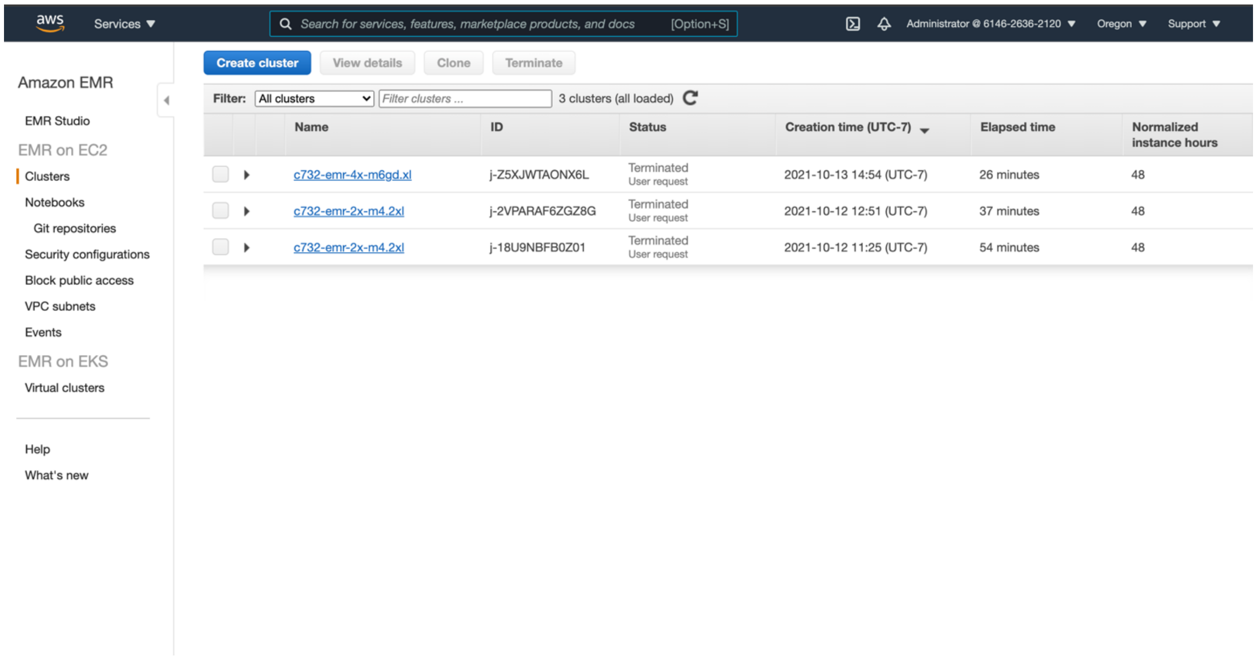
1. Take a screen shot of your list of EMR clusters (if more than one page, only the page with the most recent), showing that all have `Terminated` status.



Name	ID	Status	Creation time (UTC-7)	Elapsed time	Normalized instance hours
c732-emr-4x-m6gd.xl	j-Z5XJWTAONX6L	Terminated User request	2021-10-13 14:54 (UTC-7)	26 minutes	48
c732-emr-2x-m4.2xl	j-2VPARAF6ZGZ8G	Terminated User request	2021-10-12 12:51 (UTC-7)	37 minutes	48
c732-emr-2x-m4.2xl	j-18U9NBFB0Z01	Terminated User request	2021-10-12 11:25 (UTC-7)	54 minutes	48

2. For Section 2:
   a. What fraction of the input file was prefiltered by S3 before it was sent to Spark?

      Input file size in regular version and S3 version was 2.6 MB and 97.7 KB respectively, which means S3 has prefiltered 96.33% of the input file.

   b. Comparing the different input numbers for the regular version versus the prefiltered one, what operations were performed by S3 and which ones performed in Spark?

      The input number for the regular version was 293571 whereas the input number for the prefiltered one was 3245 and 3245 was exactly the output number for the regular version. This means that S3 did all the data filtering and column selection before it was sent to Spark. In other words, S3 has performed the below lines of code in weather_etl.py:

```
    filter_qflag = weather.filter(weather.qflag.isNull())
    canadian_data = filter_qflag.filter(weather.station.startswith('CA'))
    max_temp = canadian_data.filter(weather.observation == 'TMAX')
```

      Spark then read in the filtered data as input and performed calculation and wrote the output as json format, which corresponding to the following lines of code:

```
    etl_data = max_temp.select('station', 'date', (max_temp.value/10).alias('tmax'))
    etl_data.write.json(output, compression='gzip', mode='overwrite')
```

3. For Section 3:
    a. Reviewing the job times in the Spark history, which operations took the most time? Is the application IO-bound or compute-bound?

       The 'reduceByKey' job took the most time. As this job involves reading in and shuffling data, this application is IO-bound.

    b. Look up the hourly costs of the `m6gd.xlarge` instance on the EC2 On-Demand Pricing page. Estimate the cost of processing a dataset ten times as large as `reddit-5` using just those 4 instances. If you wanted instead to process this larger dataset making full use of 16 instances, how would it have to be organized?

       The hourly rate for m6gd.xlarge is $0.1808. Processing reddit-5 took 9 mins in total. Therefore, it will take approximately 90 mins to process a ten times large dataset, with 4 instances, it will cost about $0.1808/h * 90 min/60s*4 = $ 1.0848.

       16 instances * 4 cores = 64 executors

       We can repartition the dataset to n*64 partitions, where n can be 2 or 3.