AYUSH SINGH

CS 6240: Assignment 1

Multithreading Concepts and AWS Setup

Analyze Data with Sequential and Concurrent Programs

Weather Data Results

All time units are in **ms** (milliseconds)

NAME	AVERAGE	MINIMUM	MAXIMUM	SPEEDUP
SEQ	9535.60	9370.00	10186.00	1.000
NO-LOCK	3645.70	2575.00	5067.00	2.616
COARSE-LOCK	3920.90	2628.00	5287.00	2.432
FINE-LOCK	3468.00	2648.00	5254.00	2.750
NO-SHARING	3092.60	2783.00	5188.00	3.083

After adding Fibonacci(17) computation to all versions we get following results as below

NAME	AVERAGE	MINIMUM	MAXIMUM	SPEEDUP
SEQ	8187.10	7987.00	9387.00	1.000
NO-LOCK	3196.50	2581.00	5015	2.561
COARSE-LOCK	3976.80	2639.00	5044.00	2.059
FINE-LOCK	4025.80	2568.00	5074.00	2.034
NO-SHARING	4072.80	2674.00	5136.00	2.010

A1. As discussed in class and learning modules, theoretical speedup possible for a task divided into **n** concurrent tasks is **n**. So in this case even though NO-LOCK has no control over data access by multiple thread leading to data inconsistencies but this will indeed allow it to finish the task in hand in fastest amount of time. The experiments confirm my expectation in my case.

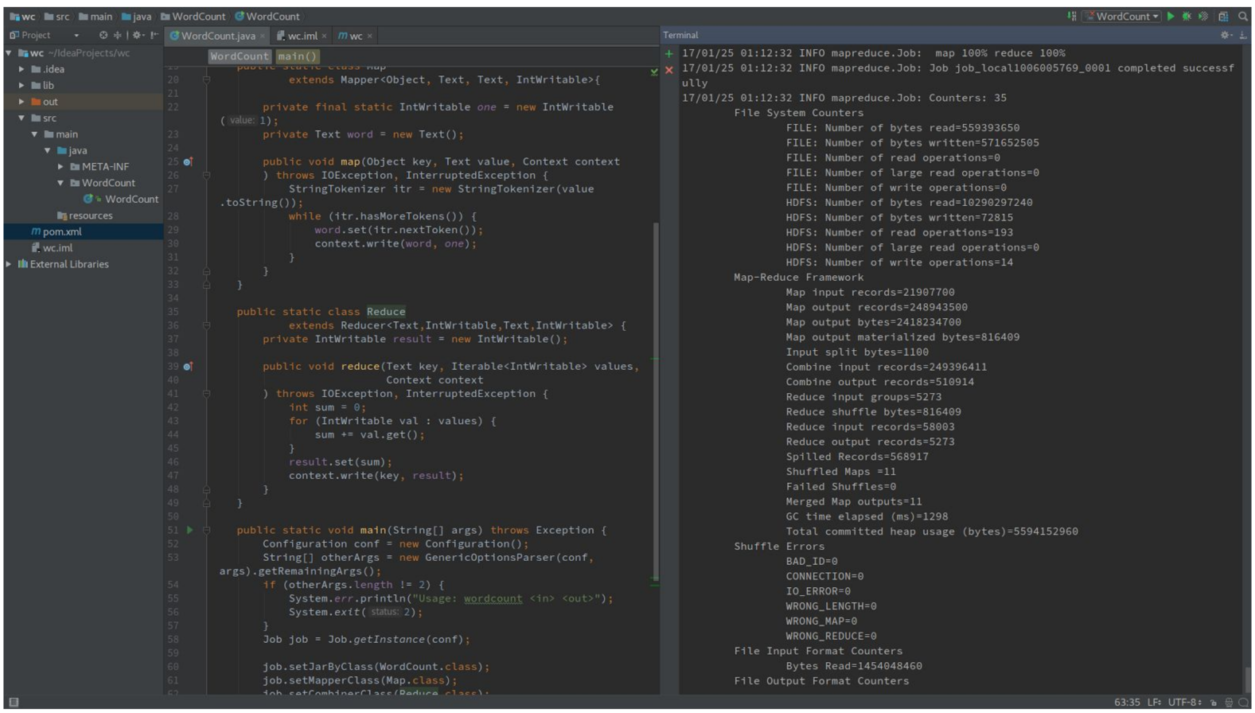
A2. I would expect SEQ to finish slowest since it acquires a lock on the since its occurs in a sequential fashion which is slow with the ones it is put in comparison with who tend to parallelize the task into N smaller units and assign separate units to separate workers for processing. Yes experiments in this case corroborate the same quantitatively performing at least in double the time compared to other algorithms in the play.

A3. Temperature averages are consistent for all programs except NO-LOCK whose averages were inconsistent when compared with others along with occasional NPEs (NullPointerException) which occurs when one thread set flag to the existence of a key before placing anything there and meanwhile another Thread enters and trIes to fetch the key which is not available there yet resulting in an NPE.

A4. Running times of SEQ and COARSE-LOCK are 9535.60 and 3920.90 which places COARSE-LOCK apprx. 2.5x faster runtimes. The reason behind this is the amount of parallelization that takes place with COARSE-LOCK by splitting the data in proportion with the number of threads. Ideally the speedup would've been a lot more if perfect parallelism was achieved by instead all parallel tasks (Threads) shared the same accumulation data structure which inturn limits how fast we can write to it since according to COARSE-LOCK strategy only one Thread could write at a time.As mentioned in my response to question 2, I expected COARSE-LOCK to finish second to SEQ since it acquires a lock on the whole data structure upon which most of the CRUD (create, read, update, delete) operations occur which render the multi threading helpless since technically the locks are acquired one after another i.e. in order making it partially sequential. Looking at results from B and C, the results did not vary much that indeed came as a revelation to me since the fibonacci I wrote used Dynamic Programming i.e. no recursion hence no significant performance drags because of memory overflow by deep recursion.

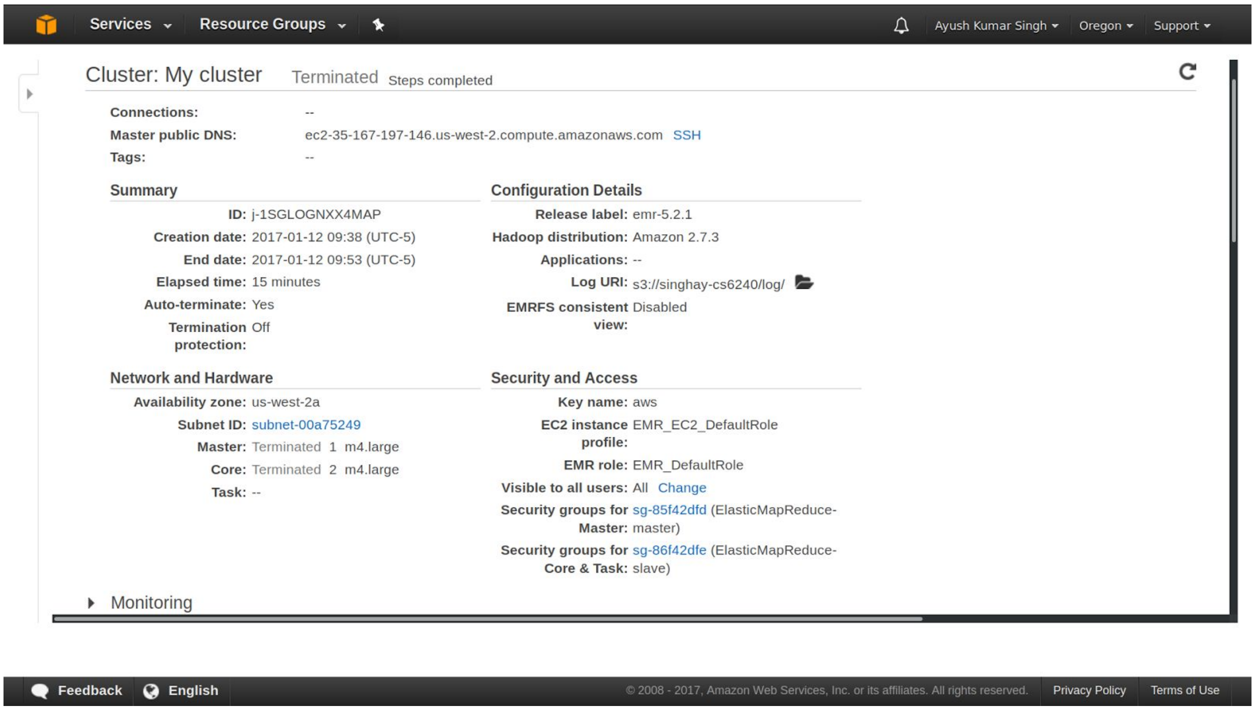
A5. The higher computation cost in part C (additional Fibonacci computation) affect no real difference between COARSE-LOCK and FINE-LOCK because of similar reasons I stated in answer A4 where I mentioned that the fibonacci algorithm I implement had an asymptotic time complexity of O(n) which is alot better than the conventional recursion version and computes really quickly which did not really add much computation overhead to the machine. But I did try with much higher numbers e.g. 45 and fine-locking was able to shave off around 1.5s off the runtime which was what I was expecting since the introduction of granular locking on accumulation objects instead of the whole data structure improved the computation.

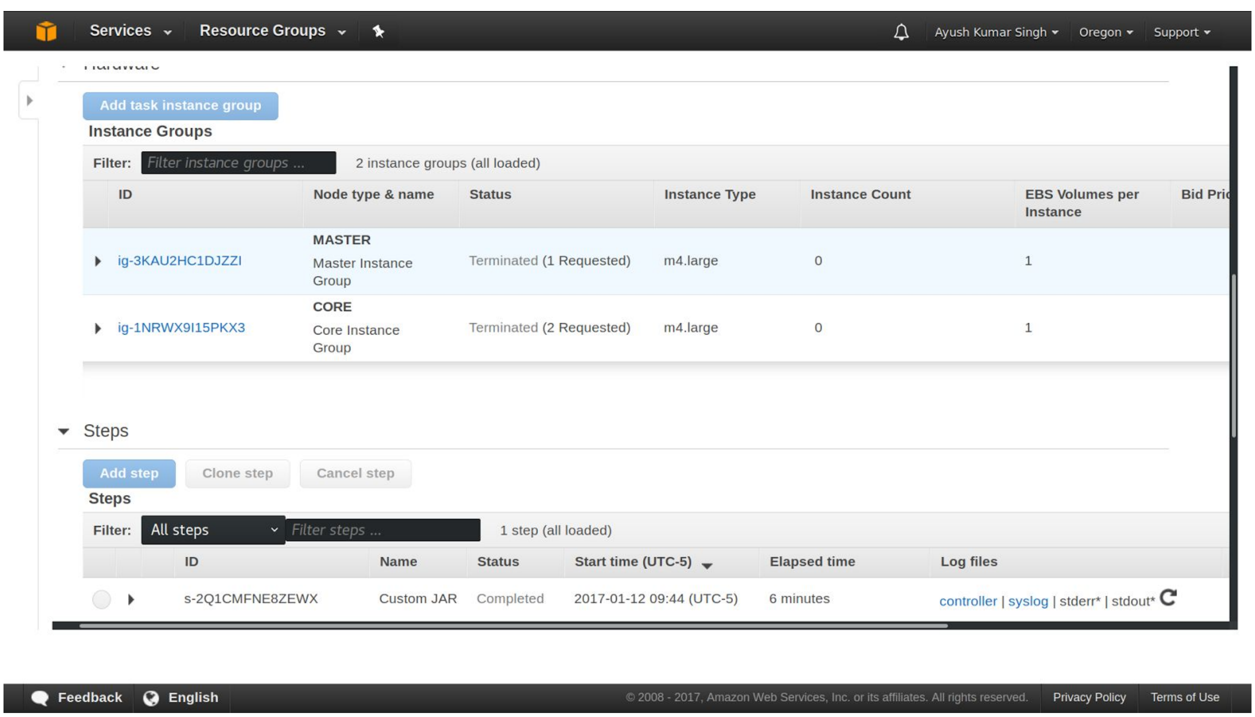
Word Count Local Execution

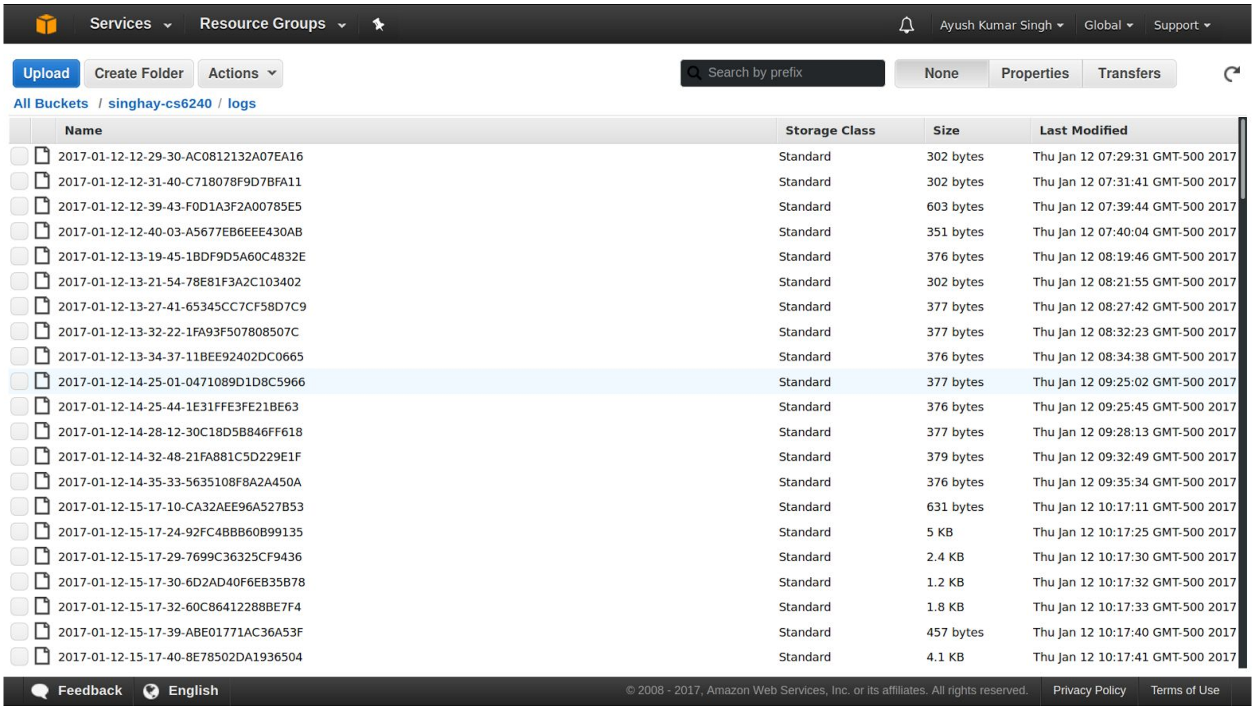


IDE: IntelliJ

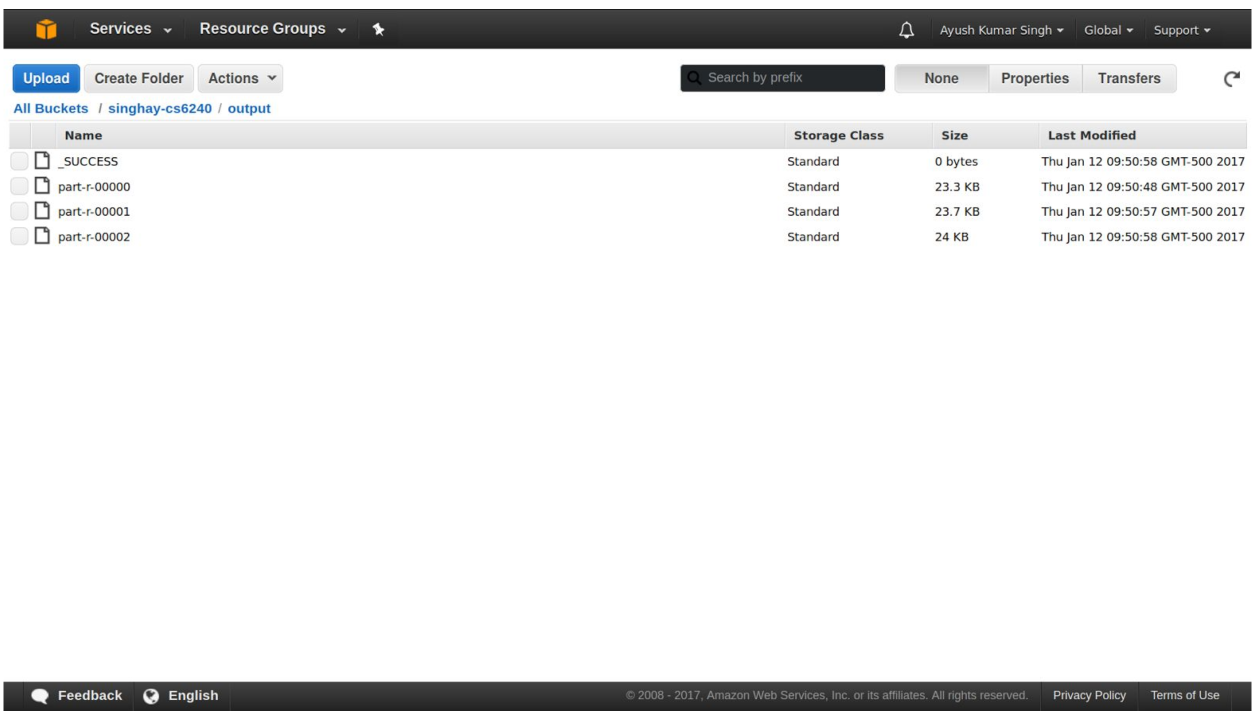
Word Count AWS Execution



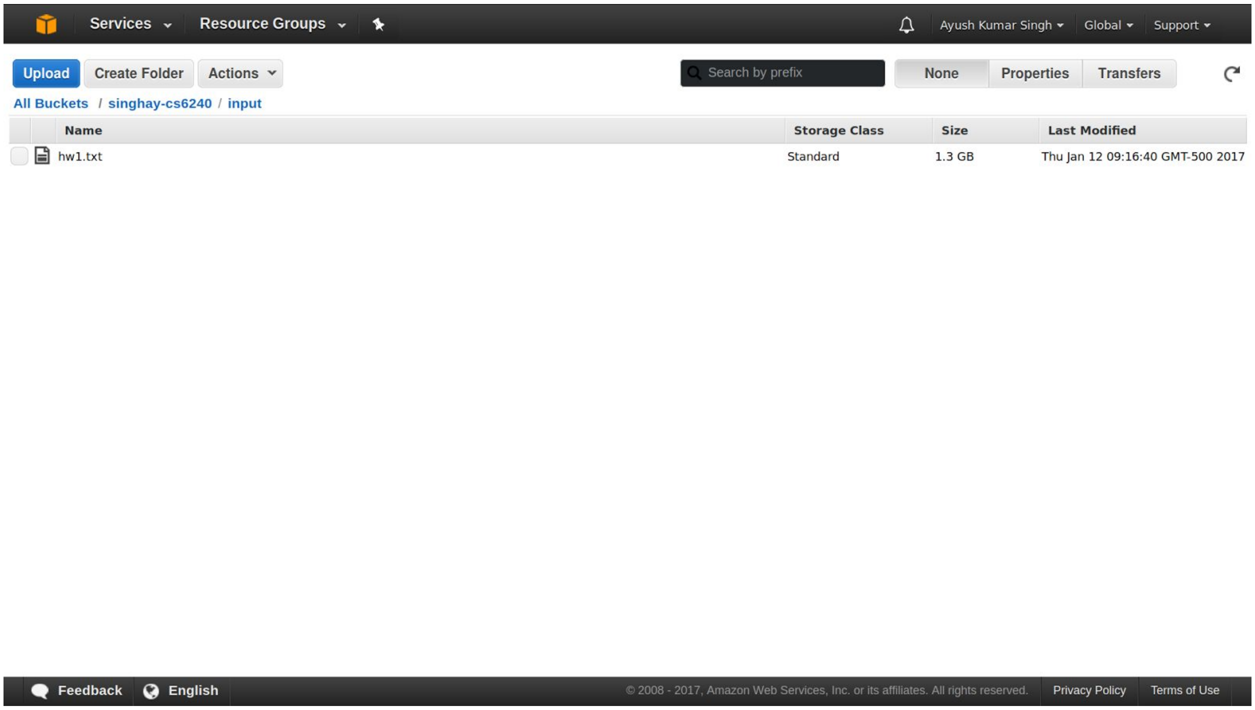


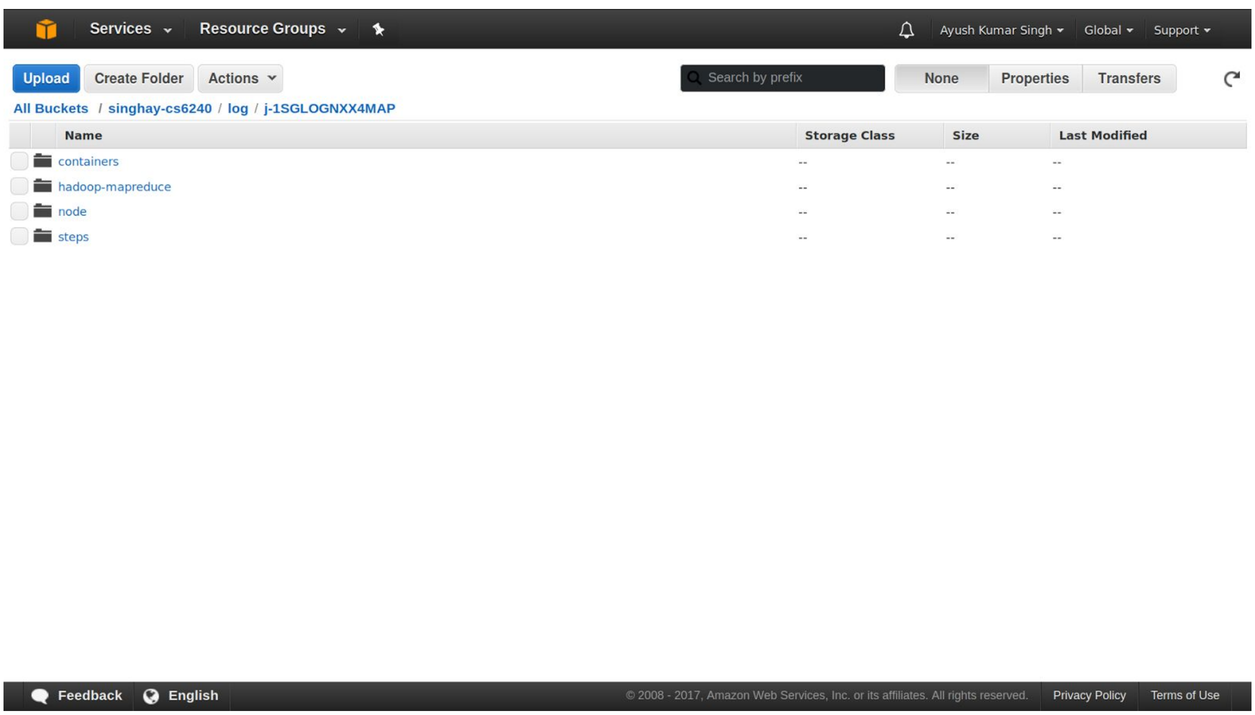


Name	Storage Class	Size	Last Modified
2017-01-12-12-29-30-AC0812132A07EA16	Standard	302 bytes	Thu Jan 12 07:29:31 GMT-500 2017
2017-01-12-12-31-40-C718078F9D7BFA11	Standard	302 bytes	Thu Jan 12 07:31:41 GMT-500 2017
2017-01-12-12-39-43-F0D1A3F2A00785E5	Standard	603 bytes	Thu Jan 12 07:39:44 GMT-500 2017
2017-01-12-12-40-03-A5677EB6EEE430AB	Standard	351 bytes	Thu Jan 12 07:40:04 GMT-500 2017
2017-01-12-13-19-45-1BDF9D5A60C4832E	Standard	376 bytes	Thu Jan 12 08:19:46 GMT-500 2017
2017-01-12-13-21-54-78E81F3A2C103402	Standard	302 bytes	Thu Jan 12 08:21:55 GMT-500 2017
2017-01-12-13-27-41-65345CC7CF58D7C9	Standard	377 bytes	Thu Jan 12 08:27:42 GMT-500 2017
2017-01-12-13-32-22-1FA93F507808507C	Standard	377 bytes	Thu Jan 12 08:32:23 GMT-500 2017
2017-01-12-13-34-37-11BEE92402DC0665	Standard	376 bytes	Thu Jan 12 08:34:38 GMT-500 2017
2017-01-12-14-25-01-0471089D1D8C5966	Standard	377 bytes	Thu Jan 12 09:25:02 GMT-500 2017
2017-01-12-14-25-44-1E31FFE3FE21BE63	Standard	376 bytes	Thu Jan 12 09:25:45 GMT-500 2017
2017-01-12-14-28-12-30C18D5B846FF618	Standard	377 bytes	Thu Jan 12 09:28:13 GMT-500 2017
2017-01-12-14-32-48-21FA881C5D229E1F	Standard	379 bytes	Thu Jan 12 09:32:49 GMT-500 2017
2017-01-12-14-35-33-5635108F8A2A450A	Standard	376 bytes	Thu Jan 12 09:35:34 GMT-500 2017
2017-01-12-15-17-10-CA32AEE96A527B53	Standard	631 bytes	Thu Jan 12 10:17:11 GMT-500 2017
2017-01-12-15-17-24-92FC4BBB60B99135	Standard	5 KB	Thu Jan 12 10:17:25 GMT-500 2017
2017-01-12-15-17-29-7699C36325CF9436	Standard	2.4 KB	Thu Jan 12 10:17:30 GMT-500 2017
2017-01-12-15-17-30-6D2AD40F6EB35B78	Standard	1.2 KB	Thu Jan 12 10:17:32 GMT-500 2017
2017-01-12-15-17-32-60C86412288BE7F4	Standard	1.8 KB	Thu Jan 12 10:17:33 GMT-500 2017
2017-01-12-15-17-39-ABE01771AC36A53F	Standard	457 bytes	Thu Jan 12 10:17:40 GMT-500 2017
2017-01-12-15-17-40-8E78502DA1936504	Standard	4.1 KB	Thu Jan 12 10:17:41 GMT-500 2017



Name	Storage Class	Size	Last Modified
_SUCCESS	Standard	0 bytes	Thu Jan 12 09:50:58 GMT-500 2017
part-r-00000	Standard	23.3 KB	Thu Jan 12 09:50:48 GMT-500 2017
part-r-00001	Standard	23.7 KB	Thu Jan 12 09:50:57 GMT-500 2017
part-r-00002	Standard	24 KB	Thu Jan 12 09:50:58 GMT-500 2017


Services
Resource Groups
Ayush Kumar Singh
Global
Support
Upload
Create Folder
Actions
Search by prefix
None
Properties
Transfers
All Buckets / singhay-cs6240 / input
Name
Storage Class
Size
Last Modified
hw1.txt
Standard
1.3 GB
Thu Jan 12 09:16:40 GMT-500 2017
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use


Services
Resource Groups
Ayush Kumar Singh
Global
Support
Upload
Create Folder
Actions
Search by prefix
None
Properties
Transfers
All Buckets / singhay-cs6240 / log / j-1SGLOGNXX4MAP
Name
Storage Class
Size
Last Modified
containers
hadoop-mapreduce
node
steps
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use