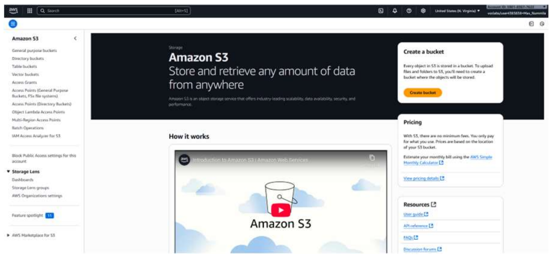
Max Nummila 2202236

Cloud Computing Assignment 4

I began by familiarizing myself with the different articles since this task was clearly a bit more complicated than the previous ones and had no clear instructions on where to begin with. I started by making a bucket with default settings in the Amazon S3 application.


Amazon S3

General purpose buckets
Directory buckets
Table buckets
Vector buckets
Access Grants
Access Points (General Purpose Buckets, FSx file systems)
Access Points (Directory Buckets)
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
AWS Marketplace for S3

Storage
Amazon S3
Store and retrieve any amount of data from anywhere
Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance.

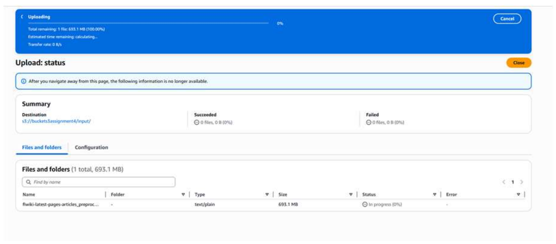
Create a bucket
Every object in S3 is stored in a bucket. To upload files and folders to S3, you'll need to create a bucket where the objects will be stored.
Create bucket

Pricing
With S3, there are no minimum fees. You only pay for what you use. Prices are based on the location of your S3 bucket.
Estimate your monthly bill using the AWS Simple Monthly Calculator
View pricing details

How it works
Introduction to Amazon S3 | Amazon Web Services
Amazon S3

Resources
User guide
API reference
FAQs
Discussion forums

I then created a folder for the text file and uploaded it to my bucket


Uploading
Total remaining: 1 file: 693.1 MB (100.00%)
Estimated time remaining: calculating...
Transfer rate: 0 B/s
0%
Cancel

Upload: status
Close
After you navigate away from this page, the following information is no longer available.

Summary
Destination: s3://buckets3assignment4/input/
Succeeded: 0 files, 0 B (0%)
Failed: 0 files, 0 B (0%)

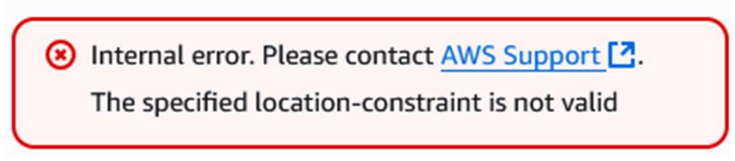
Files and folders | Configuration

Files and folders (1 total, 693.1 MB)

Name	Folder	Type	Size	Status	Error
fiwiki-latest-pages-articles_preproc...	-	text/plain	693.1 MB	In progress (0%)	-

The code from the two example websites was a good start, and with slight modifications to the mapper and an addition of the top 10 feature and the total count the reducer was ready. The mapper change was only an addition of a .lower and a change to the last line because the print statement was for python 2, not 3, and pycharm said that the print statement "seems to have no effect".

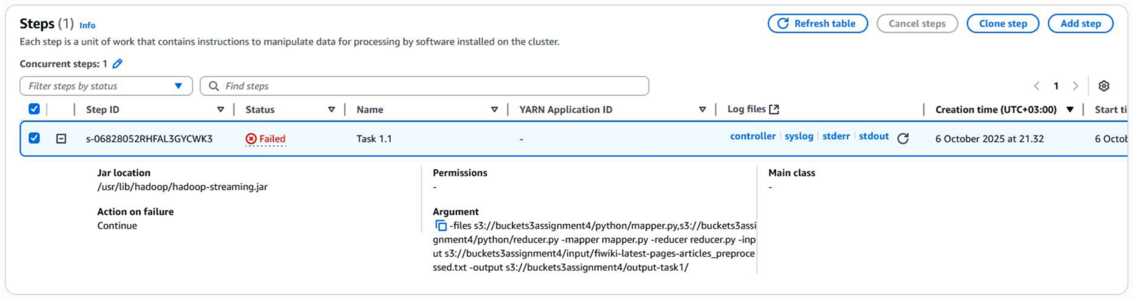
Next, I went to create an EMR cluster for which I chose the core Hadoop application bundle and fill in the rest on defaults and the m4.large ec2, but when I try to create it I get this error.


Internal error. Please contact AWS Support.
The specified location-constraint is not valid

This was because i missed the steps and the cluster logs which i got reminded of when opening the amazon resource for creating an emr. I used the bucket as the cluster logs link and created it with the hadoop application set.

I then tried to create a step but I used custom JAR at first instead of streaming program, so I had some issues.

Max Nummila 2202236


Steps (1) Info

Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.

Concurrent steps: 1

Step ID	Status	Name	YARN Application ID	Log files	Creation time (UTC+03:00)	Start ti
s-06828052RHFAL3GYCWK3	Failed	Task 1.1	-	controller | syslog | stderr | stdout	6 October 2025 at 21.32	6 Octob

Jar location
/usr/lib/hadoop/hadoop-streaming.jar

Action on failure
Continue

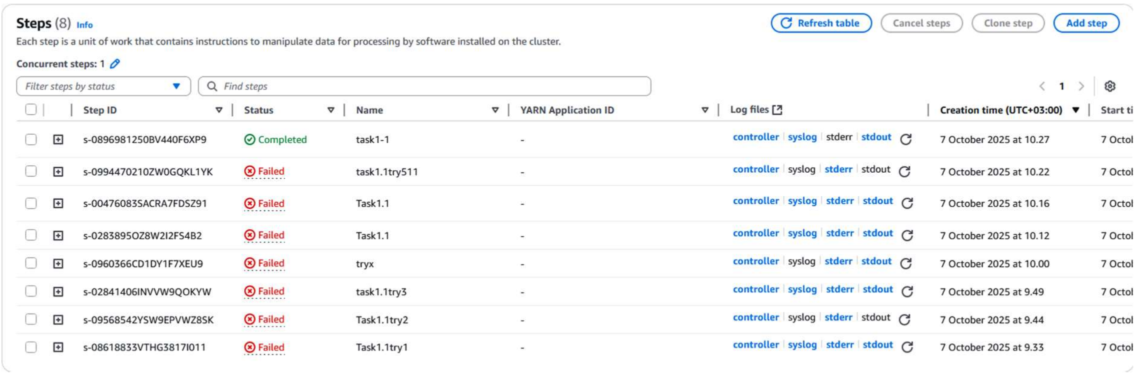
Permissions
-

Argument
```
-files s3://buckets3assignment4/python/mapper.py,s3://buckets3assignment4/python/reducer.py -mapper mapper.py -reducer reducer.py -input s3://buckets3assignment4/input/fiwiki-latest-pages-articles_preprocessed.txt -output s3://buckets3assignment4/output-task1/
```

Main class
-

After my first attempt I had an error, code 127. On the second one it was 403, which was a permission error. I decided to sleep on it and create a new cluster the next day.

As one can see from the screenshot below, I went mad once I got back and tried everything. Until I realized that my python files were of course written on windows, so they had the CRLF line endings instead of LF which works on linux. I am to be honest disappointed in myself but relieved that something worked.


Steps (8) Info

Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster.

Concurrent steps: 1

Step ID	Status	Name	YARN Application ID	Log files	Creation time (UTC+03:00)	Start ti
s-0896981250BV440F6XP9	Completed	task1-1	-	controller | syslog | stderr | stdout	7 October 2025 at 10.27	7 Octol
s-0994470210ZW0GQKL1YK	Failed	task1.1try511	-	controller | syslog | stderr | stdout	7 October 2025 at 10.22	7 Octol
s-00476083SACRA7FDSZ91	Failed	Task1.1	-	controller | syslog | stderr | stdout	7 October 2025 at 10.16	7 Octol
s-0283895OZ8W2I2FS4B2	Failed	Task1.1	-	controller | syslog | stderr | stdout	7 October 2025 at 10.12	7 Octol
s-0960366CD1DY1F7XEU9	Failed	tryx	-	controller | syslog | stderr | stdout	7 October 2025 at 10.00	7 Octol
s-02841406INVVW9QOKYW	Failed	task1.1try3	-	controller | syslog | stderr | stdout	7 October 2025 at 9.49	7 Octol
s-09568542YSW9EPVWZ8SK	Failed	Task1.1try2	-	controller | syslog | stderr | stdout	7 October 2025 at 9.44	7 Octol
s-08618833VTHG3817I011	Failed	Task1.1try1	-	controller | syslog | stderr | stdout	7 October 2025 at 9.33	7 Octol

This did not work however since there were seven different result files all with different words so I googled and tried again with the argument -numReduceTasks 1 so that it would hopefully give me one file.

I didn't get the desired total count because I had accidentally indented it so it was in an exception, but after this fix I got a part-0000 file which had this:

```
TOTAL_WORDS	80260973


TOP_10_WORDS:

ja	3008151

on	1850825

oli	870033

hän	728579
```

vuonna	610725
myös	448704
joka	355687
se	309625
ovat	275568
sekä	267504

Mapper.py

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip().lower()
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

Reducer.py

```
#!/usr/bin/env python3
import sys

currentWord = None
currentCount = 0
wordCounts = {}
totalWords = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    try:
        word, count = line.split("\t", 1)
        count = int(count)
    except ValueError:
        continue

    totalWords += count
    if currentWord == word:
        currentCount += count
    else:
        if currentWord is not None:
            wordCounts[currentWord] = wordCounts.get(currentWord, 0) + currentCount
        currentWord = word
        currentCount = count

if currentWord is not None:
    wordCounts[currentWord] = wordCounts.get(currentWord, 0) + currentCount

print(f"TOTAL_WORDS\t{totalWords}")

print("\nTOP_10_WORDS:")
for word, count in sorted(wordCounts.items(), key=lambda x: x[1], reverse=True)[:10]:
    print(f"{word}\t{count}")
```

The mapper reads from stdin line by line normalizes case with .lower(), splits on whitespace, and emits one tab-separated key\tvalue record per token: word\t1 (Hadoop Streaming expects tab-delimited pairs). The reducer consumes mapper output, parses word\tcount, accumulates a running total per word, and also updates a global TOTAL_WORDS by adding every count. At the end it prints the total words and the top 10. Output is a single part-00000 because I ran with one reducer "-numReduceTasks 1".

So finally, it was on to task 1-2

I made changes to the mapper to make it work with a compiler and the result was the same words and the same amounts, but the elapsed time for the task was 2 minutes and 58 seconds, compared to task 1-1 which had an elapsed time of 4 minutes and 40 seconds.

This was my mapper:

```
#!/usr/bin/env python3

import sys

counts = {}
```

```
FLUSH_EVERY_N_LINES = 10000

for i, line in enumerate(sys.stdin, 1):

  for w in line.lower().split():

    counts[w] = counts.get(w, 0) + 1

  if i % FLUSH_EVERY_N_LINES == 0:

    for k,v in counts.items(): print(f"{k}\t{v}")

    counts.clear()

if counts:

  for k,v in counts.items(): print(f"{k}\t{v}")
```

The changes made were so that it would not emit word\t1 for each token but to instead aggregate in a python dict and flush the partial sums. Counts holds the local word, which is the count for the current flush window. The first for loop streams the input and counts the lines to know when to flush and the second for loop again normalizes whitespaces and the case. The first if statement makes the code emit one record per distinct word with its local sum, and clears the counts to free memory, every N line.

The final if statement is to block flushes from having partials at the end of the input.

Task 1-3

I once again made changes to my mapper and then ran them both which took 2 minutes and 14 seconds. I got a total count of about 20% of the original one and the top 10 was only 3 letter words, so it seems it worked.

Code:

Mapper.py

```
#!/usr/bin/env python3
import sys, re

tok = re.compile(r"\w+", re.UNICODE)

for line in sys.stdin:
  for w in tok.findall(line.lower()):
    if w.isalpha() and 3 <= len(w) <= 5:
      print(f"{w}\t1")
```

Tokenizes with a regex \w+, filters to alphabetic tokens (isalpha()) whose length is 3–5 and emits word\t1 only for those. This excludes numbers and any tokens shorter than 3 or longer than 5.

2.1

I had to create a new cluster because my first one had auto terminated.

I created it with the same bucket, to which I just uploaded the second file and my new mapper and reducer python files which I used when creating the step.

I ran it with a new mapper and reducer (mapper2 and reducer2) and got this result:

```
TOTAL_REQUESTS	3461580

TOTAL_BYTES	65524307881

TOTAL_GiB	61.024267

COST_REQUESTS_EUR	3461.580000

COST_DATA_EUR	4.881941

COST_TOTAL_EUR	3466.461941
```

Reducer2.py

```
#!/usr/bin/env python3
import sys

req = 0
totalBytes = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    try:
        key, val = line.split("\t", 1)
        v = int(val)
    except ValueError:
        continue
    if key == "REQ":
        req += v
    elif key == "BYTES":
        totalBytes += v

KB = 1024
MB = 1024 * KB
GB = 1024 * MB
totalGib = totalBytes / GB

costReqs = req * 0.001
costData = totalGib * 0.08
totalCost = costReqs + costData

print(f"TOTAL_REQUESTS\t{req}")
print(f"TOTAL_BYTES\t{totalBytes}")
print(f"TOTAL_GiB\t{totalGib:.6f}")
print(f"COST_REQUESTS_EUR\t{costReqs:.6f}")
print(f"COST_DATA_EUR\t{costData:.6f}")
print(f"COST_TOTAL_EUR\t{totalCost:.6f}")
```

Mapper2.py

```
#!/usr/bin/env python3
import sys, re

LOG = re.compile(r'^(?P<host>\S+) \S+ \S+ \[[^\]]+\] "(?:[^"]*)" \d{3} (?P<bytes>\S+)')

for line in sys.stdin:
    m = LOG.match(line)
    if not m:
        continue
    b = m.group("bytes")
    try:
        bytesOut = int(b) if b != "-" else 0
    except ValueError:
        bytesOut = 0

    print(f"REQ\t1")
    print(f"BYTES\t{bytesOut}")
```

The mapper reads each apache log line and counts 1 request and adds its byte size. It does this by parsing the client host and the bytes field with regex. If the bytes are malformed or a – it will treat them as 0, and at the end it emits two tab-separated records per line. These are a contribution to the total request count and a contribution to the total bytes. The reducer then reads each key\tvalue and converts bytes to GiB, and uses the pricing to calculate the costs and to print them with labels.

For the submission I created multiple mappers even though I just modified the one I had each time during the task, the reducer was the same for the 1.x tasks. Doing this task has shown and thought me a lot about the buckets and clusters, and I found it quite easy to do once I had understood everything. That has really been the entire experience with aws academy, everything is straight forward when you know what you want, and the worst problems were when using the open-source testing website, which was not an aws problem.

I ran out of time since I was sick, so I focused on completing the first part and the first task in part 2 and making the documentation on it clear.