Algorithmic Methods of Data Mining

Elastic Map Reduce

Ioannis Chatzigiannakis

Sapienza University of Rome

Laboratory 7

AWS Elastic Map Reduce

- Managed Hadoop framework on EC2 instances.
- AWS EMR splits large processing jobs into smaller jobs and distributes them across many compute nodes in a Hadoop cluster.
- Easily run and scale open-source big data frameworks:
  - Apache Spark
  - Apache Flink
  - Apache Hive
  - Presto
  - Apache HBase
  - ...
- EMR Notebooks.

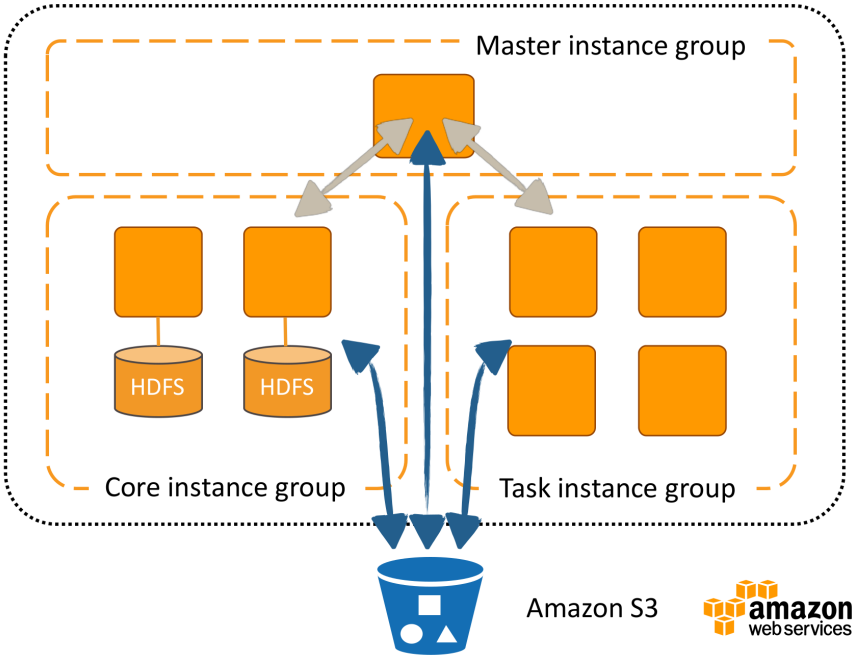
EMR: Benefits

- Easy to use – interact using Jupyter via web.
- Low cost
  - Pay a per-instance rate for every second used, with a one-minute minimum charge.
- Elastic
  - For short-running jobs, you can spin up and spin down clusters and pay per second for the instances used.
  - For long-running workloads, you can create highly available clusters that automatically scale to meet demand.
- Reliable
- Secure
- Flexible

AWS EMR Ideal Usage Patterns

- Logprocessing and analytics
- Large extract, transform, and load (ETL) data movement
- Risk modeling and threat analytics
- Ad targeting and click stream analytics
- Genomics
- Predictive analytics
- Adhoc data mining and analytics

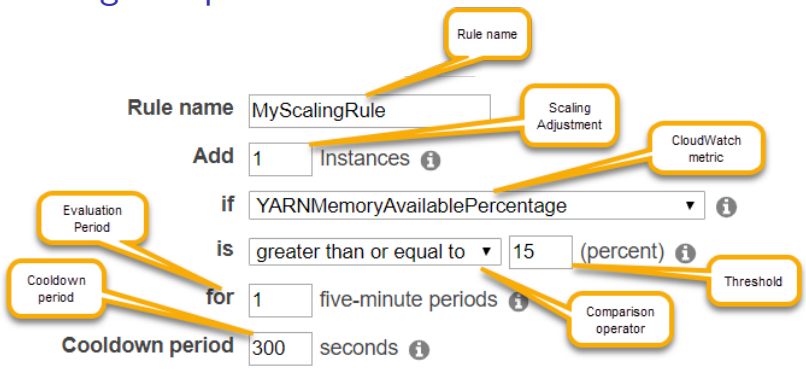
Apache EMR Architecture



AWS EMR and Apache Hadoop

- The Elastic Map Reduce is built on top Apache Hadoop.
- An open-source Java software framework that supports massive data processing across a cluster of instances.
- Distributed processing across the instances that make up the cluster.
- It can run on a single instance or thousands of instances.
- Elastic auto-scaling of cluster.
- Provides a fault-tolerant processing environment.

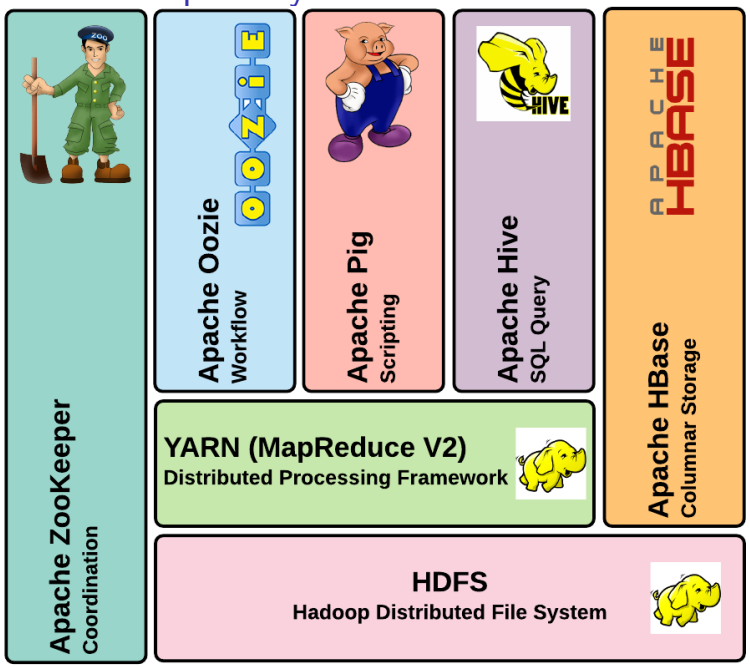
Auto-scaling rule parameters



Apache Hadoop

- Apache Hadoop includes the following modules:
  - Hadoop Common: The common utilities that support the other Hadoop modules.
  - Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
  - Hadoop YARN: A framework for job scheduling and cluster resource management.
  - Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.
  - Hadoop Ozone: An object store for Hadoop.

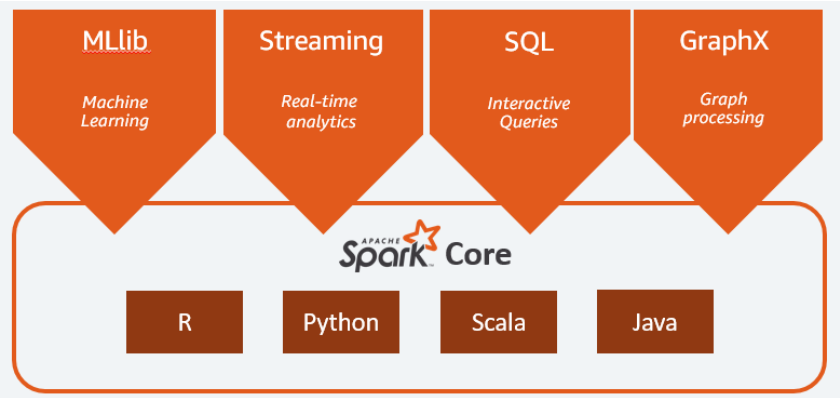
Apache Hadoop Ecosystem



Apache Spark on AWS EMR

- Started in 2009 as a research project at UC Berkley's AMPLab.
- An open-source, distributed processing system used for big data workloads.
- In contrast to Hadoop, uses in-memory caching to achieve high speed-ups.
  - Optimized query execution for fast analytic queries against data of any size.
- Development APIs in Java, Scala, Python and R.
- Supports code reuse across multiple workloads-batch processing:
  - interactive queries, real-time analytics, machine learning, and graph processing.

Apache Spark Workloads

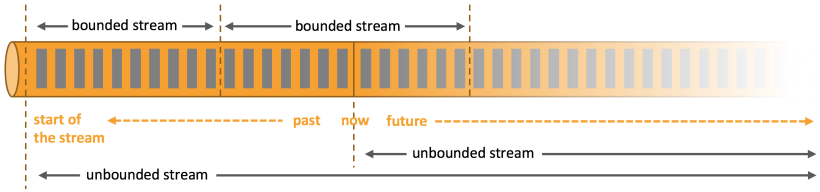


Apache Flink on AWS EMR

- Started in 2009 as a research project by the Berlin-based database research groups.
- An open-source, fast, general purpose distributed data processing system.
- Streaming dataflow engine that you can use to run real-time stream processing on high-throughput data sources
- Stateful computations over unbounded and bounded data streams.
- Up to 100x faster than Hadoop.
- Programming APIs for Java and Scala.

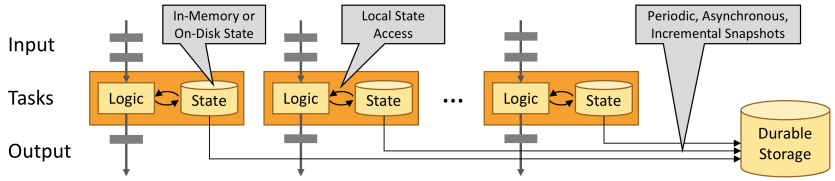
Process Unbounded and Bounded Data

- Any kind of data is produced as a stream of events.
  - Credit card transactions, sensor measurements, machine logs, user interactions on a website or mobile application, . . .
- Unbounded streams have a start but no defined end.
- Bounded streams have a defined start and end.

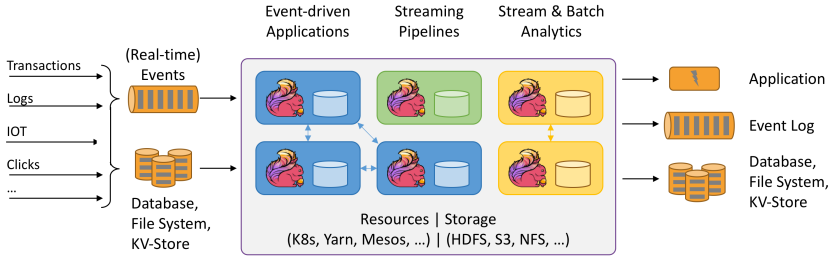


Leverage In-Memory Performance

- Stateful Flink applications are optimized for local state access.
- Task state is always maintained in memory or,
- if the state size exceeds the available memory, in access-efficient on-disk data structures.
- Flink guarantees exactly-once state consistency in case of failures.



Apache Flink Ecosystem



Apache Hive on AWS EMR

- Open-source, data warehouse, and analytic package that runs on top of a Hadoop cluster.
- Hive scripts use an SQL-like language called Hive QL.
  - Abstracts programming models,
  - supports typical data warehouse interactions.
- A command line tool and JDBC driver are provided to connect users to Hive.

Presto on AWS EMR

- Open source distributed SQL query engine for running interactive analytic queries.
- Data sources of all sizes: from gigabytes to petabytes.
- A single Presto query can combine data from multiple sources, allowing for analytics across your entire organization.
- Interactive Queries makes it easy for developers and data scientist to work with the big data.
- Supported by Linux Foundation.

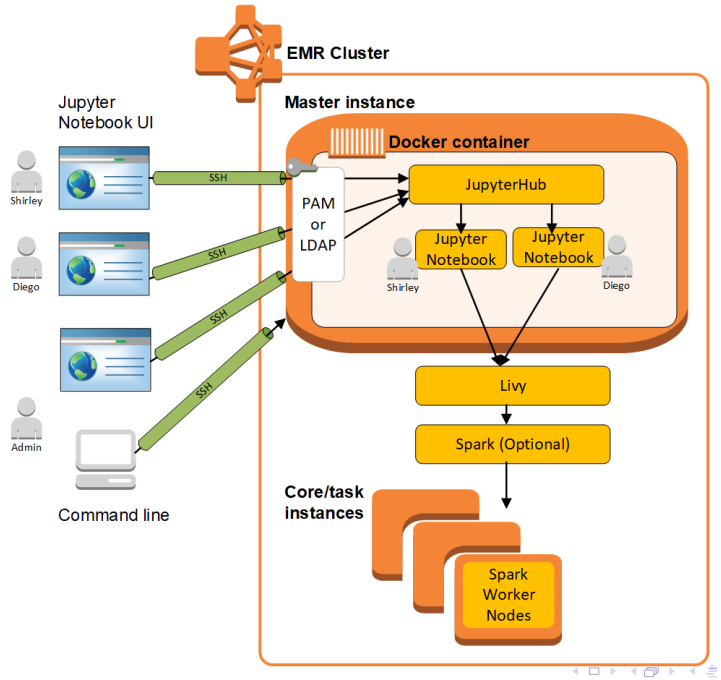
Apache HBase on AWS EMR

- An open source, distributed, versioned, non-relational database modeled after Google's BigTable.
- Enables random, realtime read/write access to Big Data.
  - Strictly consistent reads and writes.
- Allows hosting of very large tables – billions of rows X millions of columns – atop clusters of commodity hardware.
- Easy to use Java API for client access.
- Extensible jruby-based (JIRB) shell.

EMR Notebooks

- EMR Notebooks is a Jupyter Notebook environment built in to the Amazon EMR console.
- Quickly create Jupyter notebooks, attach them to Spark clusters
- Use Jupyter Notebook editor to remotely run queries and code.
- Open, attach multiple notebooks to a single cluster, and re-use a notebook on different clusters.
- You can start a cluster, attach an EMR notebook for analysis, and then terminate the cluster.
- You can also close a notebook attached to one running cluster and switch to another.
- Multiple users can attach notebooks to the same cluster simultaneously.

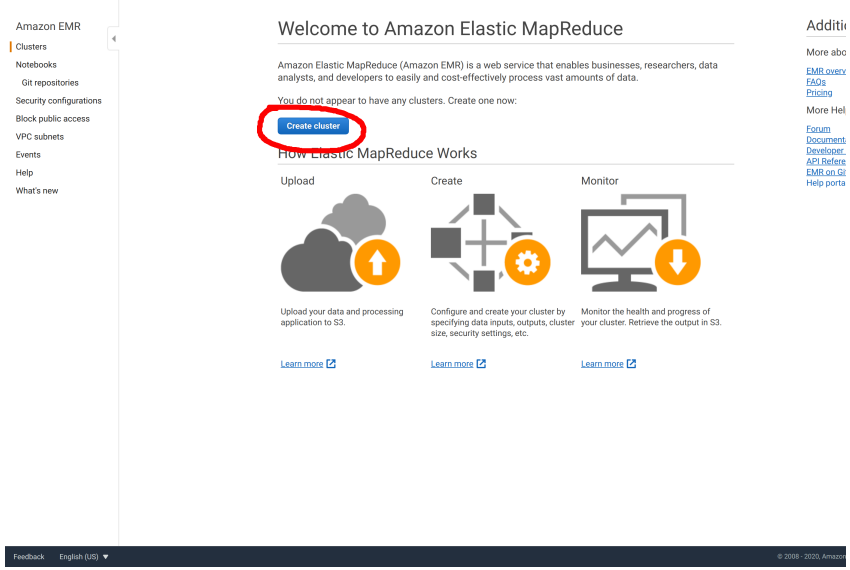
EMR Notebooks architecture



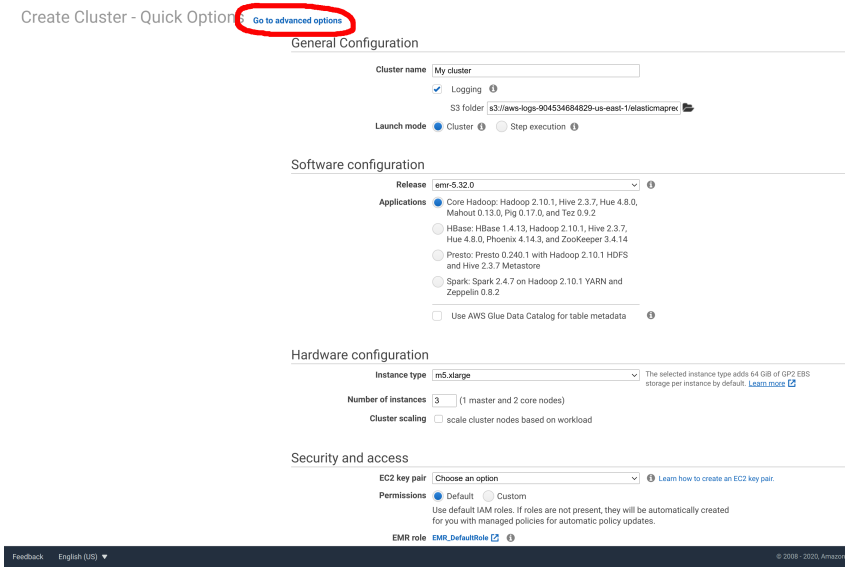
Considerations when using EMR Notebooks

- User notebooks and files are saved to the file system on the master node.
- This is ephemeral storage that does not persist through cluster termination.
- When a cluster terminates, this data is lost if not backed up.
- EMR Notebooks support persistance to S3.
- EMR Notebooks supports connection with GitHub repositories.

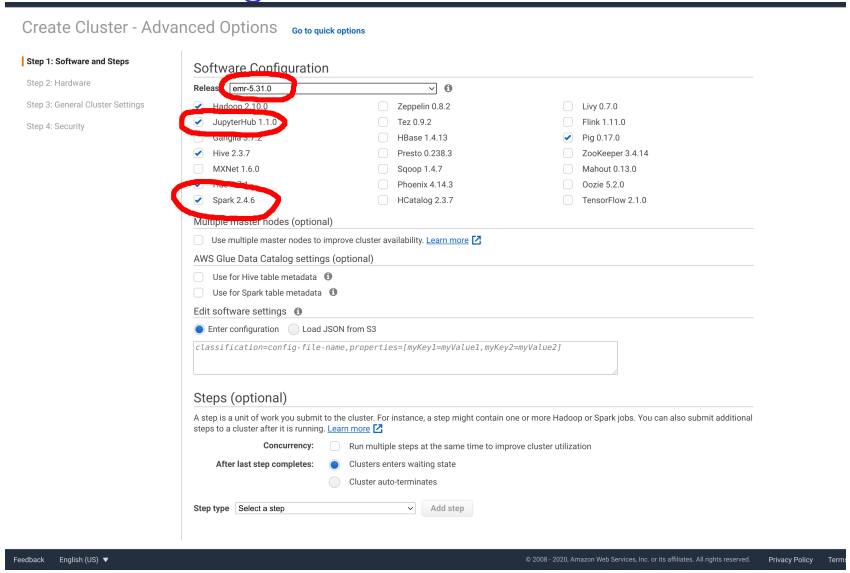
Create Cluster



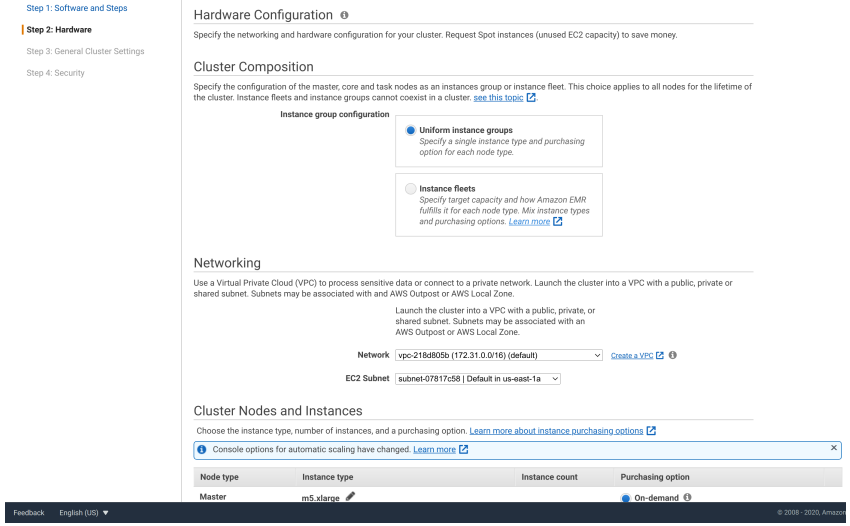
Quick Options



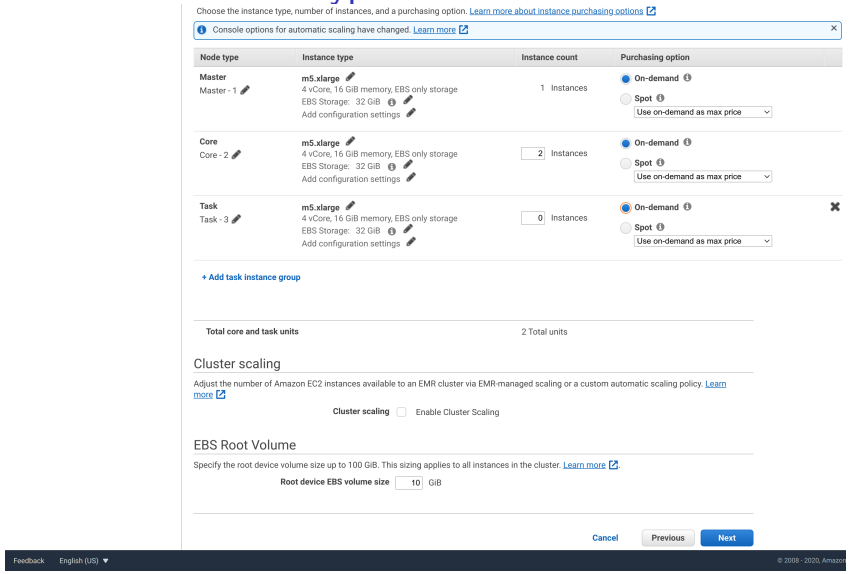
Software Configuration



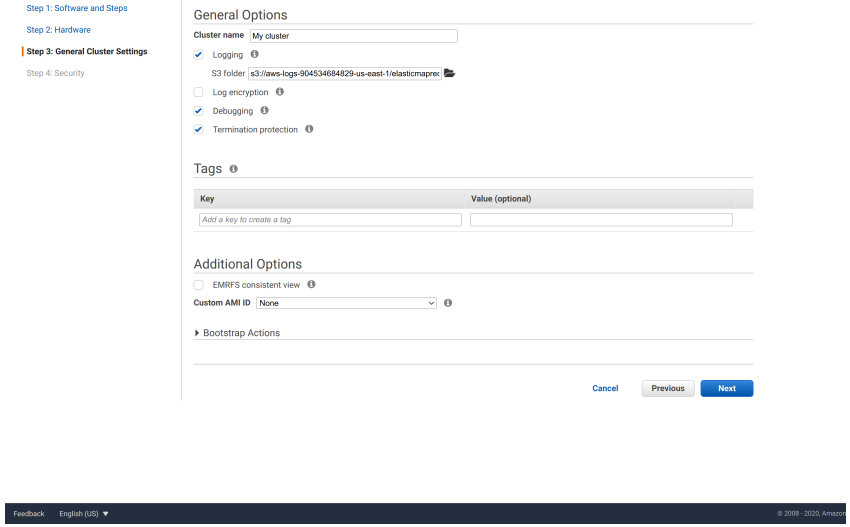
Hardware Configuration



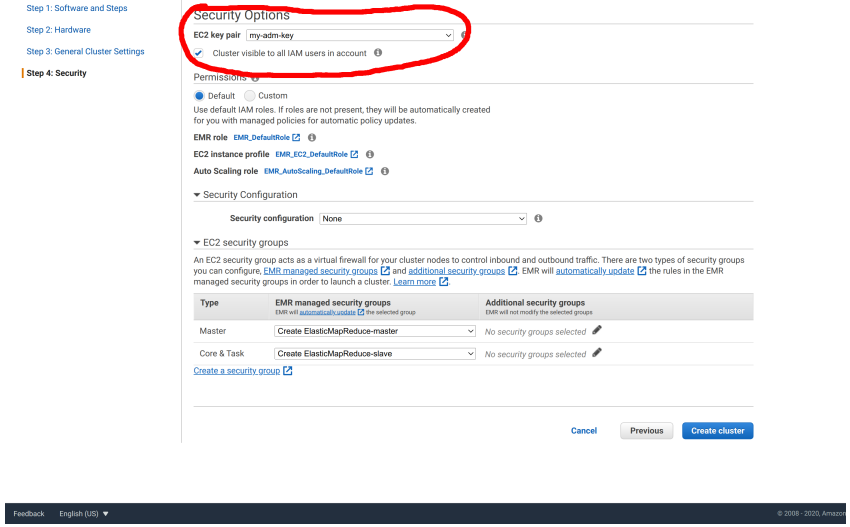
Choose Instance Types



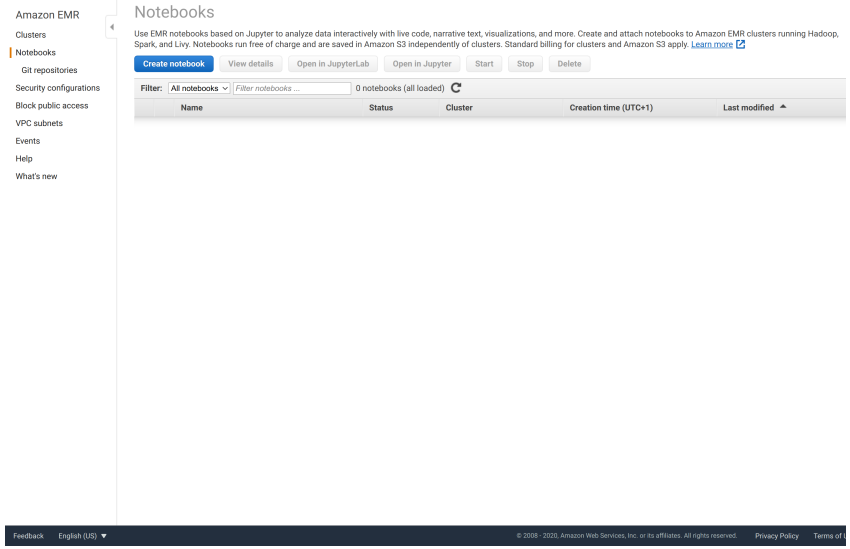
General Cluster Settings



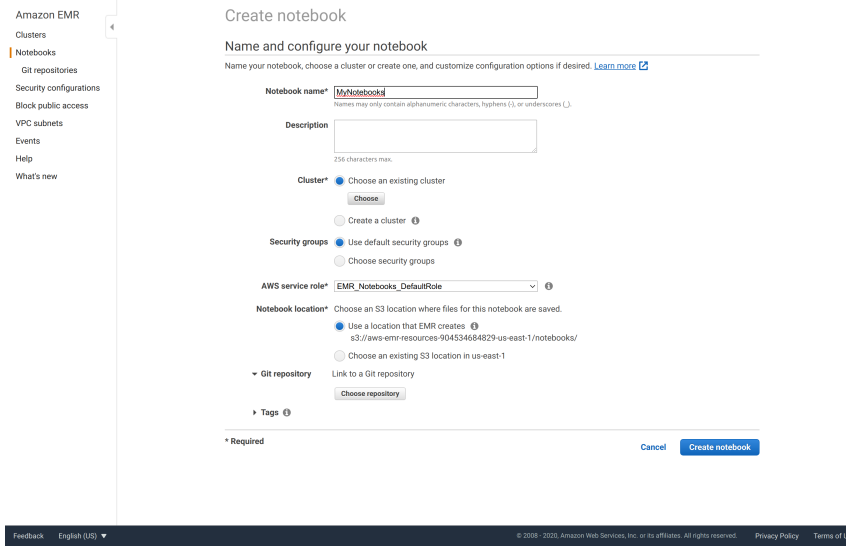
Security Options



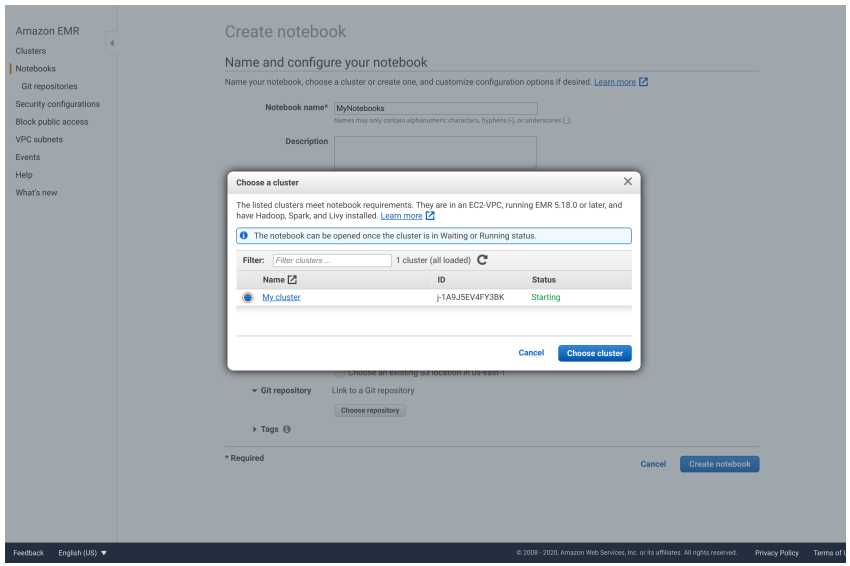
EMR Notebooks



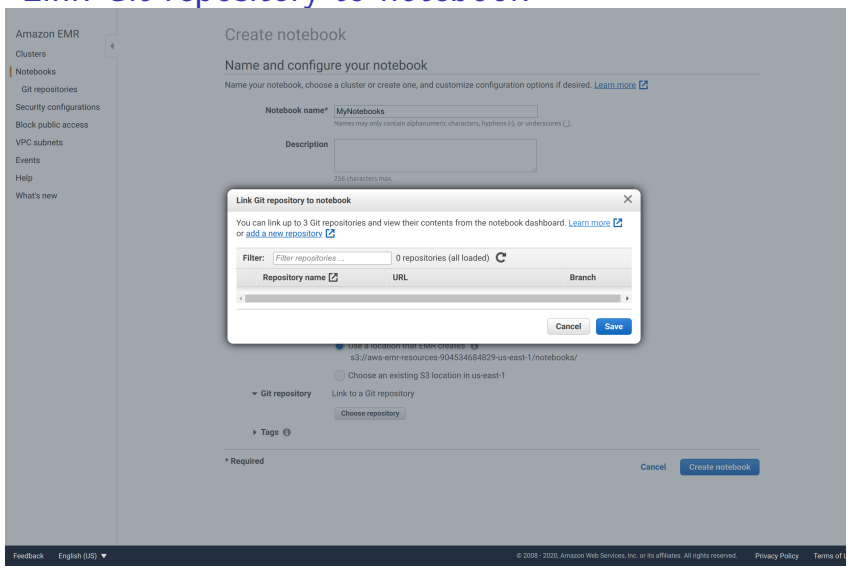
Create notebook



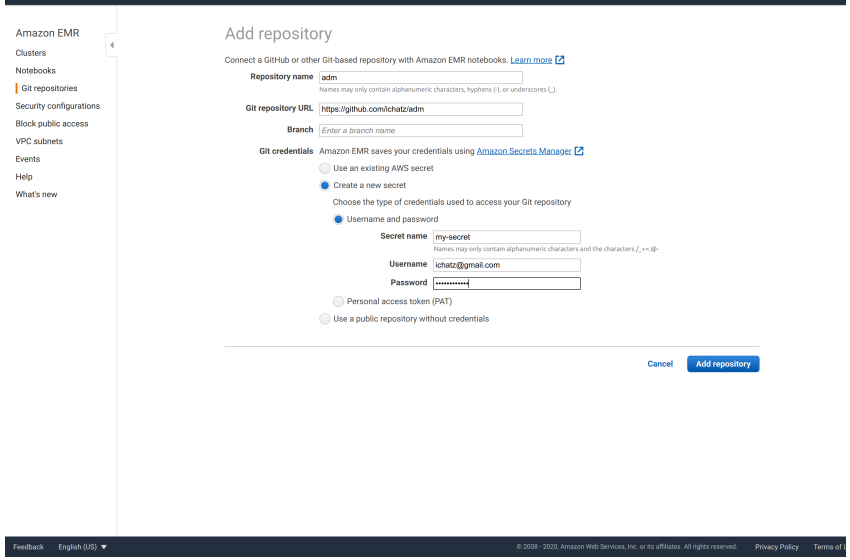
Choose a cluster



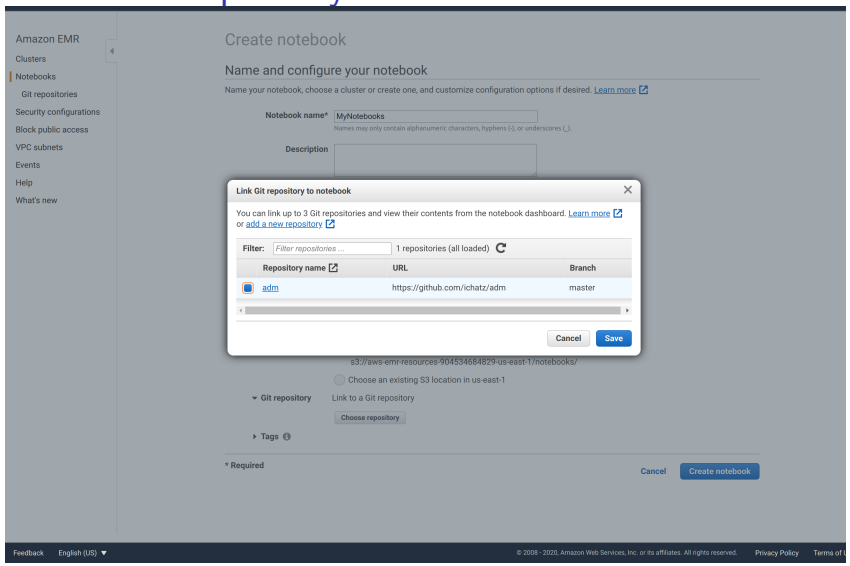
Link Git repository to notebook



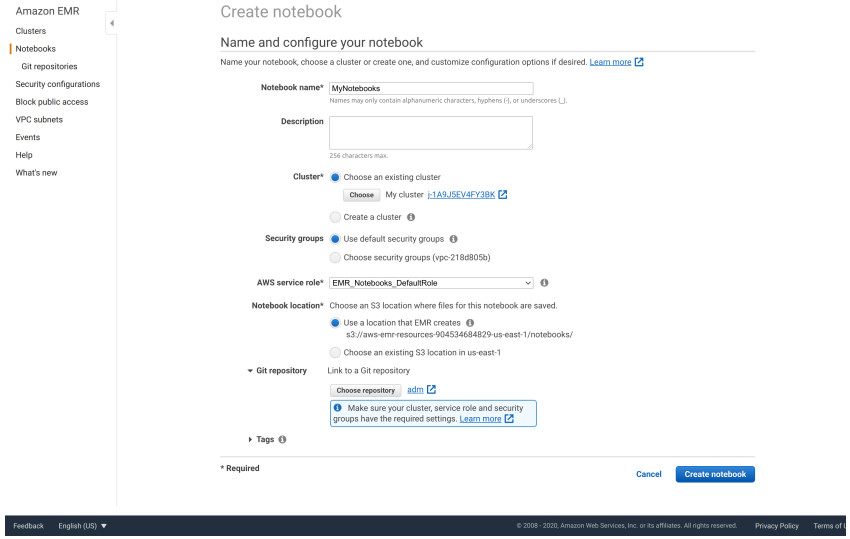
Add repository



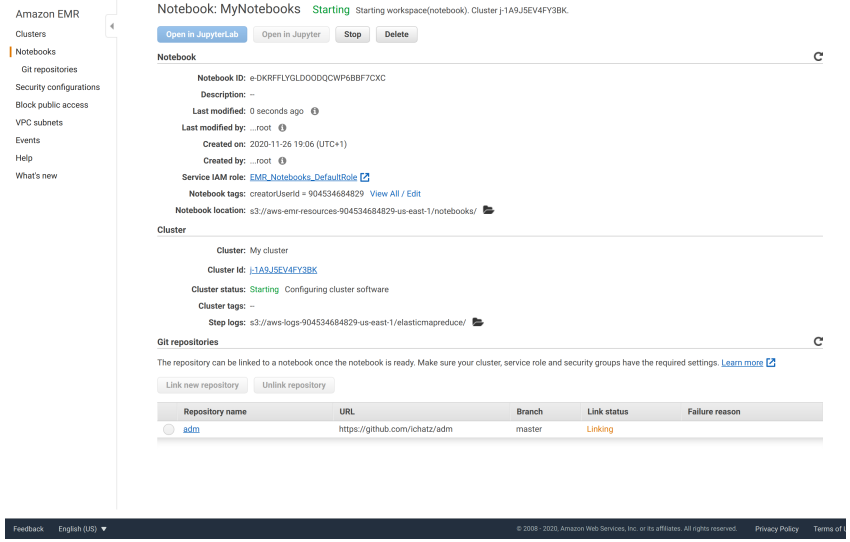
Link Git repository to notebook



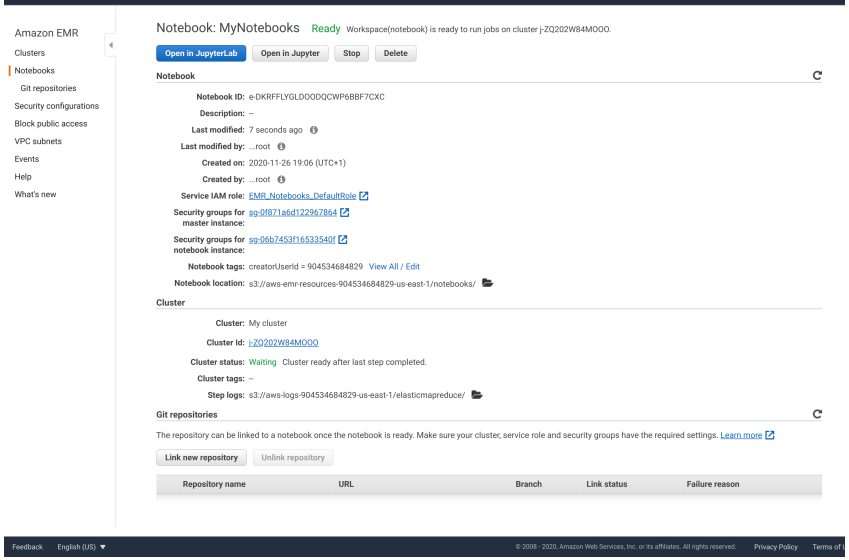
Create notebook



Starting notebook

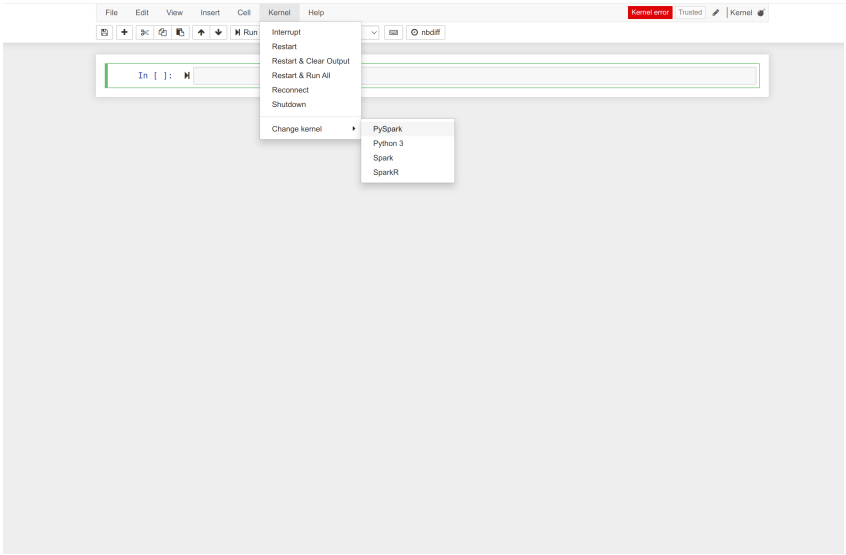


EMR Notebooks: ready

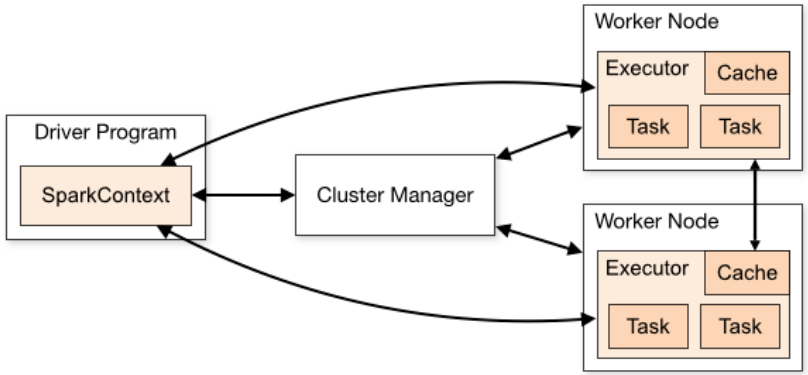


Jupyter Tree

Choose PySpark Kernel



Spark Driver Program



- Map/Reduce operations are issued to the cluster manager.
- Map/Reduce operations work on a given dataset.
- The dataset is encoded using the RDD structure.

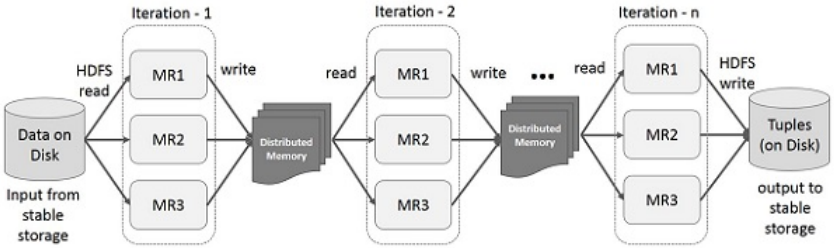
Spark Context

- SparkContext is the entry point to any spark functionality.
- A SparkContext represents the connection to a Spark cluster.
- Used to create RDD and broadcast variables on that cluster.
- Only one SparkContext should be active per session.

Resilient Distributed Datasets

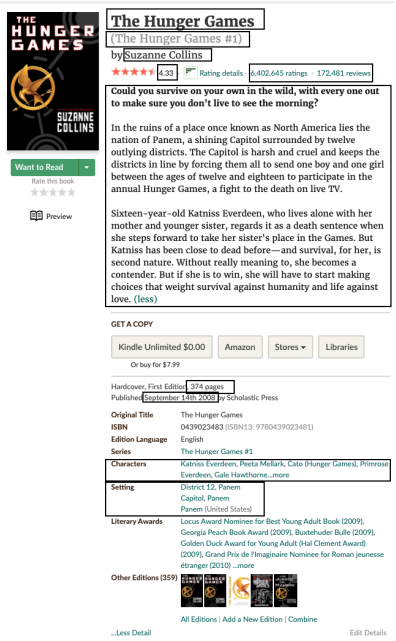
- A fundamental data structure of Spark.
- Spark makes use RDD to achieve faster and efficient MapReduce operations.
- An immutable distributed read-only collection of objects.
  - immutable = state cannot change after it is constructed.
  - Can contain any type of Python, Java, or Scala objects, including user-defined classes.
- Two ways to construct an RDD:
  1. Referencing a dataset in an external storage system: S3, HDFS, HBase, ...
  2. Through Map/Reduce opreations.
- RDD is divided into logical partitions.
  - Each logical partition may be computed on different nodes of the cluster.

Iterative Operations on MapReduce



- Reuse intermediate results across multiple computations in multi-stage applications.
- Each Map/Reduce operation works on a given/input RDD.
- Each Map/Reduce operation constructs/outputs a new RDD.
- If the Distributed memory (RAM) is not sufficient to store intermediate RDD, then it will store those results on the disk.

Book Dataset



Book Plot Summaries Dataset: A subset

- Today we will work with the CMU Book Summary Corpus.
- A collection of 16,559 book plot summaries extracted from Wikipedia:
  1. Wikipedia article ID
  2. Freebase ID
  3. Book title
  4. Author
  5. Publication date
  6. Book genres (Freebase ID:name tuples)
  7. Plot summary
- A tab-delimited text file is a file containing tabs that separate information with one record per line.
- Retrieve file `https://sapienza2020adm.s3.eu-central-1.amazonaws.com/booksummaries.txt.gz`
- Upload it to the S3 bucket associated with the EMR cluster.

Retrieve book dataset

- Use the variable sc to access the Spark Context.
- Load the Tab-delimited text file to Spark.

```
books = sc.textFile("s3://.../booksummaries.tar.gz")
```

- textFile Creates a new RDD object.
- Provides a reference to the dataset - no data loaded yet.

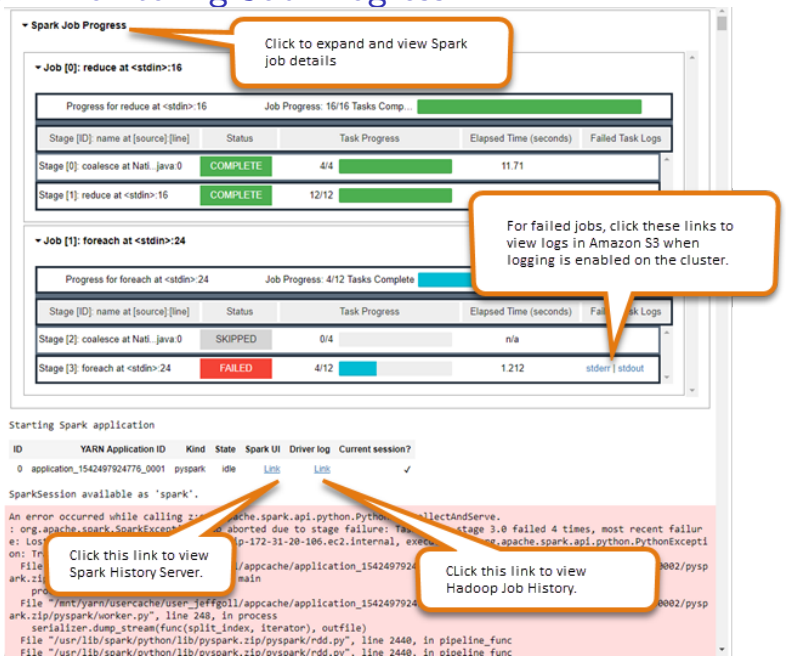
```
books.count()
```

- Retrieves the dataset from s3 and uncompress it.
- Creates 1 entry in the RDD for each line of the text file.
- Counts the entries in the RDD object.

```
books.take(5)
```

- Take the first 5 elements of the RDD.

Spark Monitoring Job Progress



Yearly statistics: Books per Year

- Identify how many books are published per year.
- Retrieve the 5th field from each record.
- Count repetitions of each year.
- RDD Operations used:
  1. Map – extract the 5th field.
  2. FlatMap – extract each word from the title.
  3. Map – convert each word to a tupple (word, 1).
  4. ReduceByKey – count appearances of each (word, *) tupple.

RDD operations – Publication date

```
book_dates = books.map(lambda a: a.split('\t')[4])
```

- Return a new RDD by applying a function to each element of this RDD.
- 1-to-1 mapping: For each original record, a new record will be generated.
- Each line is 1 record: fields are separated by tab characters.
- We split the line using the tab character, take the 4th field: the publication date.
- The new RDD contains only the publication dates of the books.
- Execution of the Map operation is delayed.

RDD operations – Examine records

```
results = book_dates.take(10)
```

- Take the first 10 elements of the RDD.
- The Map operation is executed.

```
print(results)
['1945-08-17', '1962', '1947', '', '', '1929-01-29',
'1968', '', '1996-10-01', '1995-10-01']
```

- Issues:
  1. Some values are missing,
  2. Some values also contain month + day.
  3. Some values need to be converted to number.

RDD operations – Extract year

```
book_years = book_dates.map(lambda value: 0 if not value
                            else int(value) if value.isdigit()
                            else int(value[0:4]))
```

- Return a new RDD by applying a function to each element of this RDD.
- 1-to-1 mapping: For each original record, a new record will be generated.
- The new RDD contains only the publication years of the books.
- Execution of the Map/Map operation is delayed.

RDD operations – Group by year

```
years = book_years.groupBy(lambda x: x)
```

- Group the values for each key in the RDD into a single sequence.
- Many-to-1 mapping: For all original record that have the same value, one record will be generated.
- Each new record is a tupple `(year, <Groupped values>)`
- The groupped values are encoded as `pyspark.resultiterable.ResultIterable` objects.
- Execution of the Map/Map/GroupBy operation is delayed.

RDD operations – Count items per year

```
year_stats = years.mapValues(len)
```

- Pass each value in the key-value pair RDD through a map function without changing the keys.
- The `len()` function is applied on each iterable object.
- 1-to-1 mapping: For each original record, one record will be generated.
- Each new record is a tupple `(year, number)`
- Execution of the Map/Map/GroupBy/MapValues operation is delayed.

RDD operations – Sort results and collect

```
year_stats.sortBy(lambda entry: entry[1], False).collect()
```

- Sorts the RDD by the given keyfunc
- Retrieve all entries of the RDD.

```
year_stats.values().variance()
```

- Return an RDD with the values of each tuple.
- Compute the variance of this RDD's elements.

Book title statistics: Words used in Book Titles

- Identify which words are most frequently used in book titles starting with the letter 'I'.
- Retrieve the 3rd field from each record.
- Select only book titles starting with the letter 'I'
- Extract the words from the book title.
- Count repetitions of each word.
- RDD Operations used:
  1. Map – extract the 3rd field.
  2. Filter – select records starting with the letter 'I'.
  3. FlatMap – extract each word from the title.
  4. Map – convert each word to a tupple `(word, 1)`.
  5. ReduceByKey – count appearances of each `(word, *)` tupple.

RDD operations – Book names

```
book_names = books.map(lambda a: a.split('\t')[2])
```

- Return a new RDD by applying a function to each element of this RDD.
- 1-to-1 mapping: For each original record, a new record will be generated.
- Each line is 1 record: fields are separated by tab characters.
- We split the line using the tab character, take the 3rd field: the name.
- The new RDD contains only the names of the books.

RDD operations – Sort Book Names

```
books.takeOrdered(5)
```

- Order the entries and take the first 5 elements of the RDD.

```
name_filter = books.filter(lambda a: a[0] == 'I')
```

- Return a new RDD containing only the elements that satisfy a predicate.
- No operation is executed.

```
name_filter.take(5)
```

- Execute the filter operation.
- Retrieve 5 records.

```
name_filter.collect()
```

- Collect all records.

RDD operations – Split Book Names

```
book_name_words = name_filter.flatMap(lambda a: a.split(' '))
```

- Examine each record and split into multiple records.
- Each word becomes a separate record.
- 1-to-many mapping: For each original record, multiple new records will be generated.
- No operation is executed.

```
book_name_words.take(5)
```

- Execute the filter + flatMap operation.
- Return 5 records.

RDD operations – Count Words (1)

```
book_name_words_tupples = book_name_words.map(lambda a: (a, 1))
```

- Convert each word into a tupple `(word, 1)`
- 1-to-1 mapping: For each original record, generate 1 new.
- No operation is executed.

```
book_name_words_tupples.take(5)
```

- Execute the Filter/FlatMap/Map operation.
- Return 5 records.

RDD operations – Count Words (2)

```
count_words = book_name_words_tupples.reduceByKey(lambda a, b: a + b)
```

- Merge the values for each key using reduce function.
- Many-to-1 mapping: Multiple records reduced to 1 record.
- No operation is executed.

```
count_words.take(5)
```

- Execute the Filter/FlatMap/Map/ReduceByKey operation.
- Return 5 records.

```
count_words.takeOrdered(5, key = lambda x: -x[1])
```

- Execute the Filter/FlatMap/Map/ReduceByKey operation.
- Order records based on counter, in reverse.
- Return 5 records.

RDD operations – Count Words (3)

```
top_5_words = books.map(lambda a: a.split('\t')[2])\
                   .filter(lambda a: a[0] == 'I')\
                   .flatMap(lambda a: a.split(' '))\
                   .map(lambda a: (a, 1))\
                   .reduceByKey(lambda a, b: a + b)\
                   .takeOrdered(5, key = lambda x: -x[1])
```

- Execute the Filter/FlatMap/Map/ReduceByKey operation.
- Order records based on counter, in reverse.
- Return 5 records.

Top Book Genre

- Each book is assigned to 0 or more Book Genres
  - The 6th field from each record.
  - Some records are not assigned to any Book Genre – empty string.
  - If 1+ genres, list is in json format including a dictionary `{ "genreID": "genre name", ... }`
- Count number of Books per Book Genre.
- Identify Top 10 Book Genres.

RDD operations – Top Book Genre

```
books.map(lambda entry: entry.split('\t')[5])\
    .filter(lambda a: len(a) > 0)\
    .flatMap(lambda entry: json.loads(entry).values())\
    .map(lambda a: (a, 1))\
    .reduceByKey(lambda a, b: a + b)\
    .takeOrdered(10, key = lambda x: -x[1])
```

Index Books by Book Genre

- Each book is assigned to 1 or more Book Genres
- Build an Index of Book Genres.
- Each Book Genre points to Books
- Use the Wikipedia article ID as Book ID.

RDD operations – Index Books by Book Genre (1)

```
import json

def convert_tupple(tupple):
    new_list = []
    genre = json.loads(tupple[1]).values()
    for item in genre:
        new_list.append( (item, tupple[0])  )

    return new_list

def extract_ids(tupple):
    new_list = []
    for entry in list(tupple[1]):
        new_list.append(int(entry[1]))

    return (tupple[0], sorted(new_list))
```

RDD operations – Index Books by Book Genre (2)

```
reverse_index = books.map(lambda entry: (entry.split('\t')[0],
                                         entry.split('\t')[5]))\
                     .filter(lambda tupple: len(tupple[1]) > 0)\
                     .flatMap(convert_tupple)\
                     .groupBy(lambda tupple: tupple[0])\
                     .mapValues(list)\
                     .map(extract_ids)\
                     .collect()
```