to make programmatic calls to AWS from the AWS Command Line Interface (AWS CLI), Tools for Windows PowerShell, the AWS SDKs, or direct HTTP calls using the APIs for individual AWS services.**
- **Use access keys to make secure REST or Query protocol requests to any AWS service API.**
- **For your protection, you should never share your secret keys with anyone.**

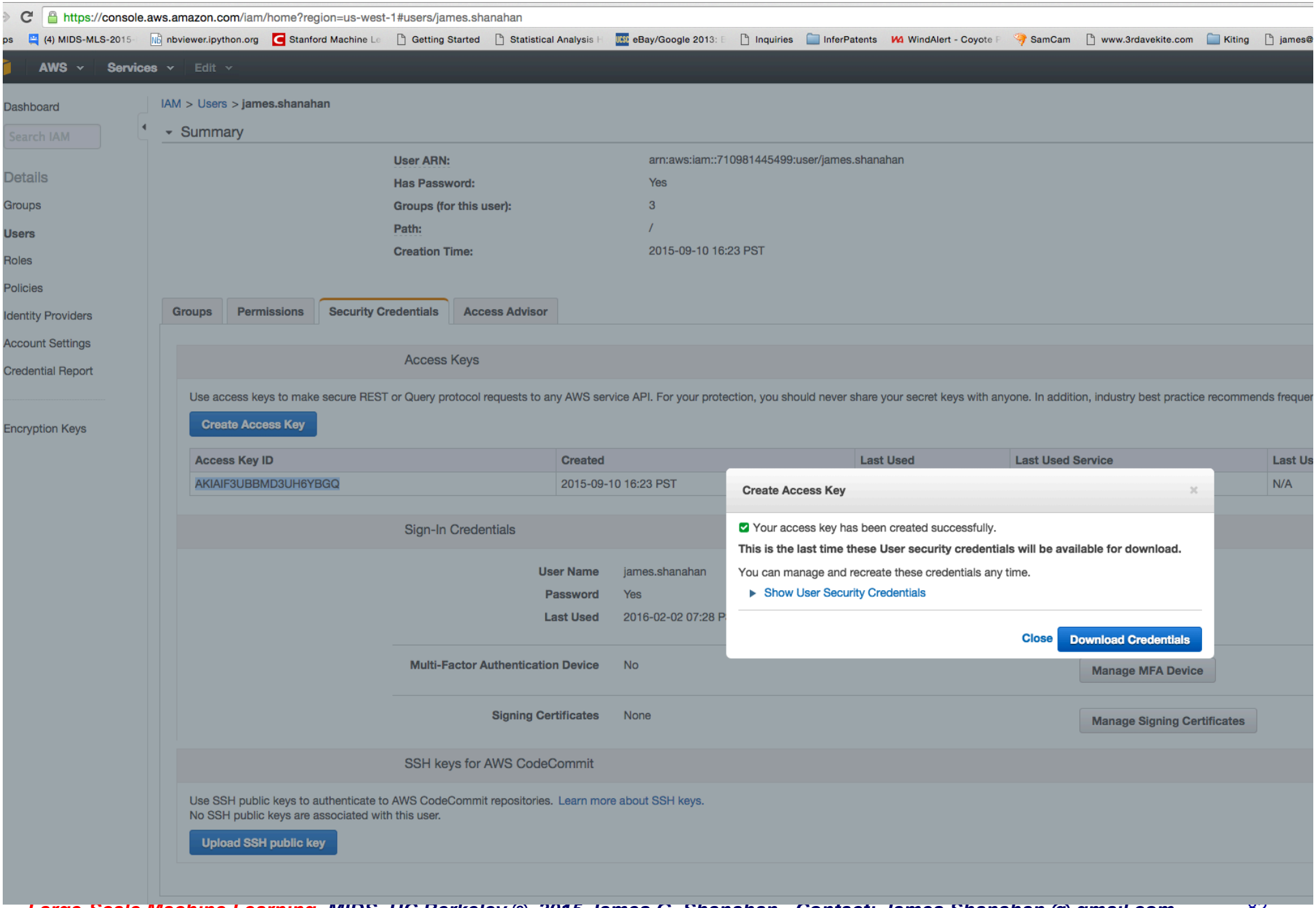

Amazon Web Services
Compute
EC2
Virtual Servers in the Cloud
EC2 Container Service
Run and Manage Docker Containers
Elastic Beanstalk
Run and Manage Web Apps
Lambda
Run Code in Response to Events
Storage & Content Delivery
S3
Scalable Storage in the Cloud
CloudFront
Global Content Delivery Network
Elastic File System PREVIEW
Fully Managed File System for EC2
Glacier
Archive Storage in the Cloud
Storage Gateway
Integrates On-Premises IT Environments with Cloud Storage
Database
RDS
MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora
DynamoDB
Predictable and Scalable NoSQL Data Store
ElastiCache
In-Memory Cache
Redshift
Managed Petabyte-Scale Data Warehouse Service
Developer Tools
CodeCommit
Store Code in Private Git Repositories
CodeDeploy
Automate Code Deployments
CodePipeline
Release Software using Continuous Delivery
Management Tools
CloudWatch
Monitor Resources and Applications
CloudFormation
Create and Manage Resources with Templates
CloudTrail
Track User Activity and API Usage
Config
Track Resource Inventory and Changes
OpsWorks
Automate Operations with Chef
Service Catalog
Create and Use Standardized Products
Identity & Access Management
Directory Service
Host and Manage Active Directory
Trusted Advisor
Optimize Performance and Security
Analytics
EMR
Mobile Services
Cognito
User Identity and App Data Synchronization
Device Farm
Test Android, Fire OS, and iOS apps on real devices in the Cloud
Mobile Analytics
Collect, View and Export App Analytics
SNS
Push Notification Service
Application Services
API Gateway
Build, Deploy and Manage APIs
AppStream
Low Latency Application Streaming
CloudSearch
Managed Search Service
Elastic Transcoder
Easy-to-use Scalable Media Transcoding
SES
Email Sending Service
SQS
Message Queue Service
SWF
Workflow Service for Coordinating Application Components
Enterprise Applications
WorkSpaces
Desktops in the Cloud
WorkDocs
Secure Enterprise Storage and Sharing Service
Resource Groups
A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.
Create a Group
Tag Editor
Additional Resources
Getting Started
Read our documentation or view our training to learn more about AWS.
AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.
AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.
AWS Lambda
Run your code without managing servers. Try AWS Lambda for free today.
Service Health

# Click “User” and select your  click your username

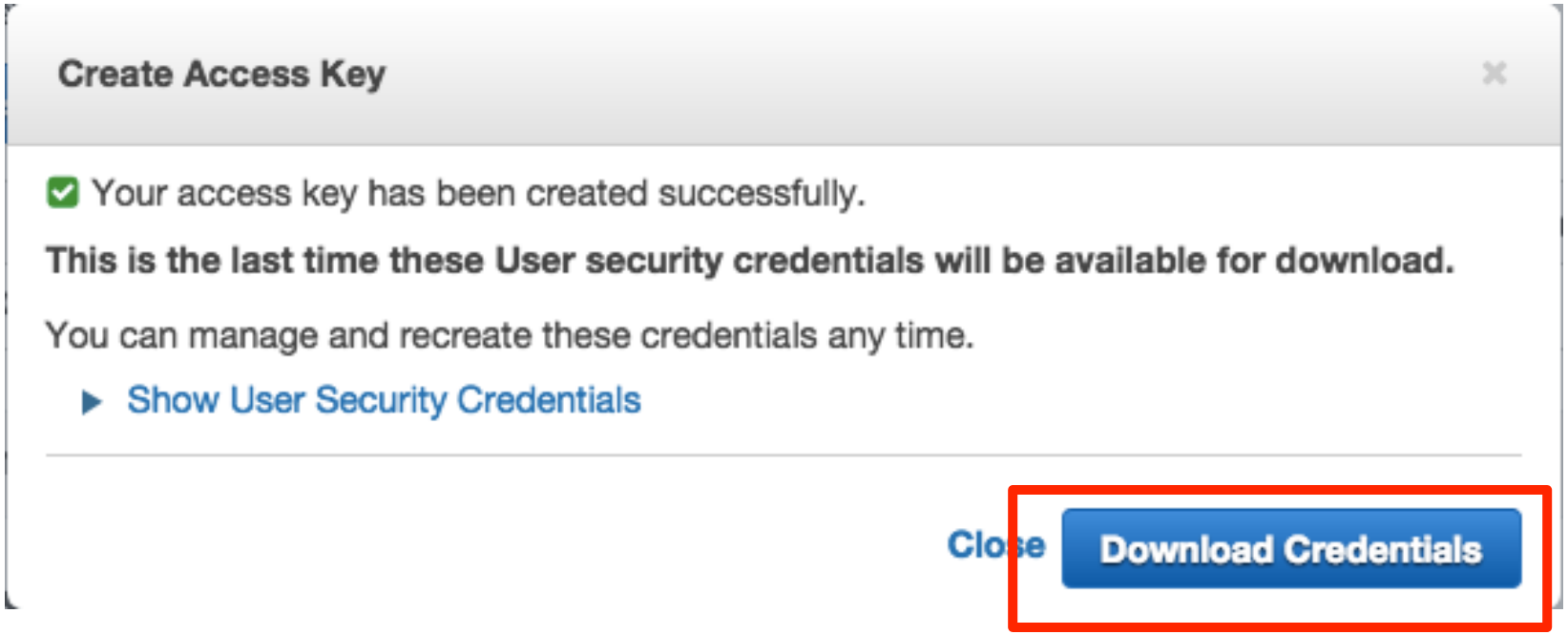


# Create access key under “Security Credentials”



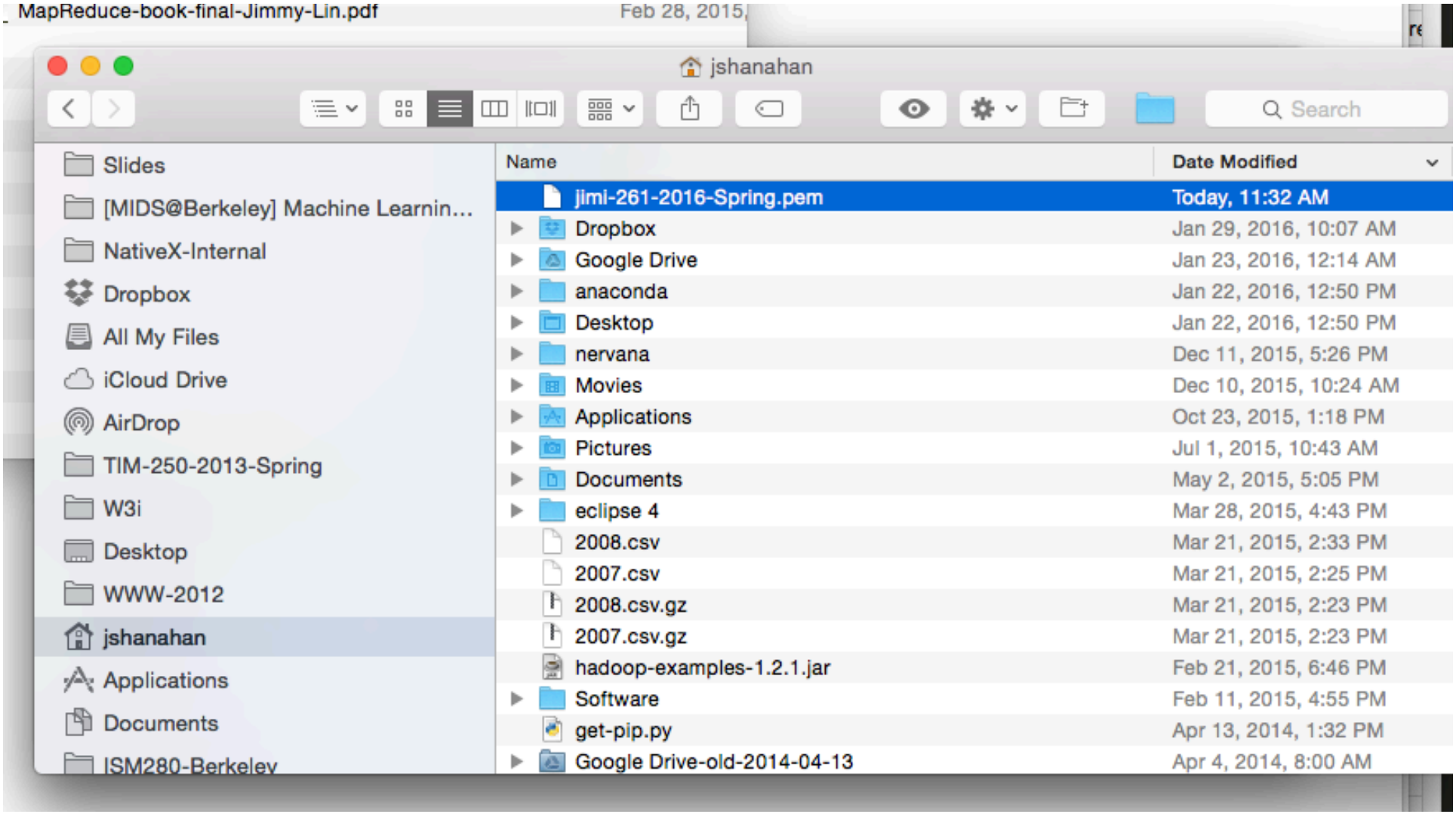

https://console.aws.amazon.com/iam/home?region=us-west-1#users/james.shanahan
AWS
Services
Edit
Dashboard
Search IAM
Details
Groups
Users
Roles
Policies
Identity Providers
Account Settings
Credential Report
Encryption Keys
IAM > Users > james.shanahan
Summary
User ARN: arn:aws:iam::710981445499:user/james.shanahan
Has Password: Yes
Groups (for this user): 3
Path: /
Creation Time: 2015-09-10 16:23 PST
Groups
Permissions
Security Credentials
Access Advisor
Access Keys
Use access keys to make secure REST or Query protocol requests to any AWS service API. For your protection, you should never share your secret keys with anyone. In addition, industry best practice recommends frequer
Create Access Key
Access Key ID
Created
Last Used
Last Used Service
AKIAIF3UBBMD3UH6YBGQ
2015-09-10 16:23 PST
N/A
Create Access Key
Your access key has been created successfully.
This is the last time these User security credentials will be available for download.
You can manage and recreate these credentials any time.
Show User Security Credentials
Close
Download Credentials
Sign-In Credentials
User Name james.shanahan
Password Yes
Last Used 2016-02-02 07:28 P
Multi-Factor Authentication Device No
Manage MFA Device
Signing Certificates None
Manage Signing Certificates
SSH keys for AWS CodeCommit
Use SSH public keys to authenticate to AWS CodeCommit repositories. Learn more about SSH keys.
No SSH public keys are associated with this user.
Upload SSH public key

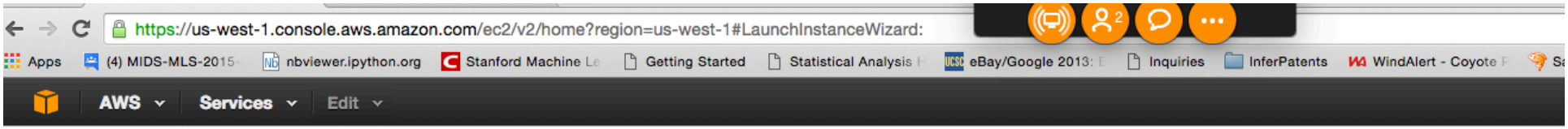
# Click download your credentials



# Open CSV to get your keys

| User Name | Access Key Ic | Secret Access Key | | |
|---|---|---|---|---|
| class.test | AKIAJVNWF7 | BVCWaJqmgyRuHlcGu0lKJF4hl3gy0OM436pJlAbD | | |


MapReduce-book-final-Jimmy-Lin.pdf
Feb 28, 2015
jshanahan
Search
Slides
[MIDS@Berkeley] Machine Learnin...
NativeX-Internal
Dropbox
All My Files
iCloud Drive
AirDrop
TIM-250-2013-Spring
W3i
Desktop
WWW-2012
jshanahan
Applications
Documents
ISM280-Berkeley
Name
Date Modified
jimi-261-2016-Spring.pem
Today, 11:32 AM
Dropbox
Jan 29, 2016, 10:07 AM
Google Drive
Jan 23, 2016, 12:14 AM
anaconda
Jan 22, 2016, 12:50 PM
Desktop
Jan 22, 2016, 12:50 PM
nervana
Dec 11, 2015, 5:26 PM
Movies
Dec 10, 2015, 10:24 AM
Applications
Oct 23, 2015, 1:18 PM
Pictures
Jul 1, 2015, 10:43 AM
Documents
May 2, 2015, 5:05 PM
eclipse 4
Mar 28, 2015, 4:43 PM
2008.csv
Mar 21, 2015, 2:33 PM
2007.csv
Mar 21, 2015, 2:25 PM
2008.csv.gz
Mar 21, 2015, 2:23 PM
2007.csv.gz
Mar 21, 2015, 2:23 PM
hadoop-examples-1.2.1.jar
Feb 21, 2015, 6:46 PM
Software
Feb 11, 2015, 4:55 PM
get-pip.py
Apr 13, 2014, 1:32 PM
Google Drive-old-2014-04-13
Apr 4, 2014, 8:00 AM




## Launch Status

**Your instances are now launching**

The following instance launches have been initiated: i-cf7a657d View launch log

**Get notified of estimated charges**

Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier).

### How to connect to your instances

Your instances are launching, and it may take a few minutes until they are in the **running** state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to

Click **View Instances** to monitor your instances' status. Once your instances are in the **running** state, you can **connect** to them from the Instances screen. Find out how to connect to your instances.

### Here are some helpful resources to get you started

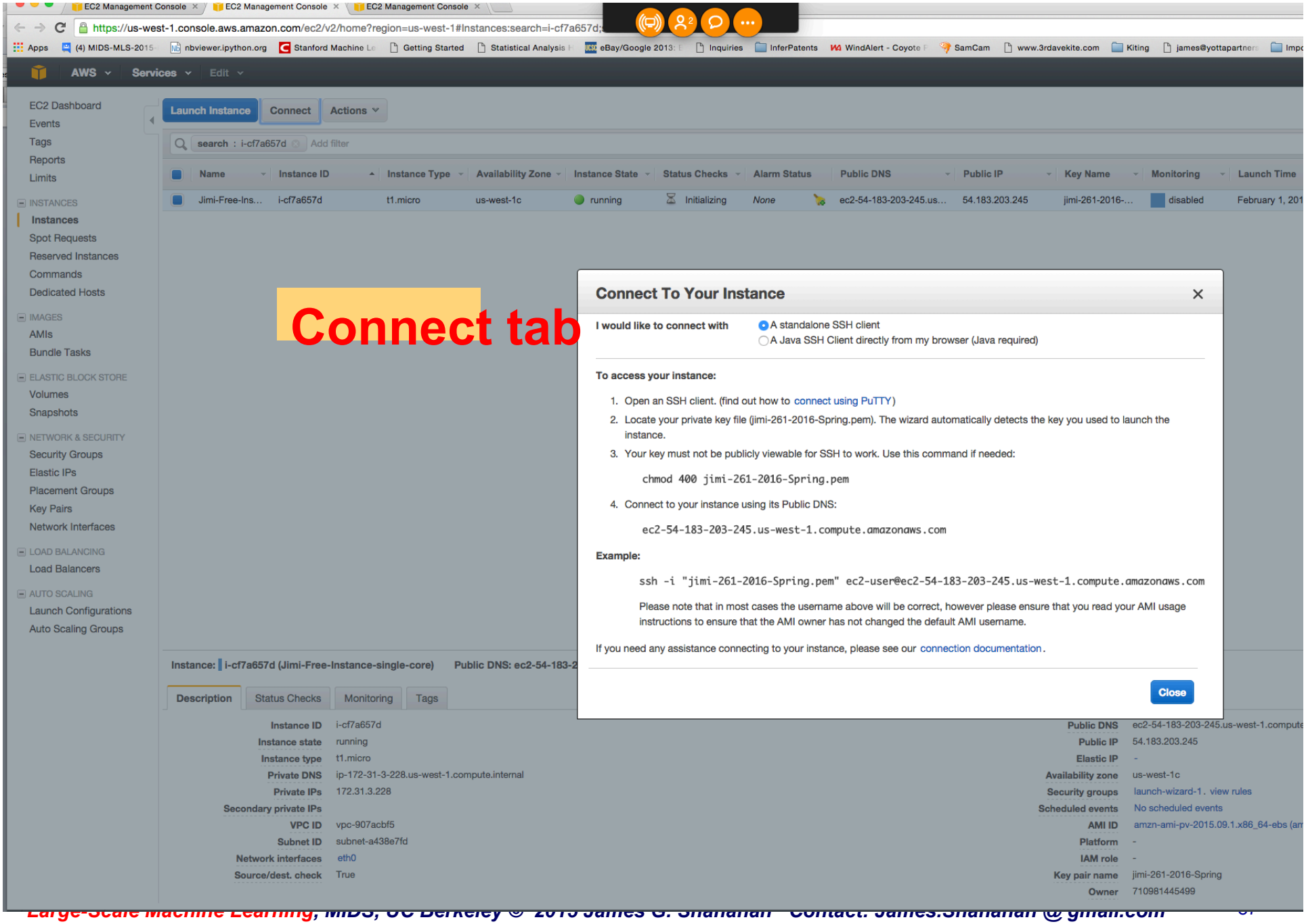
- How to connect to your Linux instance
- Learn about AWS Free Usage Tier
- Amazon EC2: User Guide
- Amazon EC2: Discussion Forum

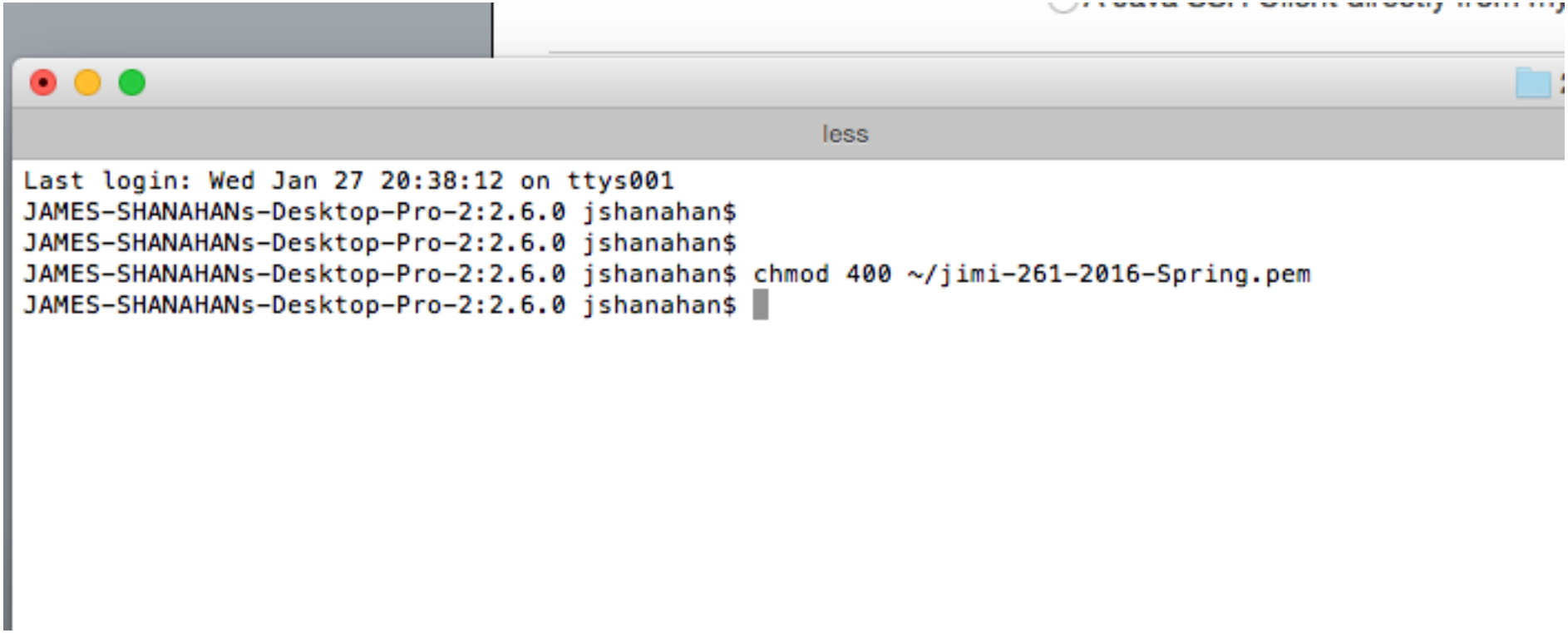
While your instances are launching you can also

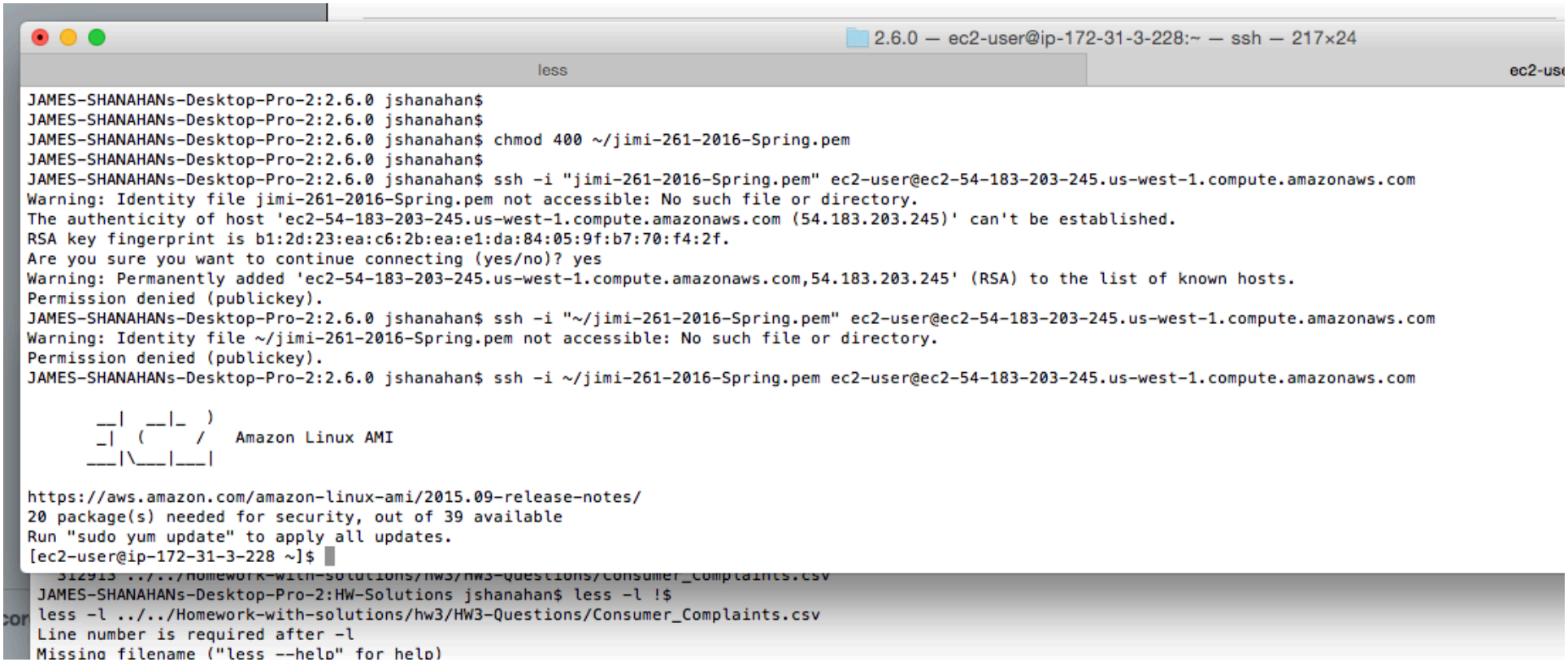
Create status check alarms to be notified when these instances fail status checks. (Additional charges may apply)

Create and attach additional EBS volumes (Additional charges may apply)

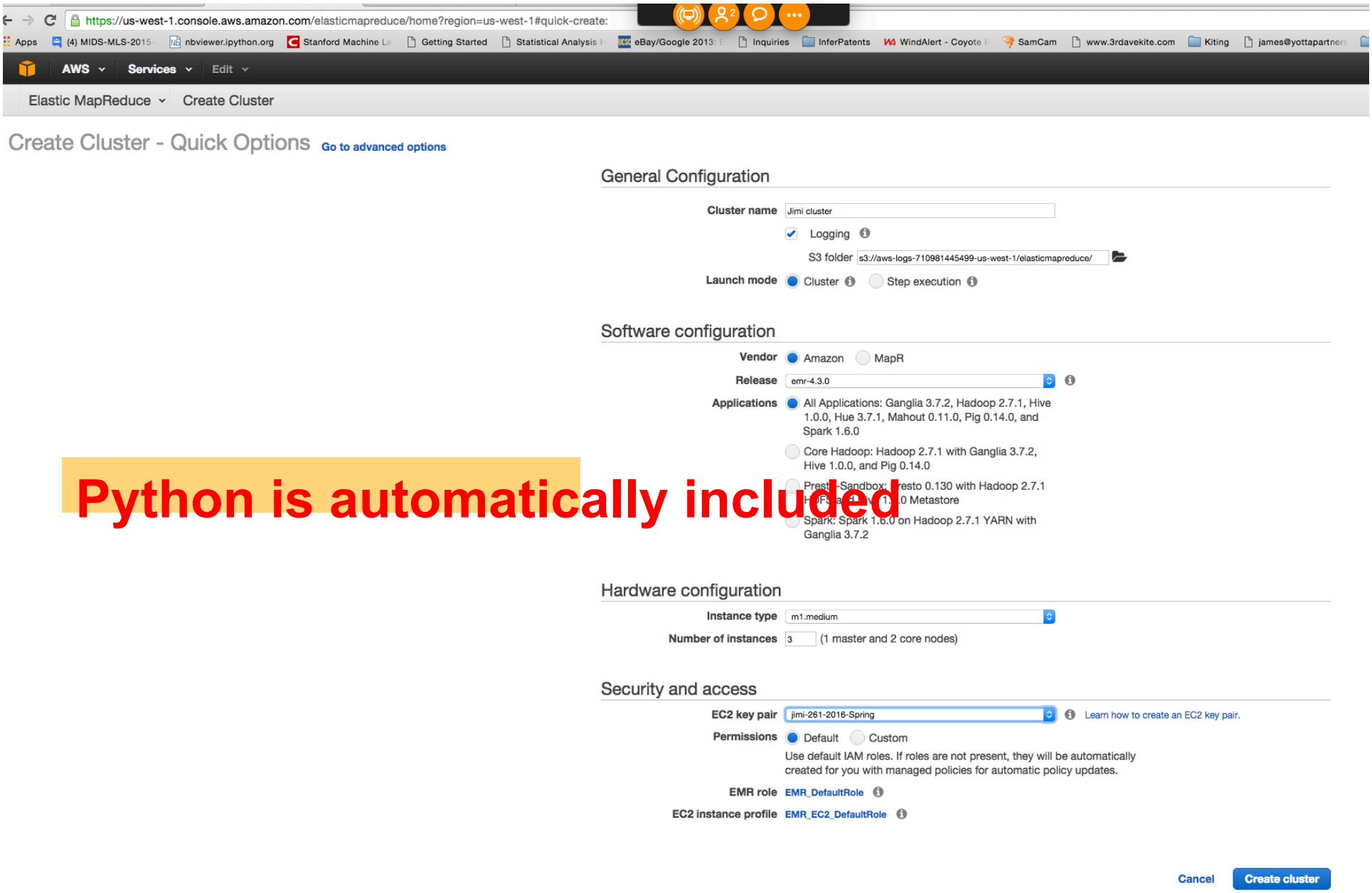
Manage security groups


EC2 Management Console
https://us-west-1.console.aws.amazon.com/ec2/v2/home?region=us-west-1#Instances:search=i-cf7a657d;
AWS
Services
Edit
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
Commands
Dedicated Hosts
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
LOAD BALANCING
Load Balancers
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
search : i-cf7a657d
Add filter
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS
Public IP
Key Name
Monitoring
Launch Time
Jimi-Free-Ins...
i-cf7a657d
t1.micro
us-west-1c
running
Initializing
None
ec2-54-183-203-245.us...
54.183.203.245
jimi-261-2016-...
disabled
February 1, 201
Connect tab
Connect To Your Instance
I would like to connect with
A standalone SSH client
A Java SSH Client directly from my browser (Java required)
To access your instance:
1. Open an SSH client. (find out how to connect using PuTTY)
2. Locate your private key file (jimi-261-2016-Spring.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:
chmod 400 jimi-261-2016-Spring.pem
4. Connect to your instance using its Public DNS:
ec2-54-183-203-245.us-west-1.compute.amazonaws.com
Example:
ssh -i "jimi-261-2016-Spring.pem" ec2-user@ec2-54-183-203-245.us-west-1.compute.amazonaws.com
Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.
If you need any assistance connecting to your instance, please see our connection documentation.
Close
Instance: i-cf7a657d (Jimi-Free-Instance-single-core)
Public DNS: ec2-54-183-2
Description
Status Checks
Monitoring
Tags
Instance ID i-cf7a657d
Instance state running
Instance type t1.micro
Private DNS ip-172-31-3-228.us-west-1.compute.internal
Private IPs 172.31.3.228
Secondary private IPs
VPC ID vpc-907acbf5
Subnet ID subnet-a438e7fd
Network interfaces eth0
Source/dest. check True
Public DNS ec2-54-183-203-245.us-west-1.compute
Public IP 54.183.203.245
Elastic IP -
Availability zone us-west-1c
Security groups launch-wizard-1. view rules
Scheduled events No scheduled events
AMI ID amzn-ami-pv-2015.09.1.x86_64-ebs (am
Platform -
IAM role -
Key pair name jimi-261-2016-Spring
Owner 710981445499

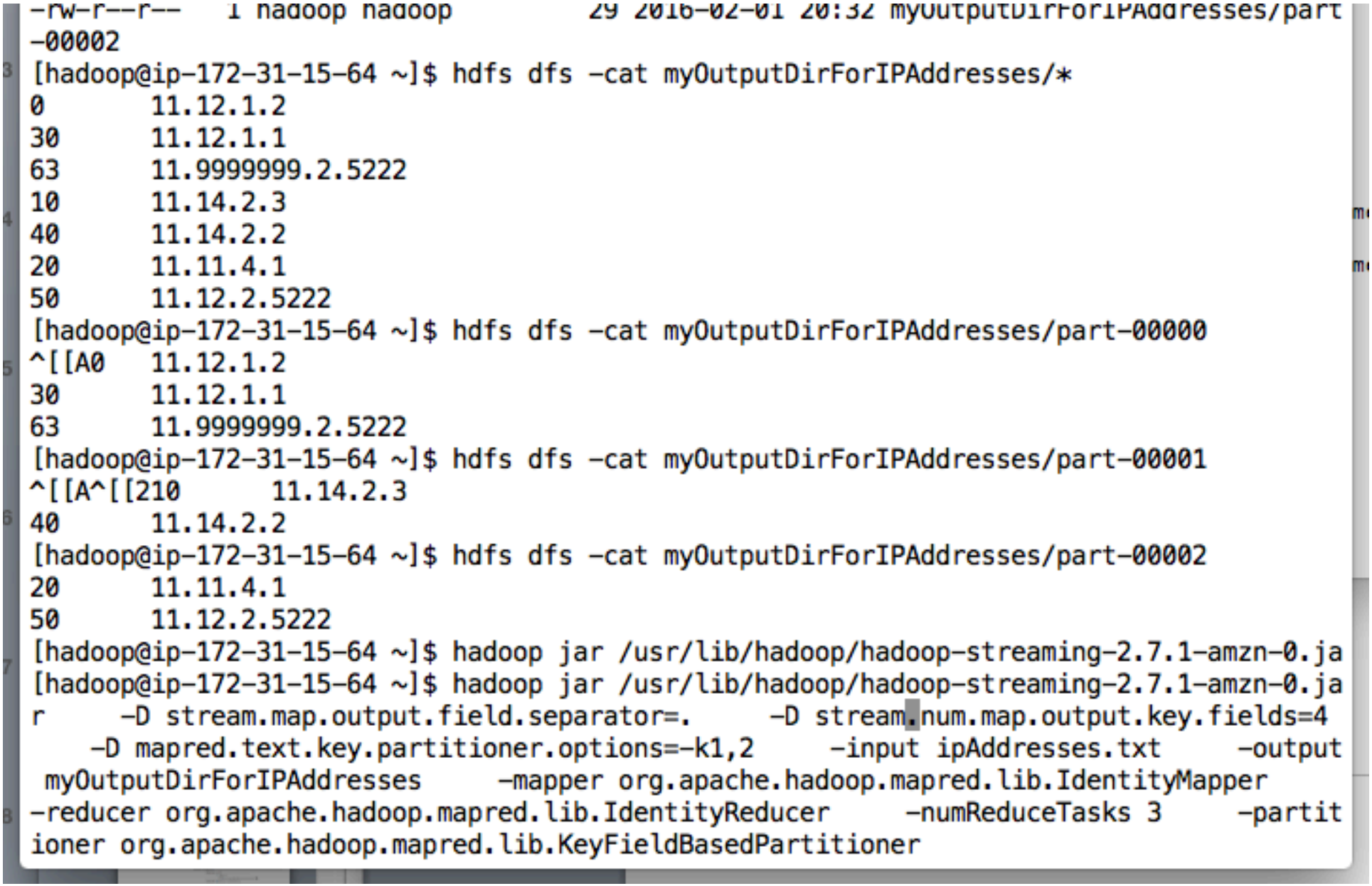

less
Last login: Wed Jan 27 20:38:12 on ttys001
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$ chmod 400 ~/jimi-261-2016-Spring.pem
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$


2.6.0 — ec2-user@ip-172-31-3-228:~ — ssh — 217×24
less
ec2-us
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$ chmod 400 ~/jimi-261-2016-Spring.pem
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$ ssh -i "jimi-261-2016-Spring.pem" ec2-user@ec2-54-183-203-245.us-west-1.compute.amazonaws.com
Warning: Identity file jimi-261-2016-Spring.pem not accessible: No such file or directory.
The authenticity of host 'ec2-54-183-203-245.us-west-1.compute.amazonaws.com (54.183.203.245)' can't be established.
RSA key fingerprint is b1:2d:23:ea:c6:2b:ea:e1:da:84:05:9f:b7:70:f4:2f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-183-203-245.us-west-1.compute.amazonaws.com,54.183.203.245' (RSA) to the list of known hosts.
Permission denied (publickey).
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$ ssh -i "~/jimi-261-2016-Spring.pem" ec2-user@ec2-54-183-203-245.us-west-1.compute.amazonaws.com
Warning: Identity file ~/jimi-261-2016-Spring.pem not accessible: No such file or directory.
Permission denied (publickey).
JAMES-SHANAHANs-Desktop-Pro-2:2.6.0 jshanahan$ ssh -i ~/jimi-261-2016-Spring.pem ec2-user@ec2-54-183-203-245.us-west-1.compute.amazonaws.com
__| __|_ )
_| ( / Amazon Linux AMI
___|___|___|
https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
20 package(s) needed for security, out of 39 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-3-228 ~]$
JAMES-SHANAHANs-Desktop-Pro-2:HW-Solutions jshanahan$ less -l !$
less -l ../../Homework-with-solutions/hw3/HW3-Questions/Consumer_Complaints.csv
Line number is required after -l
Missing filename ("less --help" for help)

- **Create an independent EMR Cluster**
  - Wont be doing this for a while
- **Boot up an EMR cluster via MrJob**
  - Can start this week!




**Python is automatically included**


-rw-r--r--  1 hadoop hadoop  29 2016-02-01 20:32 myOutputDirForIPAddresses/part-00002
[hadoop@ip-172-31-15-64 ~]$ hdfs dfs -cat myOutputDirForIPAddresses/*
0 11.12.1.2
30 11.12.1.1
63 11.9999999.2.5222
10 11.14.2.3
40 11.14.2.2
20 11.11.4.1
50 11.12.2.5222
[hadoop@ip-172-31-15-64 ~]$ hdfs dfs -cat myOutputDirForIPAddresses/part-00000
^[[A0 11.12.1.2
30 11.12.1.1
63 11.9999999.2.5222
[hadoop@ip-172-31-15-64 ~]$ hdfs dfs -cat myOutputDirForIPAddresses/part-00001
^[[A^[[210 11.14.2.3
40 11.14.2.2
[hadoop@ip-172-31-15-64 ~]$ hdfs dfs -cat myOutputDirForIPAddresses/part-00002
20 11.11.4.1
50 11.12.2.5222
[hadoop@ip-172-31-15-64 ~]$ hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.1-amzn-0.ja
[hadoop@ip-172-31-15-64 ~]$ hadoop jar /usr/lib/hadoop/hadoop-streaming-2.7.1-amzn-0.jar -D stream.map.output.field.separator=. -D stream.num.map.output.key.fields=4 -D mapred.text.key.partitioner.options=-k1,2 -input ipAddresses.txt -output myOutputDirForIPAddresses -mapper org.apache.hadoop.mapred.lib.IdentityMapper -reducer org.apache.hadoop.mapred.lib.IdentityReducer -numReduceTasks 3 -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

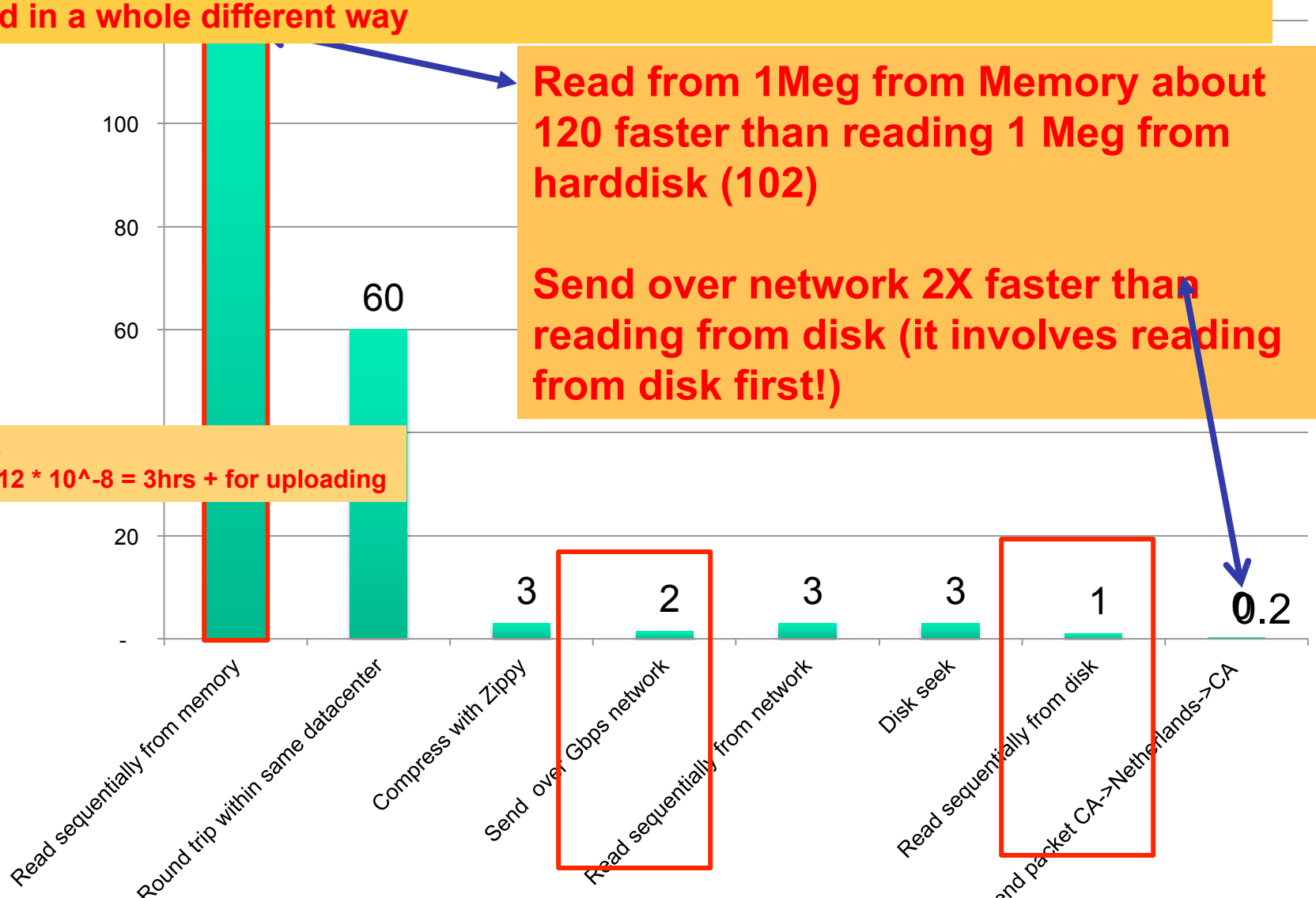
# Task vs 1M Read from Disk RAM vs. disk vs. bandwidth

**Cause us to think about memory, diskspace and network bandwidth carefully and in a whole different way**

**Read from 1Meg from Memory about 120 faster than reading 1 Meg from harddisk (102)**

**Send over network 2X faster than reading from disk (it involves reading from disk first!)**

**1TB**
**10^12 * 10^-8 = 3hrs + for uploading**



# The easiest ways to get started with Elastic MapReduce, #1

If you know Python, start with mrjob.

Here's a sample MRJob class, which you might write to wordcounter.py:

```
from mrjob.job import MRJob

class MRWordCounter(MRJob):
    def mapper(self, key, line):
        for word in line.split():
            yield word, 1

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

if __name__ == '__main__':
    MRWordCounter.run()
```

And how you run it:

```
export AWS_ACCESS_KEY_ID=...; export AWS_SECRET_ACCESS_KEY=...
python wordcounter.py -r emr < input > output
```

These samples, and much more documentation, is at http://packages.python.org/mrjob/writing-and-running.html.

# AWS Command Line Interface

- **The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.**

- **The AWS CLI introduces a new set of simple file commands for efficient file transfers to and from Amazon S3.**

# AWS Command Line Interface

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

The AWS CLI introduces a new set of simple file commands for efficient file transfers to and from Amazon S3.

Getting Started »

CLI Reference »

GitHub Project »

Community Forum »

**Windows**

Download and run the 64-bit or 32-bit Windows installer.

**Mac and Linux**

Requires Python 2.6.5 or higher.
Install using pip.

```
pip install awscli
```

**Amazon Linux**

The AWS CLI comes pre-installed on Amazon Linux AMI.

# Installing the AWS Command Line Interface

This section discusses various ways to install the AWS CLI.

> **Note**
>
> The AWS CLI comes pre-installed on the Amazon Linux AMI. Run `sudo yum update` after connecting to the instance to get the latest version of the package available via yum. If you need a more recent version of the AWS CLI than what is available in the Amazon updates repository, uninstall the package (`sudo yum remove aws-cli`) and then install using pip.

**Sections**



## Choose an Installation Method

There are a number of different ways to install the AWS CLI on your machine, depending on what operating system and environment you are using:

- **On Microsoft Windows** – use the MSI installer.
- **On Linux, OS X, or Unix** – use pip (a package manager for Python software) or install manually with the bundled installer.

> **Note**

# The easiest ways to get started with Elastic MapReduce, #2

If you know any other scripting language, start with the AWS elastic-mapreduce command line tool, or the AWS web console. Both are outlined at:

http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/

An example of the command line, assuming a file wordSplitter.py, is:

```
./elastic-mapreduce --create --stream \
    --mapper  s3://elasticmapreduce/samples/wordcount/wordSplitter.py \
    --input   s3://elasticmapreduce/samples/wordcount/input \
    --output  s3://mybucket.foo.com/wordcount \
    --reducer aggregate
```

A (not so) quick reference card for the command line tool is at:

http://s3.amazonaws.com/awsdocs/ElasticMapReduce/latest/emr-qrc.pdf

**http://s3.amazonaws.com/awsdocs/ElasticMapReduce/latest/emr-qrc.pdf**

## File Commands for Amazon S3

New file commands make it easy to manage your Amazon S3 objects. Using familiar syntax, you can view the contents of your S3 buckets in a directory-based listing.

```
$ aws s3 ls s3://mybucket
        LastWriteTime            Length Name
        -------------            ------ ----
                                PRE myfolder/
2013-09-03 10:00:00           1234 myfile.txt
```

You can perform recursive uploads and downloads of multiple files in a single folder-level command. The AWS CLI will run these transfers in parallel for increased performance.

```
$ aws s3 cp myfolder s3://mybucket/myfolder --recursive
upload: myfolder/file1.txt to s3://mybucket/myfolder/file1.txt
upload: myfolder/subfolder/file1.txt to s3://mybucket/myfolder/subfolder/file1.txt
```

A sync command makes it easy to synchronize the contents of a local folder with a copy in an S3 bucket.

```
$ aws s3 sync myfolder s3://mybucket/myfolder --exclude *.tmp
upload: myfolder/newfile.txt to s3://mybucket/myfolder/newfile.txt
```

- ..

```
: !aws s3 cp s3://ucb-mids-mls-jakewilliams/collectAndSortWordCounts/output/part-00000 ./sortedWordCounts.txt
download: s3://ucb-mids-mls-jakewilliams/collectAndSortWordCounts/output/part-00000 to ./sortedWordCounts.txt

: !head -25 sortedWordCounts.txt
```

# Run MRJob on Amazon's EMR clusters

- **Access data on Amazon's storage S3**

**Run the job with AWS EMR**

Note: With the above configuration in ~/.mrjob.conf, this job takes about 10 minutes (the input data is relatively small).

```
!./collectAndSortWordCounts.py s3://ucb-mids-mls-jakewilliams/count5gramWords/output/ -r emr \
    --output-dir=s3://ucb-mids-mls-jakewilliams/collectAndSortWordCounts/output \
    --no-output
```

```
using configs in /Users/jakerylandwilliams/.mrjob.conf
using existing scratch bucket mrjob-070799b65f5ef217
using s3://mrjob-070799b65f5ef217/tmp/ as our scratch dir on S3
```

# Download file to local machine

## Amazon's CLI tool runs on Windows/Mac/Linux

```
!aws s3 cp s3://ucb-mids-mls-jakewilliams/invertedIndex-2k/output/part-00000 ./invertedIndex-2k_part-00000.txt
!head -c 100 ./invertedIndex-2k_part-00000.txt
```

```
download: s3://ucb-mids-mls-jakewilliams/invertedIndex-2k/output/part-00000 to ./invertedIndex-2k_part-00000.txt
AIDS    {'limited': [0.0013619036908308604, 2119], 'all': [0.0017246551572799955, 6655], 'caused': [0.0
```

# Amazon Elastic MapReduce (API Version 2009-03-31)

Revised: 4/25/2011

Quick Reference Card (page 1)

### Create Job Flow

JAR $ ./elastic-mapreduce --create --alive –input *AmazonS3bucket* --output *AmazonS3bucket* --log-uri *AmazonS3bucket*
Stream $ ./elastic-mapreduce --create --alive --stream --input *AmazonS3bucket* --output *AmazonS3bucket* --log-uri *AmazonS3bucket*
Pig $ ./elastic-mapreduce --create --alive --name "Pig Test" --input *AmazonS3bucket* –output *AmazonS3bucket* --num-instances *COUNT* --instance-type *TYPE* --pig-interactive
Hive $ ./elastic-mapreduce --create --alive --name "Hive Test" --input *AmazonS3bucket* --output *AmazonS3bucket* --num-instances *COUNT* --instance-type *TYPE* --hive-interactive

### Add a Job Flow Step

JAR $ ./elastic-mapreduce -j *JobFlowId*
Stream $ ./elastic-mapreduce -j *JobFlowId* --streaming

### Terminate a Job Flow

$ ./elastic-mapreduce --jobflow *JobFlowId* --terminate

### Get Information About a Job Flow

$ ./elastic-mapreduce --describe –-jobflow *JobFlowID*

### List Job Flows

$ ./elastic-mapreduce --list [--active] [--running] [--terminated]

### SSH Into a Master Node

./elastic-mapreduce —ssh —jobflow JobFlowID

### Use Additional Files and Libraries With the Mapper or Reducer

--cache s3n://bucket/path_to_executable#local_path

### Adding Files to the Distributed Cache

Single file: --cache s3n://my_bucket/sample_dataset.dat#sample_dataset_cached.dat
Archive file: --cache-archive s3n://my_bucket/sample_dataset.tgz#sample_dataset_cached

### Enable Output Data Compression Using the Console and a Streaming Job Flow

--jobconf mapred.output.compress=true

### Hadoop File Locations

Failure logs: /mnt/var/log/hadoop/ on each node, or *JobFlowID*/node/*InstanceID*/daemons/ on Amazon S3
UI for MapReduce job tracker(s): http://*master_dns_name*:9100/
UI for HDFS name node(s): http://*master_dns_name*:9101/
Temporary files: /mnt/var/lib/hadoop/tmp
Cache: /mnt/var/lib/hadoop/mapred/taskTracker/archive/

### Credential File Fields

"access-id": "*AccessKeyID*",
"private-key": "*PrivateKey*",
"key-pair": "*KeyPair*",
"key-pair-file": "*Location_of_key_pair_PEM_file*",
"region": "*us-east-1 | us-west-1 | eu-west-1 | ap-southeast-1 | ap-northeast-1*",
"log-uri": "*Amazon_S3_bucket_for_log_files*"

### Useful Links

| | |
|---|---|
| Forum: | https://forums.aws.amazon.com/forum.jspa?forumID=52 |
| Resource Center: | http://aws.amazon.com/elasticmapreduce/ |
| Articles & Tutorials: | http://aws.amazon.com/articles/Elastic-MapReduce |
| Release Notes: | http://aws.amazon.com/releasenotes/Elastic%20MapReduce |
| Samples & Libraries: | http://aws.amazon.com/code/Elastic-MapReduce |
| Developer Tools: | http://aws.amazon.com/developertools/Elastic-MapReduce |
| Technical Documentation: | http://aws.amazon.com/documentation/elasticmapreduce/ |
| WSDL Location: | http://elasticmapreduce.amazonaws.com/doc/2009-03-31/ElasticMapReduce.wsdl |
| CLI Download: | http://aws.amazon.com/developertools/2264 |

### Log File Locations

[log-uri]/*JobFlowId*/jobs/
[log-uri]/*JobFlowId*/node/
[log-uri]/*JobFlowId*/steps/
[log-uri]/*JobFlowId*/steps/*stepNumber*/syslog
[log-uri]/*JobFlowId*/steps/*stepNumber*/stdout
[log-uri]/*JobFlowId*/steps/*stepNumber*/controller
[log-uri]/*JobFlowId*/steps/*stepNumber*/stderr
[log-uri]/*JobFlowId*/task-attempts/

amazon web services

# Amazon Elastic MapReduce (API Version 2009-03-31)

Revised: 4/25/2011

Quick Reference Card (page 2)

## Command Line Options

--active List running, starting, or shutting down job flows
--alive Create a job flow that stays running even though it has executed all of its steps
--all List all job flows in the last 2 months
--arg *ARG* Specify an argument to a JAR or a Streaming step
--cache *CACHE_FILE* A file to load into the cache, e.g. s3://mybucket/sample.py#sample.py
--create Create a new job flow
-c *CREDENTIALS_FILE* File containing *ACCESS_ID* and *SECRET_KEY*
--credentials
-a, --access_id *ACCESS_ID* AWS Access ID
-k, --secret_key *SECRET_KEY* AWS Secret Key
--debug Print stack traces when exceptions occur
--endpoint *ENDPOINT* Specify the web service endpoint
-h, --help Show help message
--hadoop-version VERSION Choose version of Hadoop, default 0.20
--hive-versions VERSION, [VERSION] Choose version(s) of Hive, default 0.5
--input *INPUT* Input to the steps, e.g. s3://mybucket/input
--instance-type *INSTANCE_TYPE* The type of the instances to launch
--JAR *JAR* Add a step that executes a JAR
--jobconf *JOB_CONF* Specify jobconf arguments to pass to streaming
-j, --jobflow *JOB_FLOW_ID* Job flow ID
--key-pair *KEYPAIR* *Location of key pair PEM file*
--list, --describe List all job flows created in the last 2 days
--log-uri *LOG_URI* Location in Amazon S3 to store logs from the job flow, for example, s3://mybucket/logs
--main-class *MAIN_CLASS* Specify main class for the JAR
--mapper *MAPPER* The mapper program or class
-n, --max-results *MAX_RESULTS* Maximum number of results to list
--name *NAME* Name of the job flow
--nosteps Do not list steps when listing jobs
--num-instances *NUM_INSTANCES* Number of instances in the job flow
--output *OUTPUT* The output to the steps, e.g. s3://mybucket/output
--reduce *REDUCER* The reducer program or class
--state *STATE* List job flows in STATE
--step-name *STEP_NAME* Add a step to the work flow
--step-action STEP_ACTION Action to take when step finishes
--stream Add a step that performs Hadoop streaming
--terminate Terminate the job flow
-v, --verbose Turn on verbose logging of program interaction
--version Print a version string

## Predefined Bootstrap Actions

--bootstrap-action "s3://[*mybucket*]/[*myfile1*]" --args "[*arg1*]","[*arg2*]"

s3://elasticmapreduce/bootstrap-actions/configure-daemons
s3://elasticmapreduce/bootstrap-actions/configure-hadoop
s3://elasticmapreduce/bootstrap-actions/configurations/latest/memory-intensive
s3://elasticmapreduce/bootstrap-actions/run-if

## Hive Commands

hive [<-f *filename*>|<-e *query-string*>] [-S] [-hiveconf x=y]* [-d *Var=Value*]*

-e '*query string*' SQL from command line (interactive)
-f *filename* SQL from file
-d Var=*Value* Passes value into Hive script as ${Var}
-S Silent mode in interactive shell where only data is emitted
-hiveconf *x=y* Use this to set Hive or Hadoop configuration variables
add *FILE value value* Adds a file to the list of resources
! *command* Execute a shell command from Hive shell
dfs *dfs command* Execute dfs command from Hive shell
list *FILE* List all the resources already added
list *FILE value* Check given resources are already added or not
*query string* Execute Hive query and send results to stdout
Quit Exit interactive shell
set *key=value* Set configuration variable
Set List configuration variables overridden by user or Hive
set -v List all Hadoop and Hive configuration variables

## Pig Relational Operators

COGROUP alias BY field_alias [INNER | OUTER] , alias BY field_alias [INNER | OUTER] [PARALLEL n] ;
CROSS alias, alias [, alias …] [PARALLEL n];
DISTINCT alias [PARALLEL n];
DUMP alias;
FILTER alias BY expression;
FOREACH { gen_blk | nested_gen_blk } [AS schema];
GROUP alias { [ALL] | [BY {[field_alias [, field_alias]] | * | [expression]] } [PARALLEL n];
JOIN alias BY field_alias, alias BY field_alias [, alias BY field_alias …] [USING "replicated"] [PARALLEL n];
LIMIT alias n;
LOAD 'data' [USING function] [AS schema];
ORDER alias BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] …] } [PARALLEL n];
SAMPLE alias size;
SPLIT alias INTO alias IF expression, alias IF expression [, alias IF expression …];
STORE alias INTO 'directory' [USING function];
STREAM alias [, alias …] THROUGH {'command' | cmd_alias } [AS schema] ;
UNION alias, alias [, alias …];

## CLI Configuration

:endpoint => "https://elasticmapreduce.amazonaws.com",
:ca_file => File.join(File.dirname(__FILE__), "cacert.pem"),
:aws_access_key => my_access_id,
:aws_secret_key => my_secret_key,
:signature_algorithm => :V2

## Resizing Running Job Flows

--modify-instance-group INSTANCE_GROUP_ID Modify an existing instance group
--add-instance-group *ROLE* Add an instance group to an existing job flow
--instance-count *INSTANCE_COUNT* Set the instance count of an instance group
--instance-type *INSTANCE_TYPE* Set the instance type of an instance group
--set-num-instances COUNT Change the number of nodes of an instance group

- Hope to get on the cloud during week 5
- (as of 2/4/2016)

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# Running MrJob on the Cloud

**James G. Shanahan**

***EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com***

**Large Scale Machine Learning**

**MIDS, UC Berkeley**

**Lecture 4, April, 2015**

# Outline

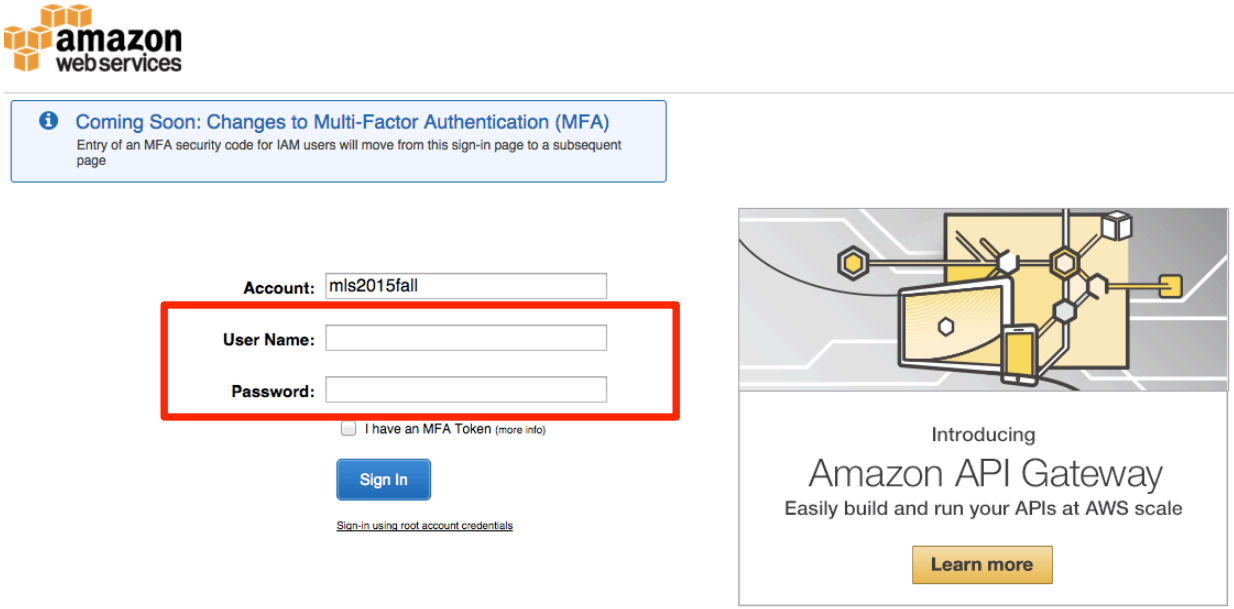
- **AWS login Info**
- **Run MrJob locally**
- **Run MrJob in AWS**

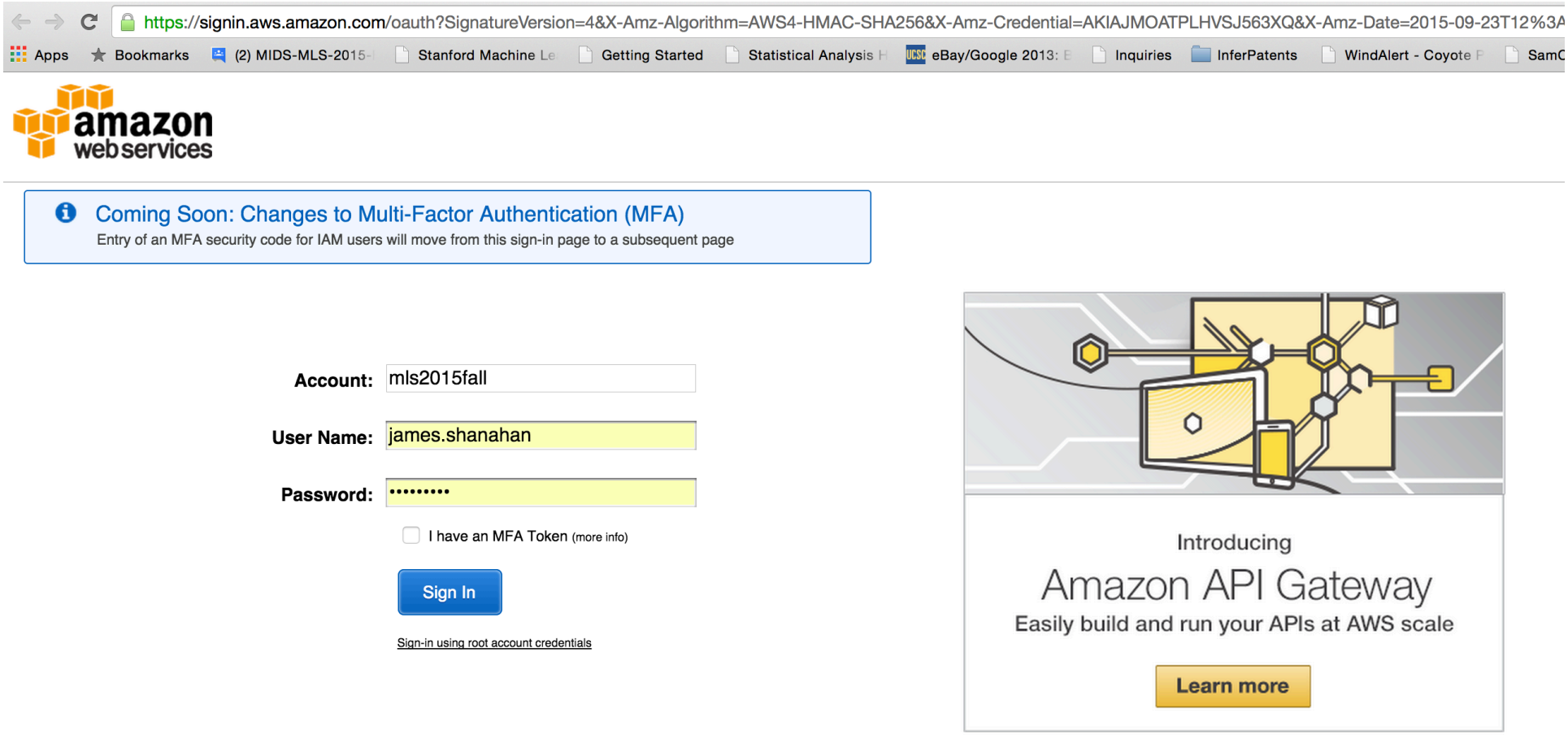
# Login

- **Link:**

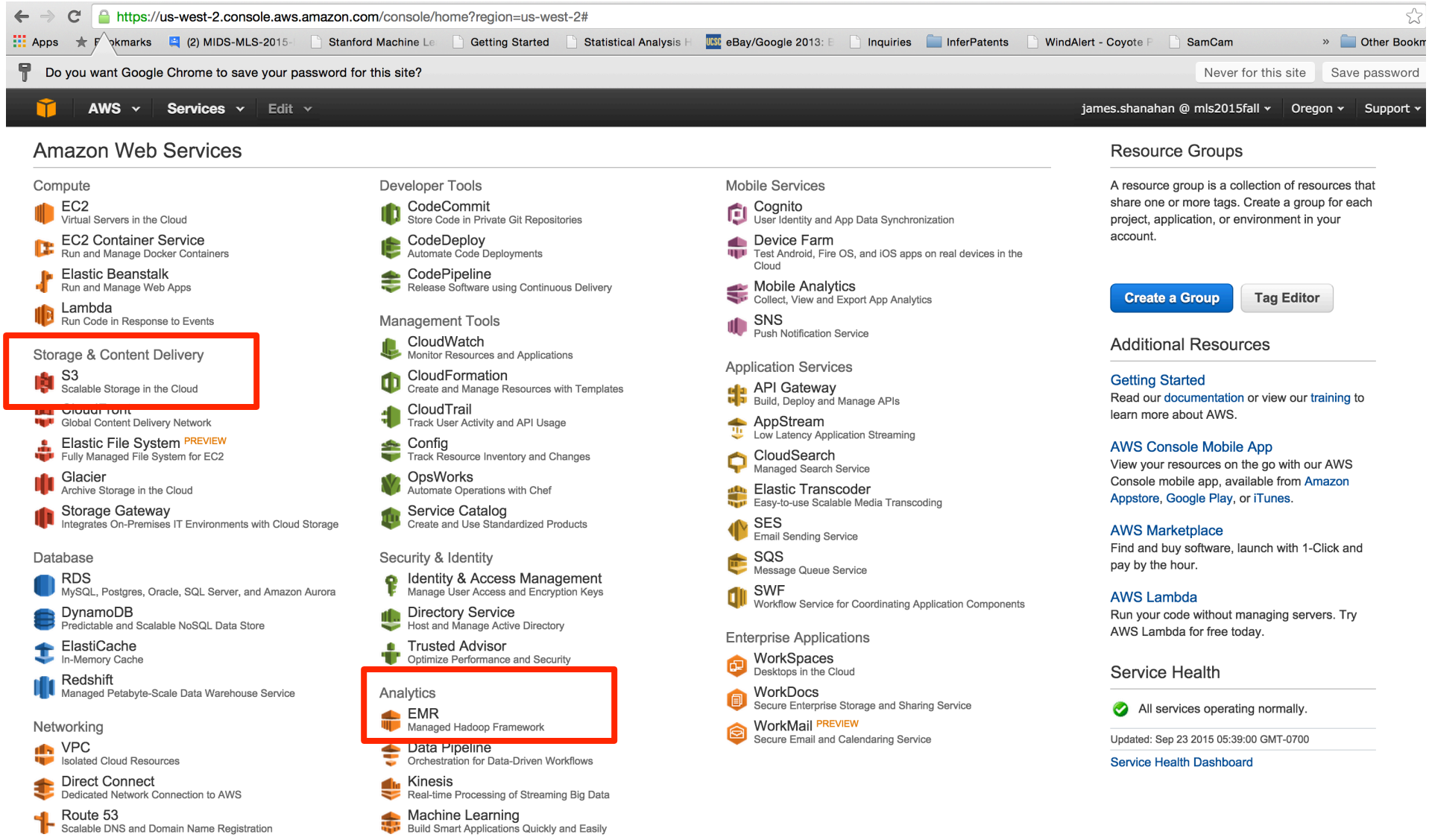
https://mls2015fall.signin.aws.amazon.com/console

- **Username: FirstName.LastName e.g. Liang.Dai**
- **Password: FirstName.LastName e.g. Liang.Dai**




https://signin.aws.amazon.com/oauth?SignatureVersion=4&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAJMOATPLHVSJ563XQ&X-Amz-Date=2015-09-23T12%3A
Apps
Bookmarks
(2) MIDS-MLS-2015-
Stanford Machine Le
Getting Started
Statistical Analysis H
eBay/Google 2013: E
Inquiries
InferPatents
WindAlert - Coyote F
SamC
amazon web services
Coming Soon: Changes to Multi-Factor Authentication (MFA)
Entry of an MFA security code for IAM users will move from this sign-in page to a subsequent page
Account:
mls2015fall
User Name:
james.shanahan
Password:
I have an MFA Token (more info)
Sign In
Sign-in using root account credentials
Introducing
Amazon API Gateway
Easily build and run your APIs at AWS scale
Learn more

# AWS: For now: S3 storage and EMR clusters



# Pricing by the hour of compute time

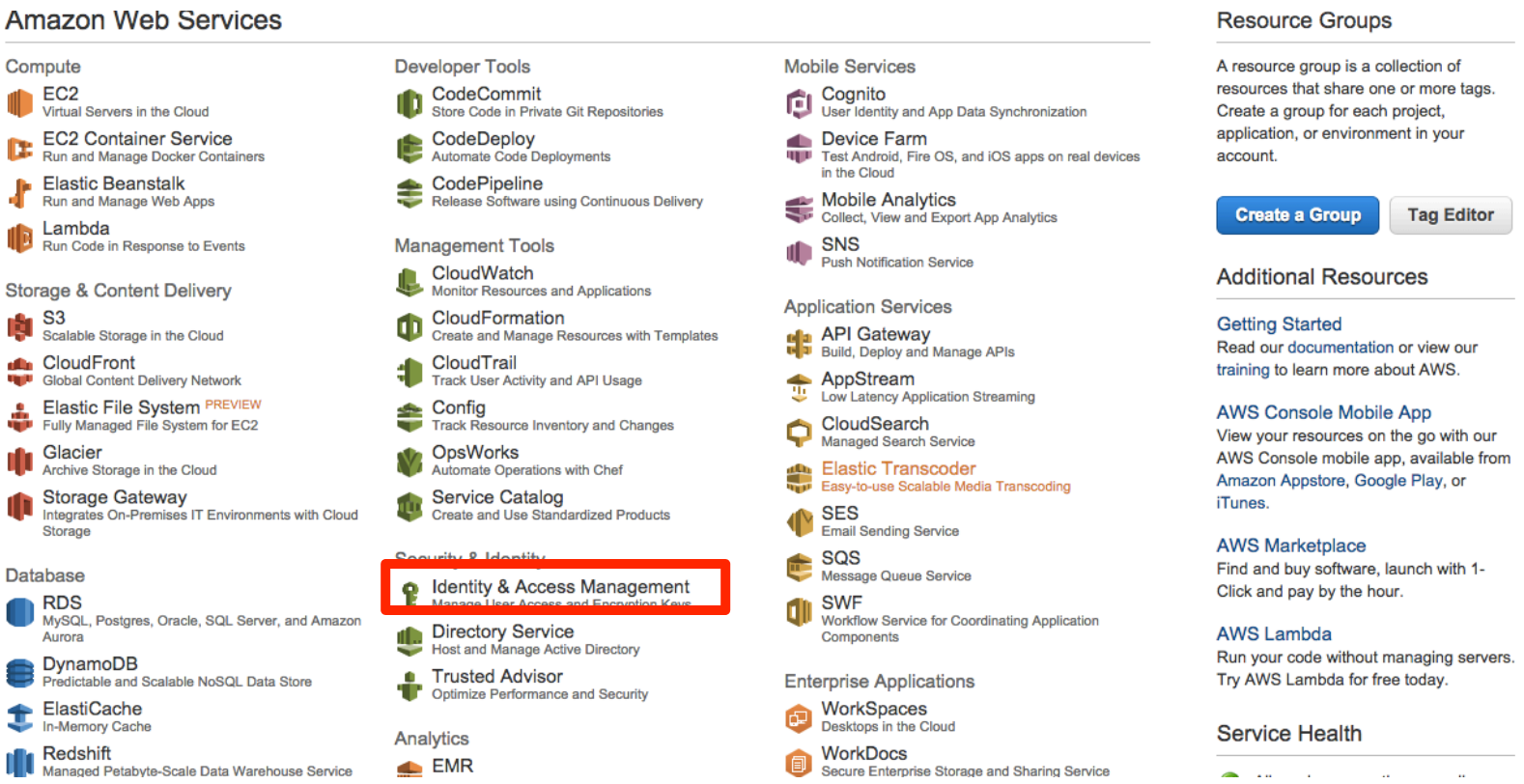
- **Pricing by the hour of compute time**
- **Pricing by the**
  - Megabyte of storage
  - Network transfer

- **So please be respectful and shutdown clusters when not in use!**

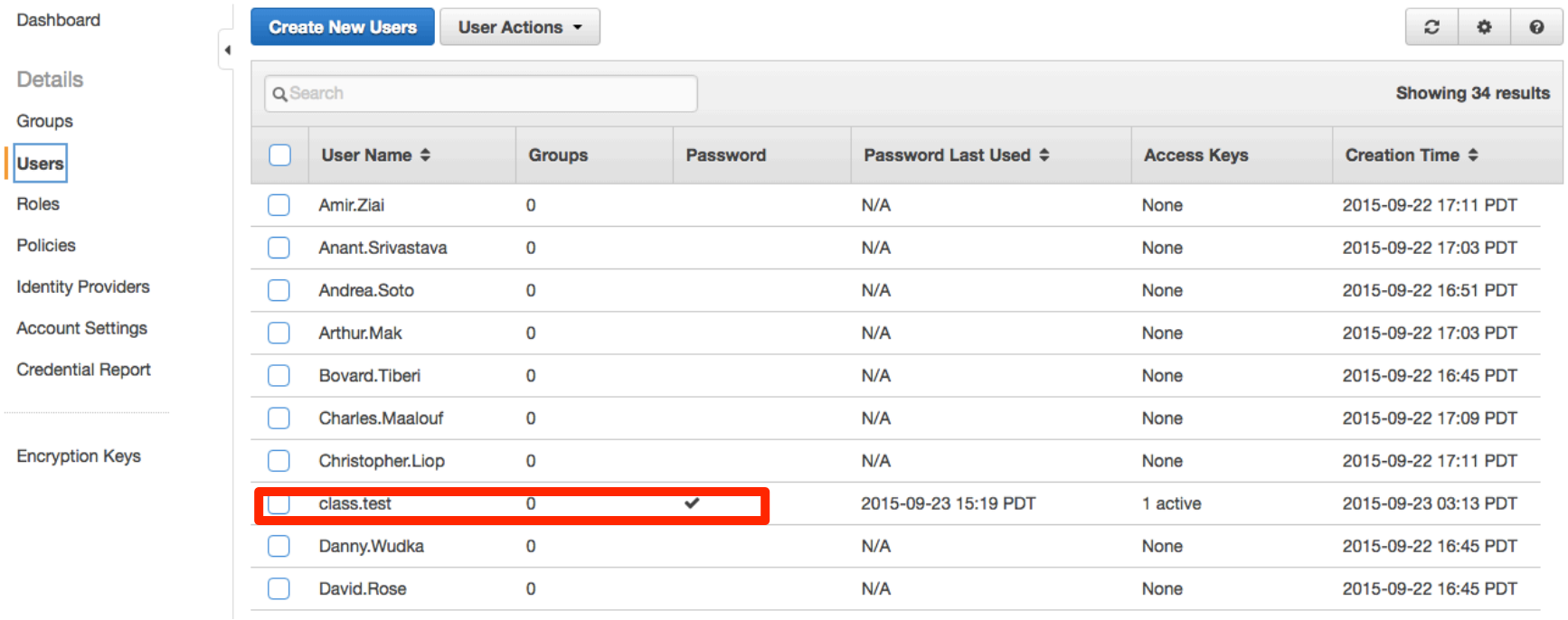
# Access Keys: programmatic calls to AWS

## Access Key will be sent to you

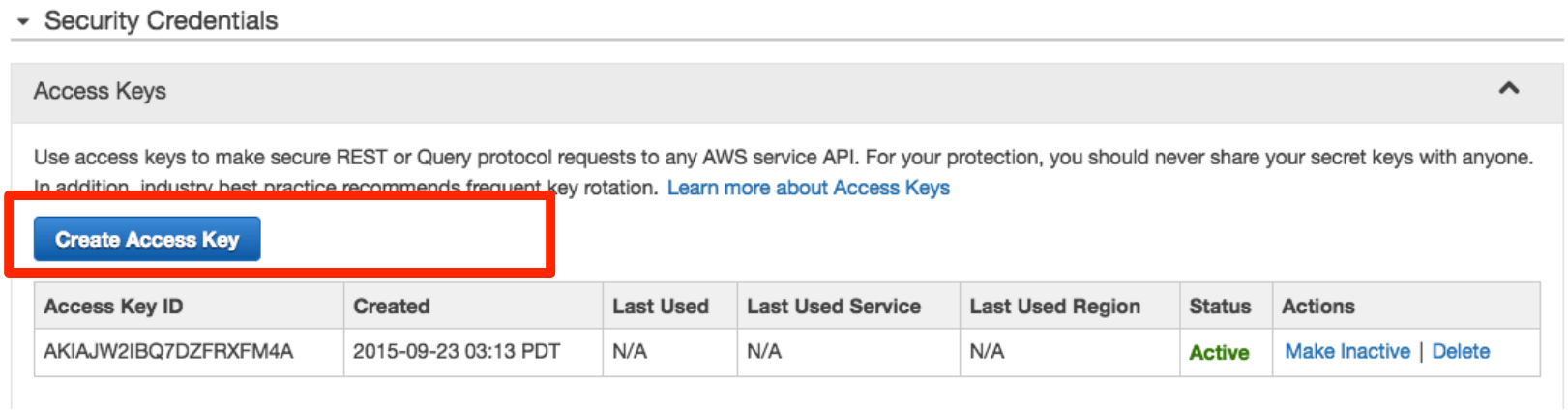
- Users need their own access keys to make programmatic calls to AWS from the AWS Command Line Interface (AWS CLI), Tools for Windows PowerShell, the AWS SDKs, or direct HTTP calls using the APIs for individual AWS services.
- Use access keys to make secure REST or Query protocol requests to any AWS service API.
- For your protection, you should never share your secret keys with anyone.

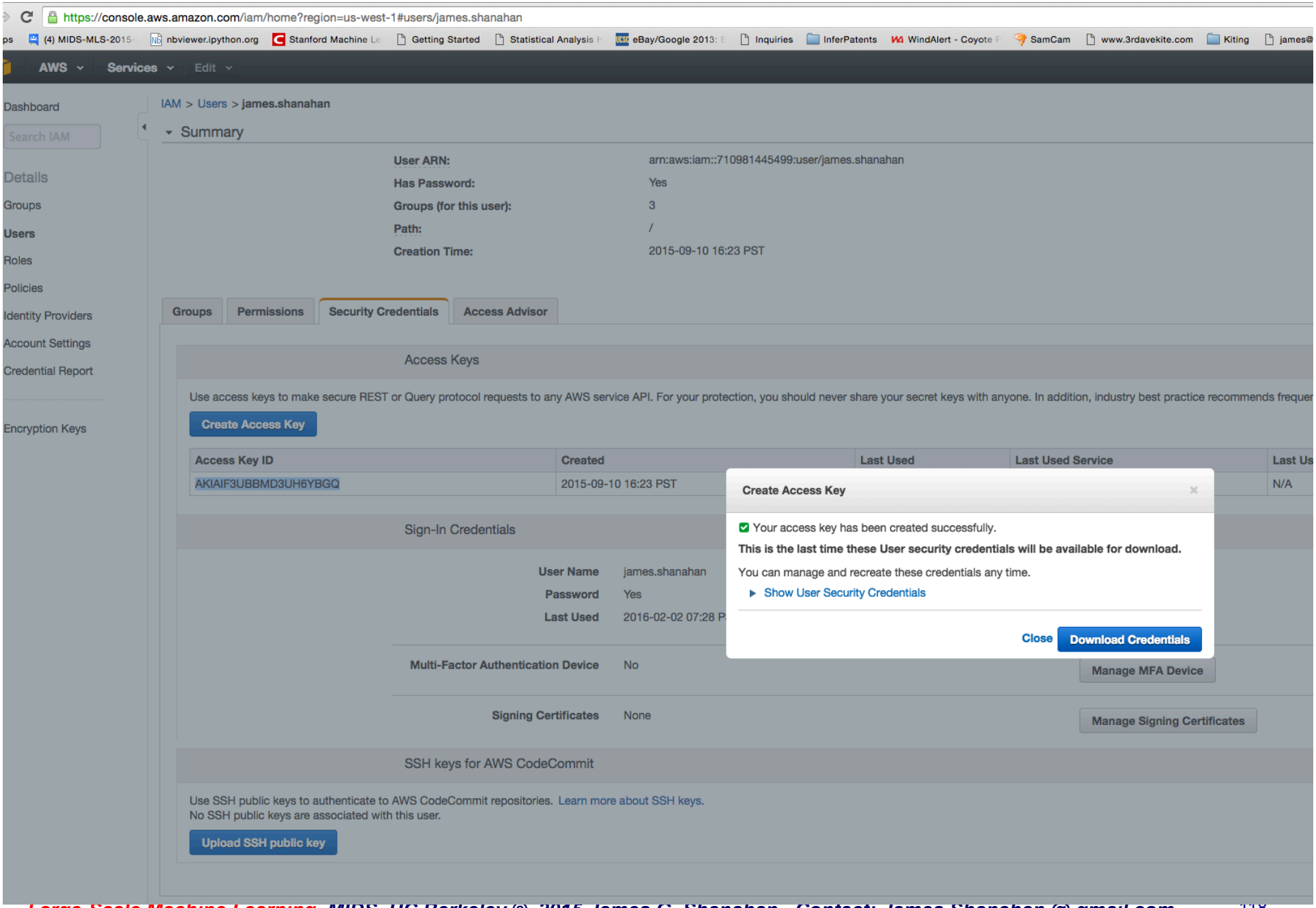

Amazon Web Services
Compute
EC2
Virtual Servers in the Cloud
EC2 Container Service
Run and Manage Docker Containers
Elastic Beanstalk
Run and Manage Web Apps
Lambda
Run Code in Response to Events
Storage & Content Delivery
S3
Scalable Storage in the Cloud
CloudFront
Global Content Delivery Network
Elastic File System PREVIEW
Fully Managed File System for EC2
Glacier
Archive Storage in the Cloud
Storage Gateway
Integrates On-Premises IT Environments with Cloud Storage
Database
RDS
MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora
DynamoDB
Predictable and Scalable NoSQL Data Store
ElastiCache
In-Memory Cache
Redshift
Managed Petabyte-Scale Data Warehouse Service
Developer Tools
CodeCommit
Store Code in Private Git Repositories
CodeDeploy
Automate Code Deployments
CodePipeline
Release Software using Continuous Delivery
Management Tools
CloudWatch
Monitor Resources and Applications
CloudFormation
Create and Manage Resources with Templates
CloudTrail
Track User Activity and API Usage
Config
Track Resource Inventory and Changes
OpsWorks
Automate Operations with Chef
Service Catalog
Create and Use Standardized Products
Identity & Access Management
Directory Service
Host and Manage Active Directory
Trusted Advisor
Optimize Performance and Security
Analytics
EMR
Mobile Services
Cognito
User Identity and App Data Synchronization
Device Farm
Test Android, Fire OS, and iOS apps on real devices in the Cloud
Mobile Analytics
Collect, View and Export App Analytics
SNS
Push Notification Service
Application Services
API Gateway
Build, Deploy and Manage APIs
AppStream
Low Latency Application Streaming
CloudSearch
Managed Search Service
Elastic Transcoder
Easy-to-use Scalable Media Transcoding
SES
Email Sending Service
SQS
Message Queue Service
SWF
Workflow Service for Coordinating Application Components
Enterprise Applications
WorkSpaces
Desktops in the Cloud
WorkDocs
Secure Enterprise Storage and Sharing Service
Resource Groups
A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.
Create a Group
Tag Editor
Additional Resources
Getting Started
Read our documentation or view our training to learn more about AWS.
AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.
AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.
AWS Lambda
Run your code without managing servers. Try AWS Lambda for free today.
Service Health

# Click “User” and select your  click your username

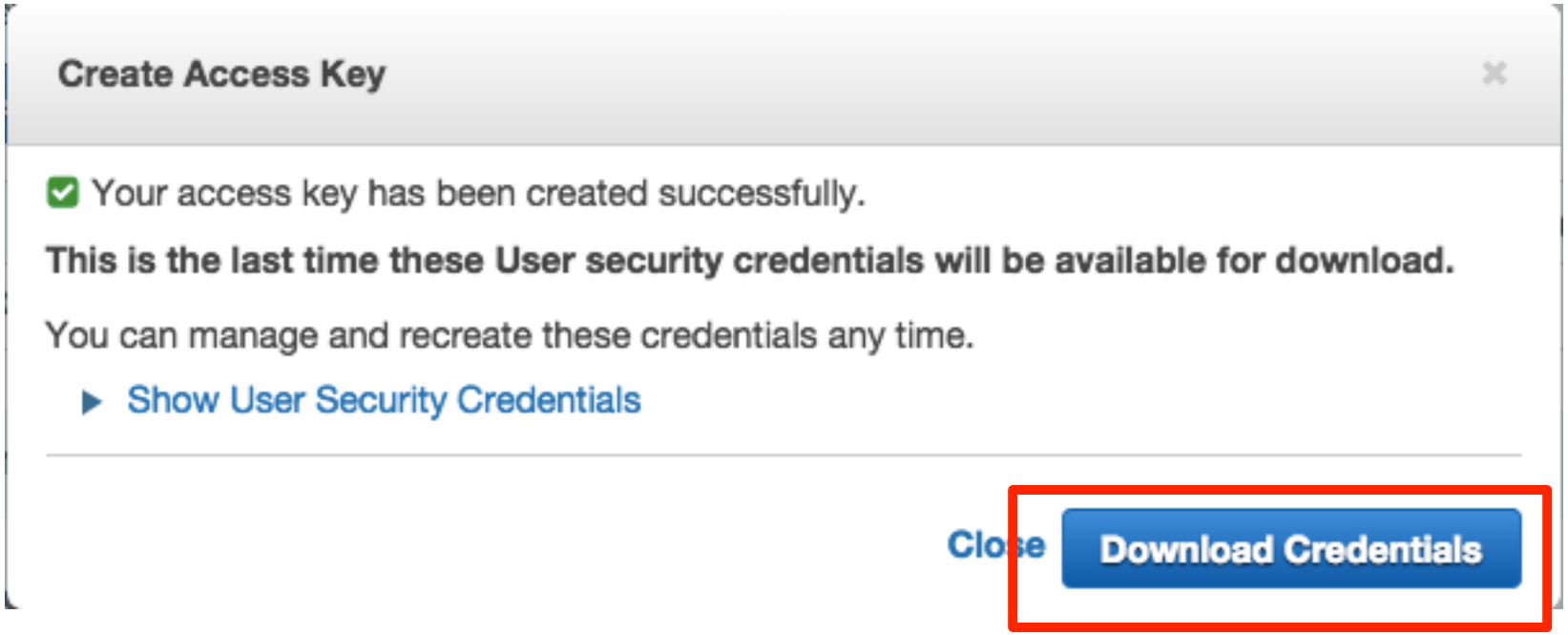


# Create access key under “Security Credentials”




https://console.aws.amazon.com/iam/home?region=us-west-1#users/james.shanahan
AWS
Services
Edit
Dashboard
Search IAM
Details
Groups
Users
Roles
Policies
Identity Providers
Account Settings
Credential Report
Encryption Keys
IAM > Users > james.shanahan
Summary
User ARN: arn:aws:iam::710981445499:user/james.shanahan
Has Password: Yes
Groups (for this user): 3
Path: /
Creation Time: 2015-09-10 16:23 PST
Groups
Permissions
Security Credentials
Access Advisor
Access Keys
Use access keys to make secure REST or Query protocol requests to any AWS service API. For your protection, you should never share your secret keys with anyone. In addition, industry best practice recommends frequer
Create Access Key
Access Key ID
Created
Last Used
Last Used Service
AKIAIF3UBBMD3UH6YBGQ
2015-09-10 16:23 PST
N/A
Create Access Key
Your access key has been created successfully.
This is the last time these User security credentials will be available for download.
You can manage and recreate these credentials any time.
Show User Security Credentials
Close
Download Credentials
Sign-In Credentials
User Name james.shanahan
Password Yes
Last Used 2016-02-02 07:28 P
Multi-Factor Authentication Device No
Manage MFA Device
Signing Certificates None
Manage Signing Certificates
SSH keys for AWS CodeCommit
Use SSH public keys to authenticate to AWS CodeCommit repositories. Learn more about SSH keys.
No SSH public keys are associated with this user.
Upload SSH public key

# Click download your credentials



# Open CSV to get your keys

| User Name | Access Key Ic | Secret Access Key | | |
|---|---|---|---|---|
| class.test | AKIAJVNWF7 | BVCWaJqmgyRuHlcGu0lKJF4hl3gy0OM436pJlAbD | | |

# Outline

- AWS login Info
- Run MrJob locally
- Run MrJob in AWS

# Mrjob Code&File

- https://www.dropbox.com/s/4417qjsrkkcywkh/MrjobWordCount.ipynb?dl=0
- http://nbviewer.ipython.org/urls/dl.dropbox.com/s/4417qjsrkkcywkh/MrjobWordCount.ipynb

## Write some words to a file

```
!echo foo foo quux labs foo bar quux > WordCount.txt
```

## MrJob class for wordcount

```
%%writefile WordCount.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFreqCount(MRJob):
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def combiner(self, word, counts):
        yield word, sum(counts)

    def reducer(self, word, counts):
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
Writing WordCount.py
```

# Run the code locally

## Run the code in command line locally

```
!python WordCount.py WordCount.txt
```

## Run the code through python driver locally

**Reminder: You cannot use the programmatic runner functionality in the same file as your job class. That is because the file with the job class is sent to Hadoop to be run. Therefore, the job file cannot attempt to start the Hadoop job, or you would be recursively creating Hadoop jobs!**

Use make_runner() to run an MRJob

1. seperate driver from mapreduce jobs
2. now we can run it within python notebook
3. In python, typically one class is in each file. Each mrjob job is a seperate class, should be in a seperate file

```
from WordCount import MRWordFreqCount
mr_job = MRWordFreqCount(args=['WordCount.txt'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
(u'bar', 1)
(u'foo', 3)
(u'labs', 1)
(u'quux', 2)
```

# Outline

- AWS login Info
- Run MrJob locally
- Run MrJob in AWS

- **MRJob on the cloud**

# MrJob

## Configuration ¶

mrjob has an overflowing cornucopia of configuration options. You'll want to specify some on the command line, some in a config file.

You can put a config file at `/etc/mrjob.conf`, `~/.mrjob.conf`, or `./mrjob.conf` for mrjob to find it without passing it via `--conf-path`.

Config files are interpreted as YAML if you have the `yaml` module installed. Otherwise, they are interpreted as JSON.

See *Config file format and location* for in-depth information. Here is an example file:

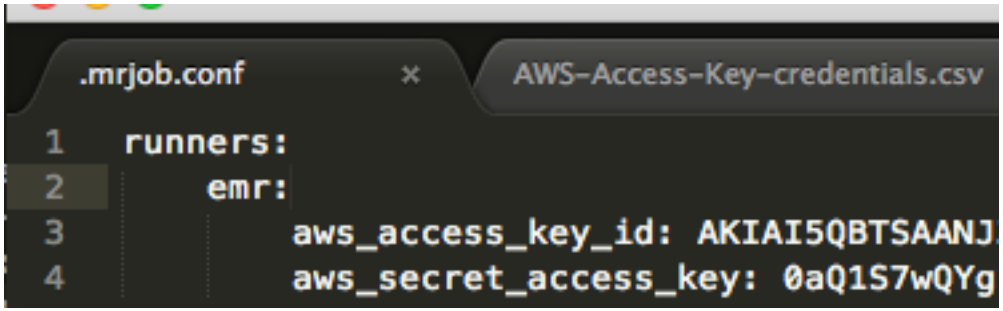
```
runners:
  emr:
    aws-region: us-west-1
    python_archives:
      - a_library_I_use_on_emr.tar.gz
  inline:
    base_tmp_dir: $HOME/.tmp
```

## Let's first take a look at mrjob.conf

We can use a pretty basic configuration for all of this, but without multiple instances, we would have to wait a long time for things to finish.

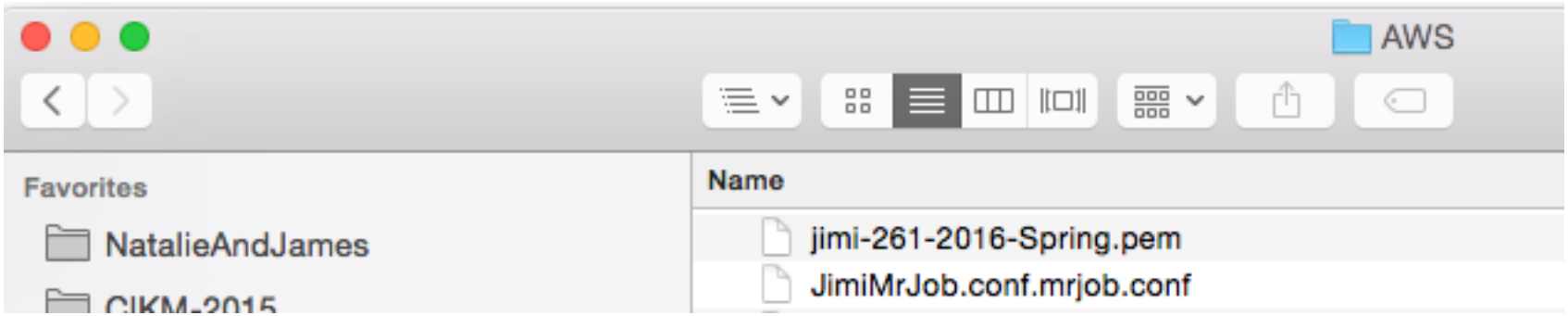
**cat ~/.mrjob.conf**

```
runners:
  emr:
    aws_access_key_id: XXXX
    aws_secret_access_key: XXXX
    num_ec2_core_instances: 4
    ec2_core_instance_type: m1.medium
    ec2_master_instance_type: m1.medium
    strict_protocols: true
  hadoop:
    strict_protocols: true
  inline:
    strict_protocols: true
  local:
    strict_protocols: true
```



# .mrjob.conf

- ..




## Dropbox Slides/AWS

## Run the code in command line in AWS

- Check you configration file path
- Create .mrjob.conf
- Put your access key info in configuration file

```
from mrjob import conf
conf.find_mrjob_conf()
```

```
'/Users/jshanahan/.mrjob.conf'
```

##Create or replace .mrjob.conf file

```
runners:
    emr:
        aws_access_key_id: your_access_key_id
        aws_secret_access_key: your_secret_access_key
```

**Run an MRJob on AWS EMR from your local computer**
**STEP 1: set up access credentials**

**1 master node +1/2 workers (cheap nodes)**
**5-10 minutes to provision the machines**
**2-3 minutes to run the job**

## Run the code in command line in AWS

```
!python WordCount.py WordCount.txt -r emr
```

```
using configs in /Users/jshanahan/.mrjob.conf
using existing scratch bucket mrjob-26635e5bb8bb690f
using s3://mrjob-26635e5bb8bb690f/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479
writing master bootstrap script to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.2218
00.233479/b.py

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --
strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-
protocols

Copying non-input files into s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-1NFNBUU1OYOMT
Created new job flow j-1NFNBUU1OYOMT
```

# Run the code in command line in A

- Check you configration file path
- Create .mrjob.conf
- Put your access key info in configuration file

```
from mrjob import conf
conf.find_mrjob_conf()
```

```
'/Users/jshanahan/.mrjob.conf'
```

##Create or replace .mrjob.conf file

```
runners:
    emr:
        aws_access_key_id: your_access_key_id
        aws_secret_access_key: your_secret_access_
```

# Run the code in command line in A

```
!python WordCount.py WordCount.txt -r emr
```

```
using configs in /Users/jshanahan/.mrjob.conf
using existing scratch bucket mrjob-26635e5bb8bb69
using s3://mrjob-26635e5bb8bb690f/tmp/ as our scra
creating tmp directory /var/folders/j4/95k348x940x
writing master bootstrap script to /var/folders/j4
00.233479/b.py

PLEASE NOTE: Starting in mrjob v0.5.0, protocols wi
strict-protocols or set up mrjob.conf as described
protocols

Copying non-input files into s3://mrjob-26635e5bb8
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-1NFNBUU1OYOMT
Created new job flow j-1NFNBUU1OYOMT
```

## Run the code in command line in AWS

```
!python WordCount.py WordCount.txt -r emr
```

```
using configs in /Users/jshanahan/.mrjob.conf
using existing scratch bucket mrjob-26635e5bb8bb690f
using s3://mrjob-26635e5bb8bb690f/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479
writing master bootstrap script to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479/b.py

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols

Copying non-input files into s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-1NFNBUU1OYOMT
Created new job flow j-1NFNBUU1OYOMT
Job launched 30.8s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 62.2s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 93.1s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 124.6s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 155.4s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 186.4s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 217.3s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 248.6s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 279.5s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 310.4s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 341.2s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 372.7s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 403.5s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job completed.
Running time was 113.0s (not counting time spent waiting for the EC2 instances)
ec2_key_pair_file not specified, going to S3
Fetching counters from S3...
Waiting 5.0s for S3 eventual consistency
Counters from step 1:
  File Input Format Counters :
    Bytes Read: 227
  File Output Format Counters :
    Bytes Written: 82
  FileSystemCounters:
    FILE_BYTES_READ: 116
    FILE_BYTES_WRITTEN: 134262
    HDFS_BYTES_READ: 588
    S3_BYTES_READ: 227
    S3_BYTES_WRITTEN: 82
  Job Counters :
    Launched map tasks: 4
    Launched reduce tasks: 1
    Rack-local map tasks: 4
    SLOTS_MILLIS_MAPS: 76840
    SLOTS_MILLIS_REDUCES: 38567
    Total time spent by all maps waiting after reserving slots (ms): 0
    Total time spent by all reduces waiting after reserving slots (ms): 0
  Map-Reduce Framework:
    CPU time spent (ms): 4790
    Combine input records: 0
    Combine output records: 0
    Map input bytes: 89
    Map input records: 3
    Map output bytes: 178
    Map output materialized bytes: 187
    Map output records: 18
    Physical memory (bytes) snapshot: 858865664
    Reduce input groups: 9
    Reduce input records: 18
    Reduce output records: 9
    Reduce shuffle bytes: 187
    SPLIT_RAW_BYTES: 588
    Spilled Records: 36
    Total committed heap usage (bytes): 606822400
    Virtual memory (bytes) snapshot: 3211935744
Streaming final output from s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/output/
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
"mining"        1
"quux"  2
"science"       1
removing tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479
Removing all files in s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/
Removing all files in s3://mrjob-26635e5bb8bb690f/tmp/logs/j-1NFNBUU1OYOMT/
Terminating job flow: j-1NFNBUU1OYOMT
```

## Run the code in command line in AWS

```
!python WordCount.py WordCount.txt -r emr
```

```
using configs in /Users/jshanahan/.mrjob.conf
using existing scratch bucket mrjob-26635e5bb8bb690f
using s3://mrjob-26635e5bb8bb690f/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479
writing master bootstrap script to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.2218
00.233479/b.py

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --
strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-
protocols

Copying non-input files into s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Job flow created with ID: j-1NFNBUU1OYOMT
Created new job flow j-1NFNBUU1OYOMT
Job launched 30.8s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 62.2s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 93.1s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 124.6s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 155.4s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 186.4s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 217.3s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 248.6s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 279.5s ago, status BOOTSTRAPPING: Running bootstrap actions
Job launched 310.4s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 341.2s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 372.7s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job launched 403.5s ago, status RUNNING: Running step (WordCount.jshanahan.20160203.221800.233479: Step 1 of 1)
Job completed.
Running time was 113.0s (not counting time spent waiting for the EC2 instances)
ec2_key_pair_file not specified, going to S3
Fetching counters from S3...
Waiting 5.0s for S3 eventual consistency
Counters from step 1:
  File Input Format Counters :
    Bytes Read: 227
  File Output Format Counters :
    Bytes Written: 82
  FileSystemCounters:
    FILE_BYTES_READ: 116
    FILE_BYTES_WRITTEN: 134262
    HDFS_BYTES_READ: 588
    S3_BYTES_READ: 227
    S3_BYTES_WRITTEN: 82
  Job Counters :
    Launched map tasks: 4
    Launched reduce tasks: 1
    Rack-local map tasks: 4
    SLOTS_MILLIS_MAPS: 76840
    SLOTS_MILLIS_REDUCES: 38567
    Total time spent by all maps waiting after reserving slots (ms): 0
    Total time spent by all reduces waiting after reserving slots (ms): 0
  Map-Reduce Framework:
    CPU time spent (ms): 4790
    Combine input records: 0
    Combine output records: 0
    Map input bytes: 89
    Map input records: 3
    Map output bytes: 178
    Map output materialized bytes: 187
    Map output records: 18
    Physical memory (bytes) snapshot: 858865664
    Reduce input groups: 9
    Reduce input records: 18
    Reduce output records: 9
    Reduce shuffle bytes: 187
    SPLIT_RAW_BYTES: 588
    Spilled Records: 36
    Total committed heap usage (bytes): 606822400
    Virtual memory (bytes) snapshot: 3211935744
Streaming final output from s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/output/
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
"mining"        1
"quux"  2
"science"       1
removing tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160203.221800.233479
Removing all files in s3://mrjob-26635e5bb8bb690f/tmp/WordCount.jshanahan.20160203.221800.233479/
Removing all files in s3://mrjob-26635e5bb8bb690f/tmp/logs/j-1NFNBUU1OYOMT/
Terminating job flow: j-1NFNBUU1OYOMT
```

# Serialization

- **Alababa (56 bits)**
- **Enode in bit form (like serialization)**
  - A:4 → represent A as binary with 2 bits  01 →4a uses 8bits
  - B: → represent B in binary  as 11 → 4 bits
  - L as itself  8bits
  - 20 bit encoding versus 56 bits in Ascii
  - Have a look at AVRO encoding

# Make files available to Mappers/Reducers

**upload_files** (`--file`) : *path list*

**Default:** `[]`

Files to copy to the local directory of the mr_job script when it runs. You can set the local name of the dir we unpack into by appending `#localname` to the path; otherwise we just use the name of the file.

In the config file:

```
upload_files:
  - file_1.txt
  - file_2.sqlite
```

On the command line:

```
--file file_1.txt --file file_2.sqlite
```

http://mrjob.readthedocs.org/en/latest/guides/configs-all-runners.html#making-files-available-to-tasks

# Passing a file to MRJob

## Additional control

```
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.protocol import RawValueProtocol
from mrjob.step import MRStep
import re

class performJoins(MRJob):

    OUTPUT_PROTOCOL = RawValueProtocol

    def steps(self):
        return [MRStep(
                mapper_init = self.mapper_init,
                mapper = self.mapper,
                mapper_final = self.mapper_final
            )]
    def mapper_init(self):
        self.URLS = {}
        self.URLS_ONRIGHT = {}
        self.left = 0
        self.right = 0
        self.inner = 0
        f = open("anonymous-msweb-URLs.data","r")
        for line in f:
            line = line.strip()
            data = re.split(",",line)
            self.URLS[data[1]] = data[4]

    def mapper(self, _, line):
        line = line.strip()
        data = re.split(",",line)
        key = data[1]
        if key in self.URLS.keys():
            self.inner += 1
            yield None, "inner,"+s
            self.left += 1
            yield None, "left,"+s
            self.right += 1
            yield None, "right,"+s
            self.URLS_ONRIGHT[key
        else:
            self.right += 1
            yield None, "right,NA"

    def mapper_final(self):
        for key in self.URLS.keys():
            if key not in self.URLS_ONRIGHT.keys():
                self.left += 1
                yield None, "left,"+self.URLS[key]+",NA,NA"
        yield None, ""
        yield None, "left-joined "+str(self.left)+" rows."
        yield None, "inner-joined "+str(self.inner)+" rows."
        yield None, "right-joined "+str(self.right)+" rows."

if __name__ == '__main__':
    performJoins.run()
```

```
!chmod +x performJoins.py
```

```
!./performJoins.py anonymous-msweb-preprocessed.data \
    --file anonymous-msweb-URLs.data > joins_output.txt
```

```
using configs in /Users/jakerylandwilliams/.mrjob.conf
```

# Output Dir

**output_dir** (`--output-dir`) : *string*

**Default:** (automatic)

An empty/non-existent directory where Hadoop streaming should put the final output from the job. If you don't specify an output directory, we'll output into a subdirectory of this job's temporary directory. You can control this from the command line with `--output-dir`. This option cannot be set from configuration files. If used with the `hadoop` runner, this path does not need to be fully qualified with `hdfs://` URIs because it's understood that it has to be on HDFS.

**no_output** (`--no-output`) : *boolean*

**Default:** `False`

Don't stream output to STDOUT after job completion. This is often used in conjunction with `--output-dir` to store output only in HDFS or S3.

```
/count5gramWords.py s3://filtered-5grams/ -r emr \
   --output-dir=s3://ucb-mids-mls-jimiShanahan/count5gramWords/output \
   --no-output
```

```
In [65]: !python WordCount.py --jobconf -numReduceTasks=3 WordCount.txt --output-dir mrJobOutput


         # mr_your_job.py --jobconf mapred.map.tasks=23 --jobconf
         #> mapred.reduce.tasks=42
```

```
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
ignoring partitioner keyword arg (requires real Hadoop): 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner'

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --
strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-
protocols

creating tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mappe
r_part-00000
Counters from step 1:
  (no counters found)
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mapper-
sorted
> sort /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-mapper_par
t-00000
writing to /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-reduce
r_part-00000
Counters from step 1:
  (no counters found)
Moving /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289/step-0-reducer_par
t-00000 -> mrJobOutput/part-00000
Streaming final output from mrJobOutput
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
"mining"        1
"quux"  2
"science"       1
removing tmp directory /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T/WordCount.jshanahan.20160202.232255.808289
```

**Output directory**

```
In [49]: !ls mrJobOutput/*
         !cat mrJobOutput/part-00000
```

```
mrJobOutput/part-00000
"bar"   1
"data"  2
"foo"   4
"is"    1
"jimi"  5
"labs"  1
"mining"        1
"quux"  2
"science"       1
```

```
%%writefile collectAndSortWordCounts.py
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.protocol import RawValueProtocol
from mrjob.step import MRStep
import re

class collectAndSortWordCounts(MRJob):

    OUTPUT_PROTOCOL = RawValueProtocol

    def jobconf(self):
        orig_jobconf = super(collectAndSortWordCounts, self).jobconf()
        custom_jobconf = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.mapred.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k1rn',
            'mapred.reduce.tasks': '1',
        }
        combined_jobconf = orig_jobconf
        combined_jobconf.update(custom_jobconf)
        self.jobconf = combined_jobconf
        return combined_jobconf

    def steps(self):
        return [MRStep(mapper = self.mapper, reducer = self.reducer)]

    def mapper(self, _, line):
        line.strip()
        [word,count] = re.split("\t",line)
        count = int(count)
        yield count,word

    def reducer(self, count, words):
        for word in words:
            yield None,word+"\t"+str(count)

if __name__ == '__main__':
    collectAndSortWordCounts.run()
```

Overwriting collectAndSortWordCounts.py

```
!chmod +x collectAndSortWordCounts.py
```

**Test the code locally on a sample of output**

```
!./collectAndSortWordCounts.py count5GramWords_output.txt > collectAndSortWordCounts_output.txt
```

using configs in /Users/jakerylandwilliams/.mrjob.conf

**Sort, partition**

**Run this code in AWS and check your result. These configurations actually does not work in local mode.**

```
class MRWordFreqCount(MRJob):
    SORT_VALUES = True
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def jobconfqqqq(self):
        orig_jobconf = super(MRWordFreqCount, self).jobconf()
        custom_jobconf = {
            'mapred.output.key.comparator.class': 'org.apache.hadoop.mapred.lib.KeyFieldBasedComparator',
            'mapred.text.key.comparator.options': '-k2,2nr',
            'mapred.reduce.tasks': '1',
        }
        combined_jobconf = orig_jobconf
        combined_jobconf.update(custom_jobconf)
        self.jobconf = combined_jobconf
        return combined_jobconf

    def steps(self):
        return [MRStep(
                mapper = self.mapper,
#                #combiner = self.combiner,
                reducer = self.reducer,
                #,
#                #jobconf = self.jobconfqqqq

            jobconf = {'mapred.output.key.comparator.class': 'org.apache.hadoop.mapred.lib.KeyFieldBasedComparator',
                       'mapred.text.key.comparator.options':'-k1r',
                       'mapred.reduce.tasks' : 1}


            )]


    def combiner(self, word, counts):
        yield word, sum(counts)

    def reducer(self, word, counts):
        yield word, sum(counts)

        ## three steps here

if __name__ == '__main__':
    MRWordFreqCount.run()
```

```
%%writefile count5gramWords.py
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.protocol import RawValueProtocol
from mrjob.step import MRStep
import re

class count5gramWords(MRJob):

    OUTPUT_PROTOCOL = RawValueProtocol

    def steps(self):
        return [MRStep(mapper = self.mapper, combiner = self.combiner, reducer = self.reducer)]

    def mapper(self, _, line):
        counts = {}
        line.strip()
        [ngram,count,pages,books] = re.split("\t",line)
        count = int(count)
        words = re.split(" ",ngram)
        for word in words:
            counts.setdefault(word,0)
            counts[word] += count
        for word in counts.keys():
            yield word,counts[word]

    def combiner(self, word, counts):
        yield word,sum(counts)

    def reducer(self, word, counts):
        yield None,word+"\t"+str(sum(counts))

if __name__ == '__main__':
    count5gramWords.run()
```

```
Writing count5gramWords.py
```

```
!chmod +x count5gramWords.py
```

**Test the code locally**

```
!./count5gramWords.py gbooks_filtered_sample.txt > count5GramWords_output.txt
```

```
using configs in /Users/jakerylandwilliams/.mrjob.conf
creating tmp directory /var/folders/3y/665tnx6s0jjcysf043nfcwm80000gn/T/count5gramWords.jakerylandwilliams.2015100
4.224112.448241
writing to /var/folders/3y/665tnx6s0jjcysf043nfcwm80000gn/T/count5gramWords.jakerylandwilliams.20151004.224112.448
241/step-0-mapper_part-00000
```

# Run the code through command line in AWS

## Run the code in command line in AWS

```
!python WordCount.py WordCount.txt -r emr
```

```
using configs in /Users/liang/.mrjob.conf
creating new scratch bucket mrjob-c860ba2c57193d26
using s3://mrjob-c860ba2c57193d26/tmp/ as our scratch dir on S3
creating tmp directory /var/folders/nn/cq08lj857z584w_96tctfg8c0000gn/T/WordCount.liang.20150923.040310.806997
writing master bootstrap script to /var/folders/nn/cq08lj857z584w_96tctfg8c0000gn/T/WordCount.liang.20150923.040310.8
06997/b.py

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --
strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-
protocols

creating S3 bucket 'mrjob-c860ba2c57193d26' to use as scratch space
Copying non-input files into s3://mrjob-c860ba2c57193d26/tmp/WordCount.liang.20150923.040310.806997/files/
Waiting 5.0s for S3 eventual consistency
Creating Elastic MapReduce job flow
Auto-created instance profile mrjob-587c2f3266759c05
Auto-created service role mrjob-b85bfa748e7c3a30
Job flow created with ID: j-R56SQZYPXQFX
Created new job flow j-R56SQZYPXQFX
Job launched 36.3s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 69.1s ago, status STARTING: Provisioning Amazon EC2 capacity
Job launched 100.8s ago, status STARTING: Provisioning Amazon EC2 capacity
```

# Run the code through python driver in AWS

## Run the code through python driver in AWS

```
: from WordCount import MRWordFreqCount
mr_job = MRWordFreqCount(args=['WordCount.txt','-r', 'emr'])
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
WARNING:mrjob.runner:
WARNING:mrjob.runner:PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you
run your job with --strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.ht
ml#ready-for-strict-protocols
WARNING:mrjob.runner:
```

```
('bar', 1)
('foo', 3)
('labs', 1)
('quux', 2)
```

# More about configuration

- http://mrjob.readthedocs.org/en/latest//en/latest/guides/configs-basics.html
- **By default, only one small ec2 instance is created. If you want more, you can pass** num_ec2_core_instances **and** ec2_core_instance_type **in command, or put them in configuration file**

```
runners:
    emr:
        num_ec2_core_instances: 2
        ec2_core_instance_type: m1.small
```

- **You can also set the output bucket and key in S3**

# Attention!!!

- We have a limited budget.
- Without instructor's notification, please only use m1.small instance.
- The number of instances should never be more than 3.

**Thanks!**

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# Clustering Algorithms

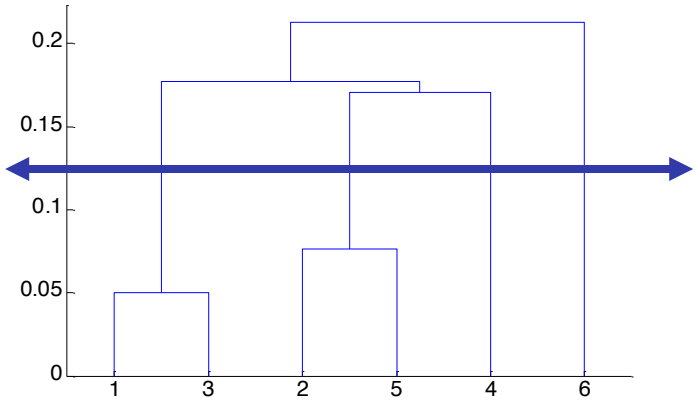
- **Non-Hierarchical Clustering**
  - Exclusive Clustering (k-means) [In R, cluster package]
  - Overlapping Clustering (fuzzy c means) [In R, e1071]
  - Probabilistic Clustering (Mixture of Gaussians)

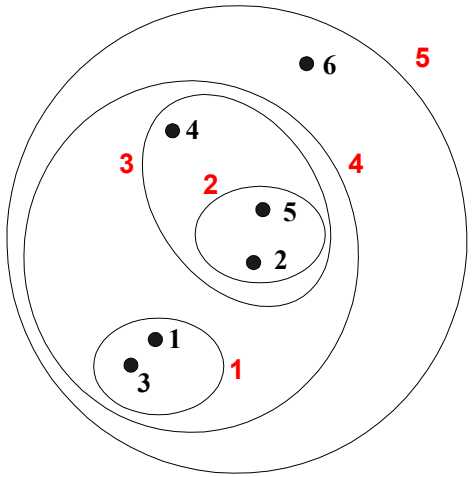
- **Hierarchical Clustering**
  - Agglomerative approach (bottom-up)
    - In R: hclust() and agnes()
  - 2. Divisive approach (top-down)
    - In R: diana()

**Question: can both types of algorithm be effectively implemented in Hadoop?**

# Clustering Algorithms

- **Hierarchical Clustering**
  - Produces a set of ***nested clusters*** organized as a hierarchical tree
  - Can be visualized as a **dendrogram**
    - A tree-like diagram that records the sequences of merges or splits



# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# Evaluating Clusterings

- **See The IR Book**
  - http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

# Thought experiment: What Is A Good Clustering?

- **Internal criterion: A good clustering will produce high quality clusters in which:**
  - the intra-class (that is, intra-cluster) similarity is high
  - the inter-class similarity is low
  - The measured quality of a clustering depends on both the document representation and the similarity measure used

# External criteria for clustering quality

- Quality measured by its ability to discover some or all of the hidden patterns or latent classes in gold standard data
- Assesses a clustering with respect to ground truth … requires *labeled data*
- Assume documents with *C* gold standard classes, while our clustering algorithms produce *K* clusters, $\omega_1, \omega_2, \ldots, \omega_K$ with $n_i$ members.

# External Evaluation of Cluster Quality

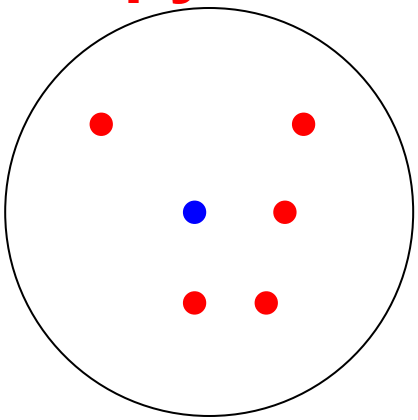
- Simple measure: purity, the ratio between the dominant class in the cluster $\omega_i$ and the size of cluster $\omega_i$

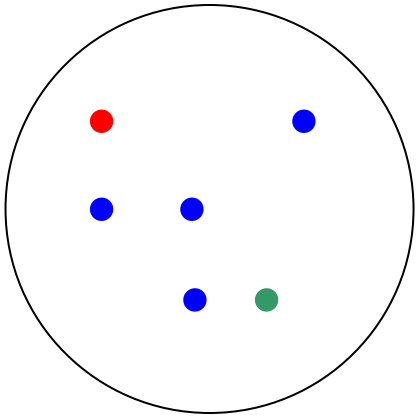
$$Purity(\omega_i) = \frac{1}{n_i} \max_j (n_{ij}) \quad j \in C$$

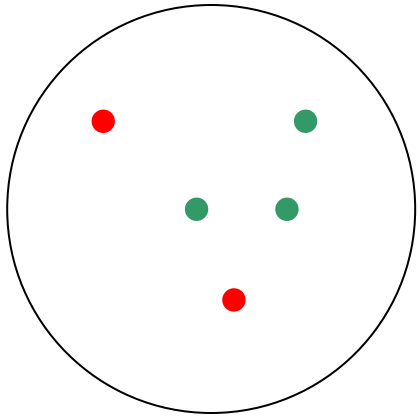
- Biased because having *n* clusters maximizes purity
- Others are entropy of classes in clusters (or mutual information between classes and clusters)

# Purity example: 3 gold classes; K=3

## Entropy= SumOver i ( $p_i$*log($p_i$))







**Cluster I**

**Cluster II**

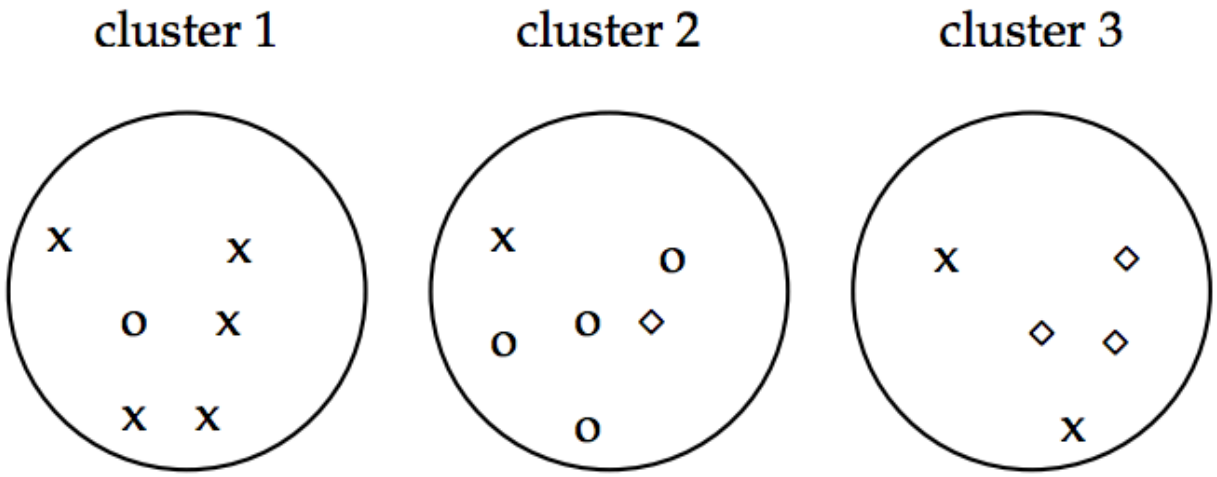
**Cluster III**

**Cluster I: Purity = 1/6 (max(5, 1, 0)) = 5/6**

**Cluster II: Purity = 1/6 (max(1, 4, 1)) = 4/6**

**Cluster III: Purity = 1/5 (max(2, 0, 3)) = 3/5**

# Overall Purity




► **Figure 16.4** Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ◇, 3 (cluster 3). Purity is $(1/17) \times (5+4+3) \approx 0.71$.

| | purity | NMI | RI | $F_5$ |
|---|---|---|---|---|
| lower bound | 0.0 | 0.0 | 0.0 | 0.0 |
| maximum | 1 | 1 | 1 | 1 |
| value for Figure 16.4 | 0.71 | 0.36 | 0.68 | 0.46 |

# Rand Index measures between pair decisions

An alternative to this information-theoretic interpretation of clustering is to view it as a series of decisions, one for each of the $N(N-1)/2$ pairs of documents in the collection. We want to assign two documents to the same cluster if and only if they are similar. A true positive (TP) decision assigns two similar documents to the same cluster, a true negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors we can commit. A false positive (FP) decision assigns two dissimilar documents to the same cluster. A false negative (FN) decision assigns two similar documents to different clusters. The *Rand index* (RI) measures the percentage of decisions that are correct. That is, it is simply accuracy (Section 8.3, page 155).

$$\mathrm{RI} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}$$

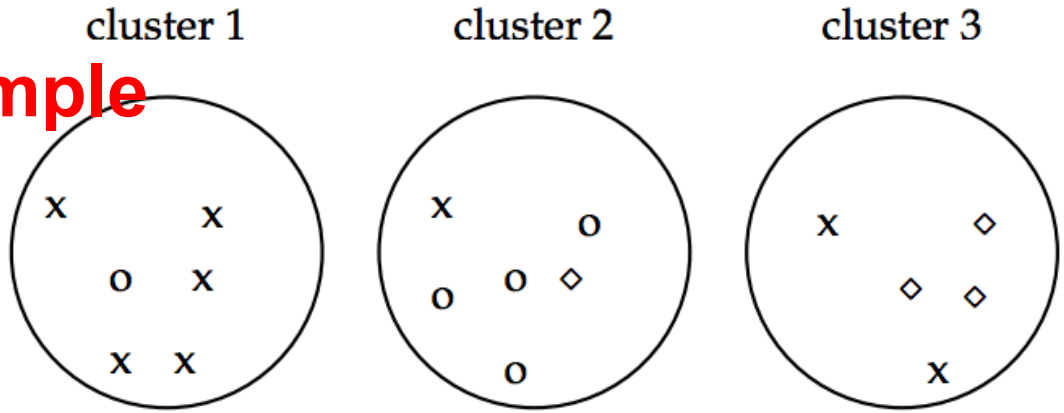
**TP = a pair of examples having the same label, are assigned to the same cluster**
**TN =a pair of examples having different labels, are assigned to different clusters**
**FP = a pair of examples having different labels, are assigned to the same cluster**
**FN = a pair of examples having the same label, are assigned to different clusters**

# Rand Index measures between pair decisions.
# Here RI = 0.68

Sec. 16.3

| **Number of point pairs** | Same Cluster in clustering | Different Clusters in clustering |
|---|---|---|
| Same class in ground truth | 20 | 24 |
| Different classes in ground truth | 20 | 72 |

# Rand Index Example




As an example, we compute RI for Figure 16.4. We first compute TP + FP. The three clusters contain 6, 6, and 5 points, respectively, so the total number of "positives" or pairs of documents that are in the same cluster is:

$$\text{TP} + \text{FP} = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the ⋄ pairs in cluster 3, and the x pair in cluster 3 are true positives:

$$\text{TP} = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

Thus, FP = 40 − 20 = 20.

FN and TN are computed similarly, resulting in the following contingency table:

| | Same cluster | Different clusters |
|---|---|---|
| Same class | TP = 20 | FN = 24 |
| Different classes | FP = 20 | TN = 72 |

RI is then $(20 + 72)/(20 + 20 + 24 + 72) \approx 0.68$.

# Rand index and Cluster F-measure

$$RI = \frac{A + D}{A + B + C + D}$$

**Compare with standard Precision and Recall:**

$$P = \frac{A}{A + B} \qquad R = \frac{A}{A + C}$$

**People also define and use a cluster F-measure, which is probably a better measure.**

# Final word and resources

- **In clustering, clusters are inferred from the data without human input (unsupervised learning)**
- **However, in practice, it's a bit less clear: there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .**

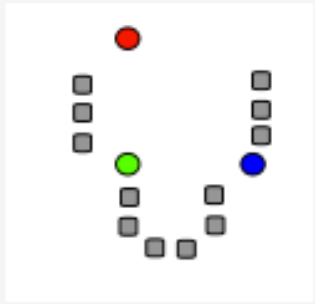
- **Resources**
  - IR Book IIR 16 except 16.5
  - IIR 17.1–17.3

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

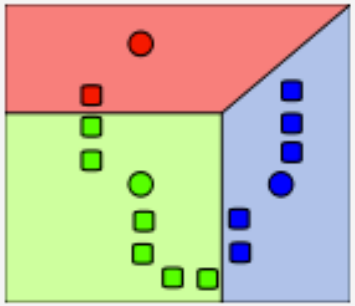
# Clustering Algorithms

## Initialization



1) *k* initial "means" (in this case *k*=3) are randomly selected from the data set (shown in color).

## E-Step "assignment" step

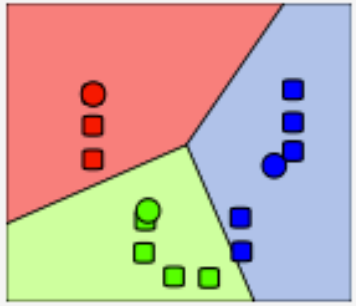


2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

## M-Step update step

3) The centroid of each of the *k* clusters becomes the new means.

## Converged



4) Steps 2 and 3 are repeated until convergence has been reached.

# K-Means Clustering Algorithm

- **Initialize our K cluster centers**
- **Perform K-Means Loop**

**E**
  - The "assignment" step is also referred to as expectation step,

**M**
  - The "update step" as maximization step, making this algorithm a variant of the *generalized* expectation-maximization algorithm.

- **Until Convergence**

# *K*-means Algorithm

- **For a current set of cluster means, assign each observation as:**

$$C(i) = \arg\min_{1 \le k \le K} \| x_i - m_k \|^2, \; i = 1, \ldots, N$$

- **For a given cluster assignment *C* of the data points, compute the cluster means $m_k$:**

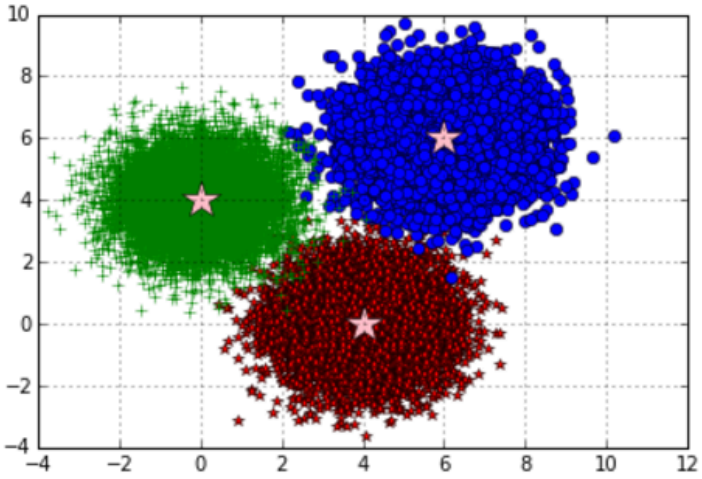
$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K.$$

- **Iterate above two steps until convergence**

```
%matplotlib inline
import numpy as np
import pylab
size1 = size2 = size3 = 10000
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('Kmeandata.csv',data,delimiter = ",")
```

## Data Visualiazation

```
pylab.plot(samples1[:, 0], samples1[:, 1],'*', color = 'red')
pylab.plot(samples2[:, 0], samples2[:, 1],'o',color = 'blue')
pylab.plot(samples3[:, 0], samples3[:, 1],'+',color = 'green')
pylab.plot((4,6,0),(0,6,4), '*', color = 'pink', markersize=20)
pylab.grid()
pylab.show()
```




**x,y coordinates**

## Driver:

Generate random initial centroids

New Centroids = initial centroids

While(1):

- Cacluate new centroids
- stop if new centroids close to old centroids
- Updates centroids

```
from numpy import random
from Kmeans import MRKmeans, stop_criterion
mr_job = MRKmeans(args=['Kmeandata.csv'])

#Geneate initial centroids
centroid_points = []
k = 3
for i in range(k):
    centroid_points.append([random.uniform(-3,3),random.uniform(-3,3)])
with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)

# Update centroids iteratively
i = 0
while(1):
    # save previous centoids to check convergency
    centroid_points_old = centroid_points[:]
    print "iteration"+str(i)+":"
    with mr_job.make_runner() as runner:
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            key,value =  mr_job.parse_output_line(line)
            print key, value
            centroid_points[key] = value
    print "\n"
    i = i + 1
    if(stop_criterion(centroid_points_old,centroid_points,0.01)):
        break
print "Centroids\n"
print centroid_points
```

```
iteration0:
0 [-2.0346556730459464, 2.6844949844288597]
1 [4.013933429097964, -0.3662183665020468]
2 [3.140748469016723, 4.67560776738735]
```

```
iteration1:
```

clusterID, center

**Input record _, (x,y)**

**clusterID with the minDist \t, x, y, 1**

**Initialize our K cluster centers**

**I**

**Repeat**

**C(i) CentroidID for example i**

**E** $$C(i) = \arg\min_{1 \le k \le K} \left\| x_i - m_k \right\|^2, \; i = 1, \ldots, N$$

**M** $$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K.$$

**Until convergence**

# Data: x,y coordinates

```
class MRKmeans(MRJob):
    centroid_points=[]
    k=3
    def steps(self):
        return [
            MRJobStep(mapper_init = self.mapper_init, mapper=self.mapper,combiner = self.combiner,reducer=self.reducer)
                ]
    #load centroids info from file
    def mapper_init(self):
        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in open("Centroids.txt").readlines()]
        open('Centroids.txt', 'w').close()
    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = (map(float,line.split(',')))
        yield int(MinDist(D,self.centroid_points)), (D[0],D[1],1)
    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        sumx = sumy = num = 0
        for x,y,n in inputdata:
            num = num + n
            sumx = sumx + x
            sumy = sumy + y
        yield idx,(sumx,sumy,num)
    #Aggregate sum for each cluster and then calculate the new centroids
    def reducer(self, idx, inputdata):
        centroids = []
        num = [0]*self.k
        for i in range(self.k):
            centroids.append([0,0])
        for x, y, n in inputdata:
            num[idx] = num[idx] + n
            centroids[idx][0] = centroids[idx][0] + x
            centroids[idx][1] = centroids[idx][1] + y
        centroids[idx][0] = centroids[idx][0]/num[idx]
        centroids[idx][1] = centroids[idx][1]/num[idx]
        with open('Centroids.txt', 'a') as f:
            f.writelines(str(centroids[idx][0]) + ',' + str(centroids[idx][1]) + '\n')
        yield idx,(centroids[idx][0],centroids[idx][1])

if __name__ == '__main__':
    MRKmeans.run()
```

Mapper_init

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2, \; i = 1, \ldots, N$$

Partial sum for a cluster centroid
Idx =clusterID, sumx, sumy, num=number of examples considered so far
*Cluster=3, (10,15, num =1) record output from mapper? Not from mapper*

Partial sum for a cluster centroid
Idx =clusterID, sumx, sumy, num=number of examples considered so far
*idx-=Cluster3, (10,15, num =4) record output from combiner?*

(3, ((10, 10, 1)).... *(2, (1, 3, 1)), (2, (3, 1, 1))*.....

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K.$$

x=(1,3,4,5,)/4

# Debug this code

Mapper, combiner do partial sums centroids and denomi
Reducers sum up the

**Matrix**

$m_k=(x, y)$
$m_1=(x_1, y_1)$
$m_2=(x_2, y_2)$
$m_3=(x_3, y_3)$

**How many reducers for kMeans?**

**Can have [1, K] reducers**

# Clustering Algorithms

- **Non-Hierarchical Clustering**
  - Exclusive Clustering (k-means) [In R
  - Overlapping Clustering (fuzzy c mea
  - Probabilistic Clustering (Mixture of G
- **Hierarchical Clustering**
  - Agglomerative approach (bottom-up)

**Question: can both types of algorithm be effectively implemented in Hadoop?**

**Hierarchical Clustering is tough to deploy:**
**Distributing a bottom-up algorithm is tricky because each distributed process needs the entire dataset to make choices about appropriate clusters. It also needs a list of clusters at its current level so it doesn't add a data point to more than one cluster at the same level.**

# Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
- **Week 4**
  - From MapReduce to MrJob
  - MrJob sample jobs (WordCount; Most frequent word, Counters)
  - Cloud computing
  - AWS, Logging into AWS and running EMR MRJobs
  - AWS Command Line Interface
  - Running MRJob jobs on EMR
  - Clustering, K-Means
  - Clustering Metrics
  - Kmeans (recap of algo. Kmeans in MrJob)
  - Canopy Clustering, Expectation Maximization [Week 5 live session]
- **Next week (Big data pipelines)**
- **Wrapup**

# Intialiatization

- Random
- Initialization with examples from training data
- Canopy clustering
- Other algorithms

## K-means

K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The `spark.mllib` implementation includes a parallelized variant of the k-means++ method called kmeans||. The implementation in `spark.mllib` has the following parameters:

- *k* is the number of desired clusters.
- *maxIterations* is the maximum number of iterations to run.
- *initializationMode* specifies either random initialization or initialization via k-means||.
- *runs* is the number of times to run the k-means algorithm (k-means is not guaranteed to find a globally optimal solution, and when run multiple times on a given dataset, the algorithm returns the best clustering result).
- *initializationSteps* determines the number of steps in the k-means|| algorithm.
- *epsilon* determines the distance threshold within which we consider k-means to have converged.
- *initialModel* is an optional set of cluster centers used for initialization. If this parameter is supplied, only one run is performed.

# Scalable K-Means++

Bahman Bahmani*†
Stanford University
Stanford, CA
bahman@stanford.edu

Benjamin Moseley*‡
University of Illinois
Urbana, IL
bmosele2@illinois.edu

Andrea Vattani*§
University of California
San Diego, CA
avattani@cs.ucsd.edu

Ravi Kumar
Yahoo! Research
Sunnyvale, CA
ravikumar@yahoo-inc.com

Sergei Vassilvitskii
Yahoo! Research
New York, NY
sergei@yahoo-inc.com

## ABSTRACT

Over half a century old and showing no signs of aging, $k$-means remains one of the most popular data processing algorithms. As is well-known, a proper initialization of $k$-means is crucial for obtaining a good final solution. The recently proposed $k$-means++ initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the $k$-means++ is its inherent sequential nature, which limits its applicability to massive data: one must make $k$ passes over the data to find a good initial set of centers. In this work we show how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing $k$-means that have mostly focused on the post-initialization phases of $k$-means. We prove that our proposed initialization algorithm $k$-means|| obtains a nearly optimal solution after a logarithmic number of passes, and then show that in practice a constant number of passes suffices. Experimental evaluation on real-world large-scale data demonstrates that $k$-means|| outperforms $k$-means++ in both sequential and parallel settings.

## 1. INTRODUCTION

Clustering is a central problem in data management and has a rich and illustrious history with literally hundreds of different algorithms published on the subject. Even so, a single method — $k$-means — remains the most popular clustering method; in fact, it was identified as one of the top 10 algorithms in data mining [34]. The advantage of $k$-means is its simplicity: starting with a set of randomly chosen initial centers, one repeatedly assigns each input point to its nearest center, and then recomputes the centers given the point assignment. This local search, called *Lloyd's* iteration, continues until the solution does not change between two consecutive rounds.

The $k$-means algorithm has maintained its popularity even as datasets have grown in size. Scaling $k$-means to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points. Indeed parallel implementations of $k$-means are readily available (see, for example, `cwiki.apache.org/MAHOUT/k-means-clustering.html`).

From a theoretical standpoint, $k$-means is not a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case [32, 4] and even though the final solution is locally optimal, it can be very far away from the global optimum (even under repeated random initializations). Nevertheless, in practice the speed and simplicity of $k$-means cannot be beat. Therefore, recent work has focused on improving the initialization procedure: deciding on a better way to initialize the clustering dramatically changes the performance of the Lloyd's iteration, both in terms of quality and convergence properties.

step in this direction was taken by Ostro- and Arthur and Vassilvitskii [5], who showed lure that both leads to good theoretical guar- uality of the solution, and, by virtue of a good improves upon the running time of Lloyd's ctice. Dubbed $k$-means++, the algorithm se- st center uniformly at random from the data. t center is selected with a probability pro- contribution to the overall error given the previous selections (we make this statement precise in Section 3). Intuitively, the initialization algorithm exploits the fact that a good clustering is relatively spread out, thus when selecting a new cluster center, preference should be given to those further away from the previously selected centers. Formally, one can show that $k$-means++ initialization

*Part of this work was done while the author was visiting

**Kmeans++ versus kmeans||**
**Canopy iniatlizaion**

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

# Kmeans++

- Arthur and Vassilvitskii [5] modified the initialization step in a careful manner and obtained a randomized initialization algorithm called k-means++.
- The main idea in their algorithm is to choose the centers one by one in a controlled fashion, where the current set of chosen centers will stochastically bias the choice of the next center (Algorithm 1).
- (running Lloyd's iteration on top of this will only improve
- the solution, but no guarantees can be made).
- The central drawback of k-means++ initialization from a scalability point of view is its inherent sequential nature: the choice of the next center depends on the current set of centers.

# kmeans||

## Scalable K-Means++

Bahman Bahmani*†
Stanford University
Stanford, CA
bahman@stanford.edu

Benjamin Moseley*‡
University of Illinois
Urbana, IL
bmosele2@illinois.edu

Andrea Vattani*§
University of California
San Diego, CA
avattani@cs.ucsd.edu

Ravi Kumar
Yahoo! Research
Sunnyvale, CA
ravikumar@yahoo-inc.com

Sergei Vassilvitskii
Yahoo! Research
New York, NY
sergei@yahoo-inc.com

**ABSTRACT**

Over half a century old and showing no signs of aging, *k*-means remains one of the most popular data processing algorithms. As is well-known, a proper initialization of *k*-means is crucial for obtaining a good final solution. The recently proposed *k*-means++ initialization algorithm achieves this, obtaining an initial set of centers that is provably close to the optimum solution. A major downside of the *k*-means++ is its inherent sequential nature, which limits its applicability to massive data: one must make $k$ passes over the data to find a good initial set of centers. In this work we show how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing *k*-means that have mostly focused on the post-initialization phases of *k*-means. We prove that our proposed initialization algorithm *k*-means|| obtains a nearly optimal solution after a logarithmic number of passes, and then show that in practice a constant number of passes suffices. Experimental evaluation on real-world large-scale data demonstrates that *k*-means|| outperforms *k*-means++ in both sequential and parallel settings.

### 1. INTRODUCTION

Clustering is a central problem in data management and has a rich and illustrious history with literally hundreds of different algorithms published on the subject. Even so, a single method — *k*-means — remains the most popular clustering method; in fact, it was identified as one of the top 10 algorithms in data mining [34]. The advantage of *k*-means is its simplicity: starting with a set of randomly chosen initial centers, one repeatedly assigns each input point to its nearest center, and then recomputes the centers given the point assignment. This local search, called *Lloyd's* iteration, continues until the solution does not change between two consecutive rounds.

The *k*-means algorithm has maintained its popularity even as datasets have grown in size. Scaling *k*-means to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points. Indeed parallel implementations of *k*-means are readily available (see, for example, `cwiki.apache.org/MAHOUT/k-means-clustering.html`).

From a theoretical standpoint, *k*-means is not a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case [32, 4] and even though the final solution is locally optimal, it can be very far away from the global optimum (even under repeated random initializations). Nevertheless, in practice the speed and simplicity of *k*-means cannot be beat. Therefore, recent work has focused on improving the initialization procedure: deciding on a better way to initialize the clustering dramatically changes the performance of the Lloyd's iteration, both in terms of quality and convergence properties.

A important step in this direction was taken by Ostrovsky et al. [30] and Arthur and Vassilvitskii [5], who showed a simple procedure that both leads to good theoretical guarantees for the quality of the solution, and, by virtue of a good starting point, improves upon the running time of Lloyd's iteration in practice. Dubbed *k*-means++, the algorithm selects only the first center uniformly at random from the data. Each subsequent center is selected with a probability proportional to its contribution to the overall error given the previous selections (we make this statement precise in Section 3). Intuitively, the initialization algorithm exploits the fact that a good clustering is relatively spread out, thus when selecting a new cluster center, preference should be given to those further away from the previously selected centers. Formally, one can show that *k*-means++ initialization

*Part of this work was done while the author was visiting Yahoo! Research.

†Research supported in part by William R. Hewlett Stanford Graduate Fellowship, and NSF awards 0915040 and IIS-0904325.
‡Partially supported by NSF grant CCF-1016684.
§Partially supported by NSF grant #0905645.

- **End of live session week**