

# Documentation

50.043

Group3

He Yuhang 1003775

Li Jiayi 1003778

Leng Sicong 1003777

Wang Zilin 1003764

Zhang Jiazheng 1003772

[Run Instruction](#)

[Automation](#)

[Production System Automation](#)

[Analytic System Automation](#)

[System Design](#)

[Back-End](#)

[Web Server](#)

[Relational Database](#)

[Non-Relational Database](#)

[Front-End](#)

[Back-End - Front-End Communication](#)

[Automation Environment Variable Updates](#)

[Analytic System](#)

[Web Features](#)

[Home Page](#)

[Search Engine](#)

[Search Categories](#)

[Trending & Hot Books Lists](#)

[Add Book Box](#)

[User Login & Signup](#)

[Search Result Page](#)

[Search Filter](#)

[View More Button](#)

[Book Detail Page](#)

[Add Review Box](#)

[Review Filter](#)

[More Reviews Button](#)

[People Also Viewed Interface](#)

[Mobile Adaptation](#)

[Analytic Task](#)

[TF-IDF](#)

[Correlation](#)

[Extra Effort](#)

[Data Cleaning and Pre Processing](#)

[Clean Book Metadata Datasets](#)

[Cleaning Book Category and Book Title, author](#)

[Clean Original Datasets](#)

[Clean Prof Datasets](#)

[Combine Original and Prof Datasets](#)

[Including Review Information and Clean Related Field](#)

[Decompose Review Datasets](#)

[User System in Production System](#)

[Extra Features in Front End](#)

[Extra Effort in Automation Script](#)

[GitHub Full Continuous Integration and Delivery\(CD/CI\)](#)

[Appendix](#)

[MongoDB Schema](#)

[API List](#)

[Automation Scripts Running Time \(min:second\)](#)

[Analytic Task Running Time](#)

[Analytic Task Resource Utilization](#)

[3 data nodes](#)

[4 data nodes](#)

# Run Instruction

- Start a new instance using `ami-04613ff1fdcd2eab1` (or `ami-0f82752aa17ff8f5d` for us-east-1 region)
- SSH into the new instance
- Git clone the project (**You must clone it instead of downloading ZIP**)

```
$ git clone https://github.com/heyuhang0/ISTD50043-Project.git
$ cd ISTD50043-Project
```

- Setup the deploy environment. The script will prompt you to input your aws credentials and will then install the necessary dependencies. This is not the actual deploy script, and the script is only designed for `ami-04613ff1fdcd2eab1/ami-0f82752aa17ff8f5d` based on the dependencies that should be installed on these 2 amis.

```
$ ./scripts/setup_deployer.bash
```

- Now you are ready to launch the system (please use **python3.7** instead of python3)

```
# The script will deploy production (about 5-8 minutes) and cluster (about 3-5 minutes) IN PARALLEL. System cannot be launched if there already exists one.
```

```
# OPTION1
```

```
If you do not have a pem key
```

```
# you can launch with a new key (keyname must be unique, i.e., <keyname>.pem
```

```
# must not exists in ec2s keypairs)
```

```
# the pem file of the new keypair will be saved at the given path
```

```
$ python3.7 scripts/deploy.py --newkey <keyname>.pem launch --num-nodes 3
```

```
# OPTION 2
```

```
# If you launched once or already have the pem key file
```

```
# you can use the existing key
```

```
# IMPORTANT:(the filename must be same as the EC2 keypair name with .pem suffix)
```

```
$ python3.7 scripts/deploy.py --keyfile <path to pem file> launch --num-nodes 3
```

- To run analytics task (it will take about 4-5 minutes, including data ingestion)

```
# result of correlation and the url to download tf-idf result
```

```
# will be printed
```

```
# If you launch with `--newkey <keyname>.pem`
```

```
# then just use `--keyfile <keyname>.pem`
```

```
# i.e., the --keyfile <keyname>.pem must be the pem key used in launching
```

```
# production system and analytics cluster
```

```
$ python3.7 scripts/deploy.py --keyfile <keyname>.pem analyse
```

- To scale the cluster (it will take about 5-7 minutes)

```
# <n> is number of datanodes after scaling
```

```
# IMPORTANT: the --keyfile <keyname>.pem must be the pem key used in launching
```

```
# production system and analytics cluster
```

```
$ python3.7 scripts/deploy.py --keyfile <keyname>.pem scale --num-nodes <n>
```

- To terminate the system

```
# IMPORTANT: the --keyfile <keyname>.pem must be the pem key used in launching
```

```
# production system and analytics cluster
```

```
$ python3.7 scripts/deploy.py --keyfile <keyname>.pem terminate
```

**IMPORTANT about vCPU exceed limit error:** normal aws user has a vCPU limit of 32 cores at any given time, our production system takes 8 vcpu, and each of the nodes in the analytics cluster has 4 vcpu cores. So theoretically, we can have **a maximum of 4 data-nodes** for the analytics cluster. **If you want to have more than 4 datanodes, please increase the vCPU limit first.**

In addition, **please avoid launching another system immediately after destroying one, wait 30 seconds before doing so.** This may cause the vCPU exceed limit error, due to previous instances not fully terminated yet. In this case, wait 30 second, then destroy the cluster, then wait for another 30 seconds, launch the cluster again, the error should be resolved.

Below is an example of the error, cascating aborting will happen.

```
Exception in thread Thread-6:
Traceback (most recent call last):
  File "/usr/lib/python3.7/threading.py", line 926, in _bootstrap_inner
    self.run()
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 418, in run
    self._target()
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 593, in deploy_web_app
    .with_inbound_rule('tcp', 80))
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 265, in launch
    UserData=config.user_data
  File "/home/ubuntu/.local/lib/python3.7/site-packages/botocore/client.py", line 357, in _api_call
    return self._make_api_call(operation_name, kwargs)
  File "/home/ubuntu/.local/lib/python3.7/site-packages/botocore/client.py", line 676, in _make_api_call
    raise error_class(parsed_response, operation_name)
botocore.exceptions.ClientError: An error occurred (VcpuLimitExceeded) when calling the RunInstances operation: You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit.

Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.7/threading.py", line 926, in _bootstrap_inner
    self.run()
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 418, in run
    self._target()
  File "scripts/deploy.py", line 14, in <lambda>
    lambda: launch_production(sssh_config),
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 636, in launch
    run_in_parallel(deploy_mongodb, deploy_mysql, build_react, deploy_web_app)
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 450, in run_in_parallel
    raise Exception('Abort all tasks because of exception: ' + str(e))
Exception: Abort all tasks because of exception: An error occurred (VcpuLimitExceeded) when calling the RunInstances operation: You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit.

Traceback (most recent call last):
  File "scripts/deploy.py", line 54, in <module>
    main()
  File "scripts/deploy.py", line 44, in main
    launch_all(sssh_config, args.num_nodes)
  File "scripts/deploy.py", line 15, in launch_all
    lambda: launch_cluster(sssh_config, num_nodes)
  File "/home/ubuntu/ISTD50043-Project/scripts/deploy_production.py", line 450, in run_in_parallel
    raise Exception('Abort all tasks because of exception: ' + str(e))
Exception: Abort all tasks because of exception: Abort all tasks because of exception: An error occurred (VcpuLimitExceeded) when calling the RunInstances operation: You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit.
```

# Automation

Benchmarks of automation scripts running time are shown in [Automation Scripts Running Time \(min:second\)](#) section.

## Production System Automation

In production automation script, we run four tasks in parallel using python thread:

- **Deploy MySQL server**

The setting up of MySQL server is done by scripts/setup/setup\_mysql.bash. The setup script does the following things:

- Generate a random MySQL root password and apply it via debconf-set-selections
- Install MySQL server
- Config MySQL server including enabling UTF-8, setting bind-address and allowing remote access
- Install PHPMyAdmin in the background (only checked at the end)
- Create a new database
- Download data and import it to a temporary table
- Normalize data to two tables: review and user
- Save password to a credential file

After the setup script, the deploy script will export those saved credentials to be used by the web-app instance. To prevent losing data, the setting up process only happens if the instance does not exist. Otherwise, the deploy script will just export those credentials unless the override flag is set.

- **Deploy MongoDB server**

The setting up of MongoDB server is done by scripts/setup/setup\_mongo.bash. The setup script does the following things:

- Install MongoDB using apt
- Start service
- Create an admin user and project user with a random password
- Enable access control and config bin address
- Restart service
- Import data
- Create indexes
- Save credentials

Same as the MySQL server, the deploy script will then export the saved credentials. And the setting up also only happens if the instance is not found.

- **Build React**

Since react is used for frontend, a building process is required. It was originally done by the web-app server itself. However, as the project grows, the task becomes extremely CPU and RAM heavy. To minimize the web-app server downtime and better utilize EC2 resources, a separated instance is therefore used. It runs on a t2.xlarge instance and the instance is destroyed immediately after finishing the building. This task does the following thing:

- Download and install node.js
- Archive project on the deployer machine and send it to the builder instance

- Unzip the project and install required packages
  - Build
  - Download the generated files back to the deployer machine.
- **Deploy web app server (both frontend and backend)**  
 The deploying of the web app server is done in two stages. The first stage is setting up the environment which only runs once. It is done by scripts/setup/detup\_web.bash with following steps:
    - Install node.js and Nginx
    - Install pm2 (for node.js production management)
    - Config Nginx to reverse proxy /api to node.js localhost:8080
 The second stage happens every time which applies the latest changes to the server. It does the following things:
    - Upload project archive to the server and unzip
    - Install node.js dependencies
    - Upload the front-end build done by "Build React" task to /var/www/html folder
    - Generate .env file using credentials from MySQL and MongoDB server
    - Restart node.js process using pm2

During the deployment, there are a lot of data exchanges going on between instances and threads. To ensure correctness and efficiency, two important concepts are used:

- **Session and environment variables**  
 Session and environment variables are used to transfer data between the deployer machine and remote instances. Paramiko is used as the ssh client. By default, a non-interactive ssh connection is created. It has a simple API - takes in command and returns the stdout and stderr. However, it does not have a session between calls. For example, a `cd` command will have no effect on the next command. Although interactive session can be created, it seems to be buggy because we need to handle the stdin ourselves. Therefore, we implemented a session ourselves. After executing a command, we save all environment variables to a file. And before executing the next command, we restore environment variables from the file. This way, we implemented an interactive session like experience which allows us to directly translate command in terminal to automation code. And with environment variables saved, we can easily transfer the value between the instance and deployer instead of parsing the stdout with complex logic.  
 Another unexpected advantage of this approach is that since our session is stored by a file, we can easily restore it even after a connection error. The retry logic is implemented in the library level to make the automation more reliable.
- **Future Value**  
 With the session to transfer data between instances and deployer, we need another mechanism to sync data between deployer threads. Therefore, the FutureValue data structure is proposed. It is a thread-safe value that can be set once and get multiple times in different threads. If no value is set, the get call will be blocked until the value is set. It can be used to easily transfer data between threads, like the database passwords.

Those two concepts are also widely used in cluster automation.

The below sequence diagram shows the parallel execution and each server's responsibility in the production system.

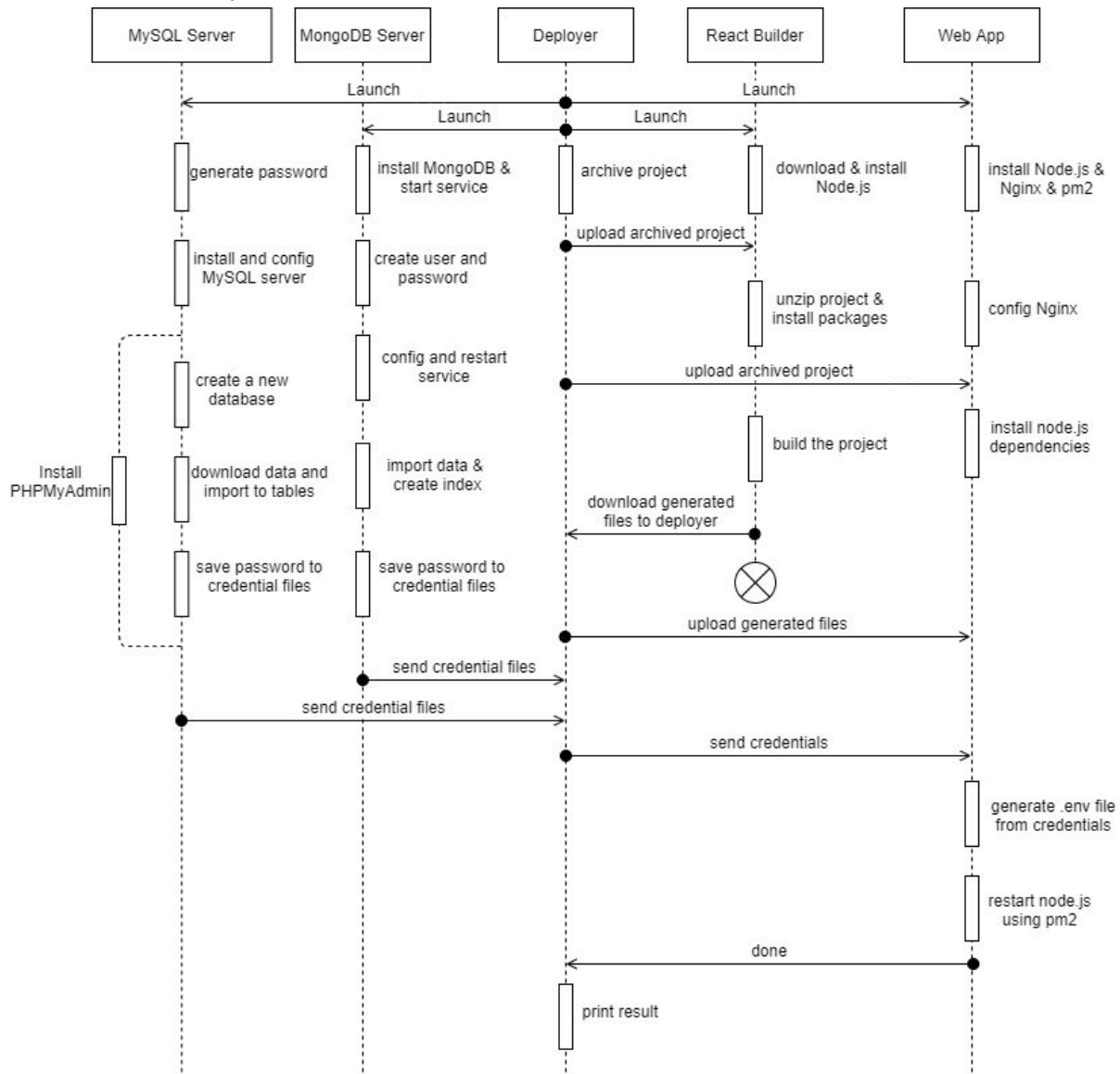


figure 1.1 Sequence Diagram of Production System

## Analytic System Automation

In analytic system automation, we also used the two concepts: session and future values mentioned in the last section, the sequence diagram below shows the parallel execution and each server's responsibility in the production system.

### - Launch Spark Cluster:

1. Input the number of data nodes needed for this cluster
2. Launch EC2 instances in AWS, then setup cluster in parallel:

For the name node:

1. Configure swappiness and install Java, Python pip, MongoDBTools, MySQLClient and apache2(for downloading result) in the background(by AWS cloud-init)

2. Export private IP address
3. Generate an ssh key pair and export the public key
4. Wait for data nodes' IP addresses to be reported and config the hosts file
5. Download and configure Hadoop
6. Wait for data nodes to setup ssh public key, then distribute configured Hadoop to data nodes via SCP
7. Install Hadoop
8. Download and configure Spark, then distribute configured Spark to data nodes
9. Install Spark
10. Wait for step 1 to finish (so we have java and pip)
11. Install the required python libraries - PySpark and NumPy
12. Configure Apache2 server to disable directory indexes since we want our result to be private
13. Wait for data nodes to finish installing Hadoop, start Hadoop cluster
14. Wait for data nodes to finish installing Spark, then start Spark cluster

For data nodes:

1. Configure swappiness and install Java and Python pip in the background (by AWS cloud-init)
2. Export private IP address
3. Wait for other nodes' IP addresses to be reported and config the hosts file
4. Wait for name node's public key and put it in authorized keys
5. Wait for name node to distribute Hadoop, then install it
6. Wait for name node to distribute Spark, then install it
7. Wait for step 1 to finish, so we have JDK installed, then report that it is ready to launch Hadoop
8. Install required python libraries - PySpark and NumPy
9. Report that it is ready to launch Spark

#### - **Do Analytic Task on Built Spark Cluster:**

We run analytic tasks in parallel to better utilize the cluster resources.

For the TF-IDF task:

1. Get credentials from MySQL instance
2. Connect to existing Spark cluster's name node
3. Export reviews data using MySQL client
4. Put data to HDFS
5. Submit TF-IDF analytical task
6. Export results from HDFS to name node
7. Move the TF-IDF result to apache2 folder and export the download URL

For the correlation task:

1. Get credentials from MongoDB instance
2. Connect to existing Spark cluster's name node
3. Export books data using mongoexport tool
4. Upload books data into HDFS system
5. Submit PySpark correlation task
6. Export results from HDFS to name node



## 7. Export the correlation result which is a number

After two tasks have finished, the automation script will print the correlation value and the download link for the TF-IDF result.

### - **Terminate Spark Cluster:**

1. Terminate all ec2 instances (name node and data nodes) filtered by tags

### - **Scale Spark Cluster:**

1. Terminate current Spark Cluster
2. Relaunch a new Spark Cluster with new input DataNode number

The below sequence diagram shows the parallel execution and each node's responsibility in the analytic system.

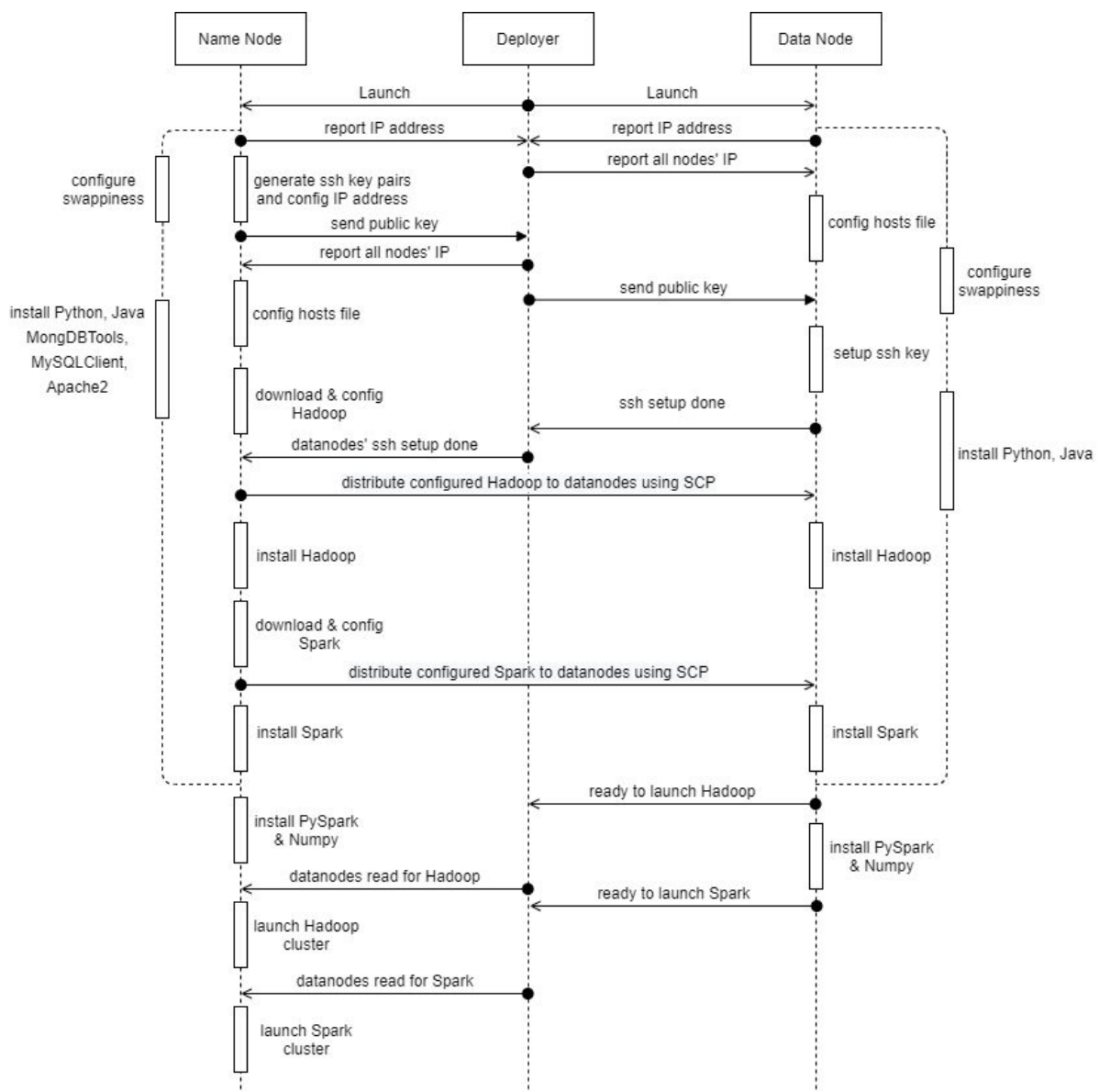


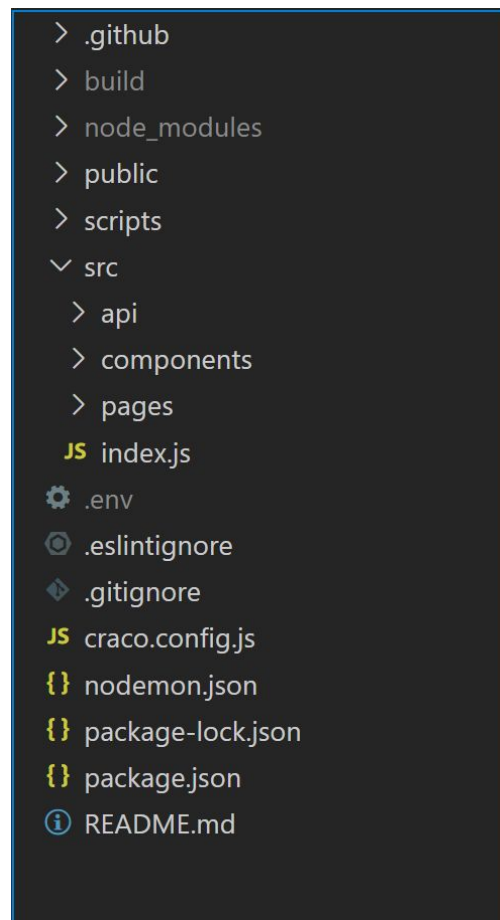
figure 1.2 Sequence Diagram of Analytic System

# System Design

The following bullet points summaries our production system structure and design.

- Separation of back-end and front-end for better resource management, scalability.
- Back-end, front-end communication: HTTP RESTful APIs
- Back-end:
  - Node.js + Express, for HTTP RESTful app
  - Environment variables for configuration management
  - Databases and Object Relational Mapping(ORM)
    - MongoDB and Mongoose ORM
    - MySQL and Sequelize ORM

Below is the structure of our WebApp including both front-end and back-end.



*figure 2.0.1 Web App Structure*

It contains following part:

- **.env**: environment variables to be used by back-end app
- **package-lock.json**, **package.json**, **node\_modules**: npm packages requirements and actual packages
- **craco.config.js**: react app configuration
- **nodemon.json**: a tool for automatically restarting app when file changes
- **src/**: actual app
  - **api/**: back-end app, details will be explained in the Back-End session
  - **components/**: common front-end components for all pages, e.g. search bar
  - **pages/**: front-end pages
  - **index.js**: front-end app entry point
- **public/**: public static resources, e.g., icons, website logos

## Back-End

We use Node.js and Express as our back-end HTTP server.

We use Object Relational Mapping(ORM) to interact with relational database management systems (RDBMS) and non-relational database management systems(RDBMS), for the ease of development and maintenance. We use the following databases and ORM

- MongoDB and Mongoose ORM
- MySQL and Sequelize ORM

The following diagram illustrates how different parts fit together.

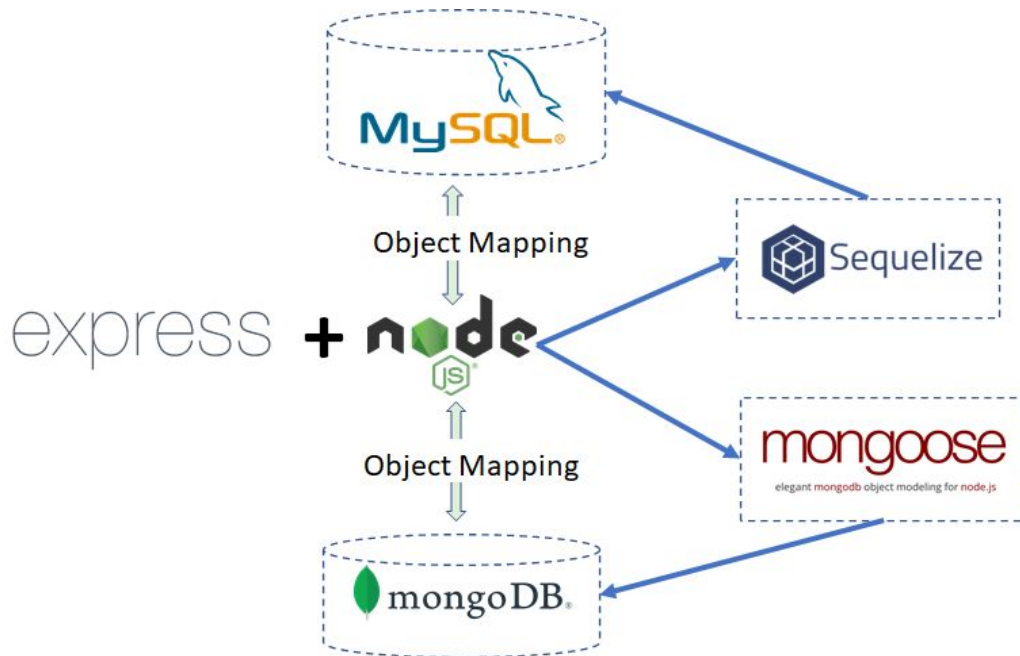


figure 2.1.1 Back-end Components Relations

## Web Server

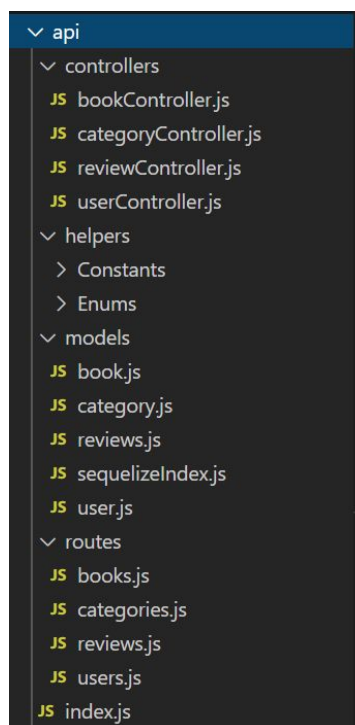


figure 2.1.2 Back-end app structure

The above image shows the structure of our back-end server. It contains the following parts:

- **index.js**, entry point to the back-end app. It has the following functionalities: load environment variables, connect to database, define error handlers and loggers (logging to mongoDB), create actual express app, define app routes.
- **routes/**, which defines all the routes and end points we use.
- **controllers/**, which defines the actual business logic. Separation of the logics and routes allows better structure and easier maintenance.
- **models/**, which defines the ORM models we use for MySQL and MongoDB.
- **helpers/**
  - **Constants/**, define application constant, e.g.email and ASIN regular expression.
  - **Enums/**, define the enums in the application to avoid redundancy, e.g., error message json, sort\_by statement which are fixed options.

The above structure allows us to make the back-end app modularized and clean, there are several other features that make our code clean and structural.

- **Separation of routes and actual logics.**
- **Unified configuration management using environment variables ( .env file)** instead of the hard-coded value for configuration.
- **Avoiding callback hells by using async await keywords.** We use async await pattern for all the asynchronous calls to avoid nested callbacks, thus avoiding callback hells.
- **Avoiding repeated code by defining constants and enums in helpers/.** All the constants and enums we use are stored in a separated file instead of hard-coded.

## Relational Database

We use MySQL as the database to store user's review information. Sequelize ORM is used to perform relational database operations.

From the original schema provided, we do schema decomposition to make the schemas 3rd normal forms. Below are the two schemas after decomposition.

**review table:**

(reviewId, reviewerId, asin, helpful, rating, summary, reviewText, createdAt, updatedAt)

primaryKey: reviewId

Index: asin, rating, helpful, reviewerId, createdAt, updatedAt

**user table:**

(reviewerId, reviewer\_name, email, password, createdAt, updatedAt)

primaryKey: reviewerId

foreignKey: reviewerId reference reviewerId in review table, using cascade policy.

As shown in the user table, we add email and password(hash) columns in the user table to support our user system. All the users existing in the tables have email and password(hash) null.

## Non-Relational Database

We use MongoDB to store book metadata information and HTTP request information. Mongoose ORM is used to perform non-relational database operations.

We have 3 collections in our non MongoDB

- **Categories**: storing all categories metadata in our database
- **Books**: storing all book metadata in our database.
- **Logs**: storing all user request logs to the system.

See Appendix session [MongoDB Schema](#) for the detailed schema.

## Front-End

We use React as our front-end framework.

We use Ant Design as our front-end design standard and use some of the components it provides, e.g., pre-defined button, list, search text input, form, etc.

## Back-End - Front-End Communication

Our back-end and Front-end app communicate using HTTP RESTFUL APIs.

GET for query resources.

POST for creating resources

PUT for updating resources.

See Appendix session [API List](#) for full list of APIs.

## Automation Environment Variable Updates

After automation is finished, the environment variables, such as MongoDB URL, MySQL URL, is updated to .env file, for the production system to use.

## Analytic System

Ingestion tools: MongoDBTools, MySQLClient

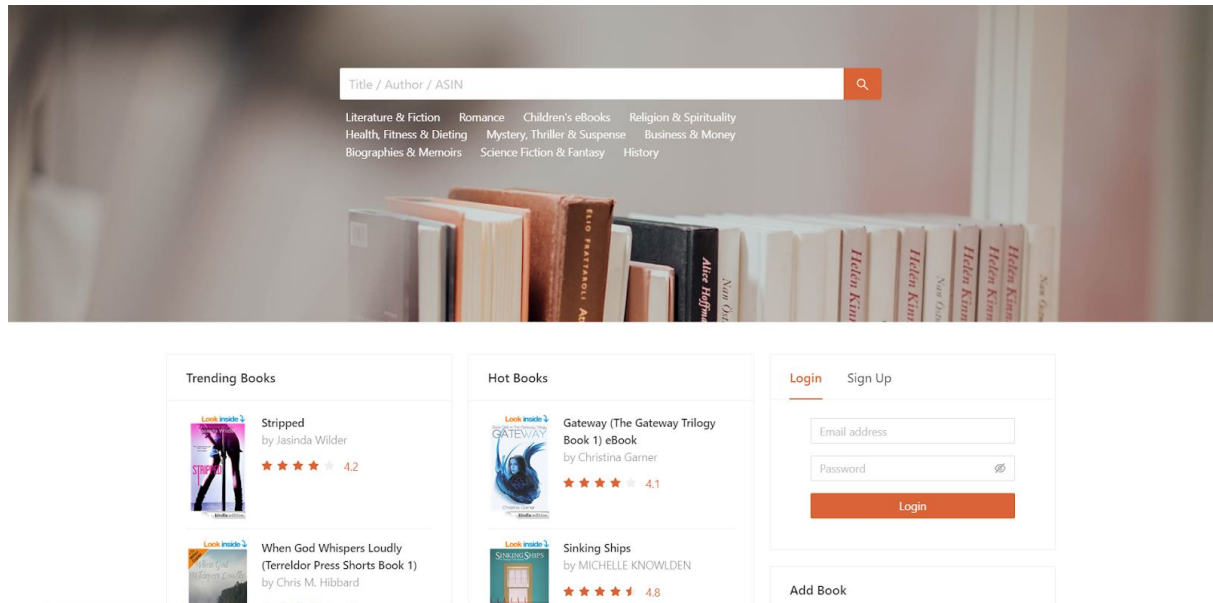
Analytics tools: HDFS + Spark (PySpark) + Yarn (resource manager)

- correlation: basic Spark operators
- TF-IDF: pyspark.ml.feature CountVectorizer, IDF, Tokenizer

Other tools: Numpy, Apache2

# Web Features

## Home Page



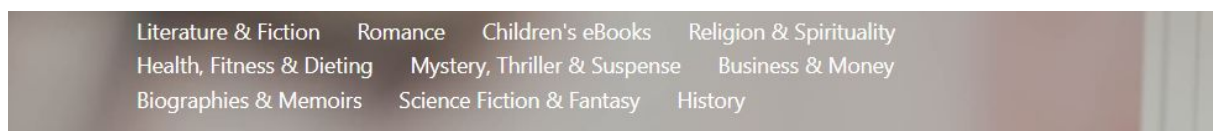
## Search Engine

We developed a search engine which takes in keywords or ASINs of books and returns the searching results. The placeholder inside the search box indicates the valid content of keywords.





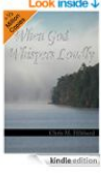


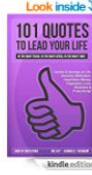


## Search Categories

There are the top 10 categories sorted by book numbers. They are clickable and return books with the specific category.



## Trending & Hot Books Lists

At our home page, there are two columns of books showing as “trending books” and “hot books”. The trending books and the hot books are retrieved and returned based on both the number of reviews and the ratings, and the ranks of the books respectively.

Trending Books		Hot Books	
	<p><b>Stripped</b> by Jasinda Wilder</p> <p>★★★★☆ 4.2</p>		<p><b>Gateway (The Gateway Trilogy Book 1) eBook</b> by Christina Garner</p> <p>★★★★☆ 4.1</p>
	<p><b>When God Whispers Loudly (Terreldor Press Shorts Book 1)</b> by Chris M. Hibbard</p> <p>★★★★☆ 4.3</p>		<p><b>Sinking Ships</b> by MICHELLE KNOWLDEN</p> <p>★★★★☆ 4.8</p>
	<p><b>Shooting Scars: The Artists Trilogy 2: (The Artists Trilogy 2)</b> by Karina Halle</p> <p>★★★★☆ 4.7</p>		<p><b>Yuhang Very Very NB</b> by Yuhang</p> <p>★★★★★ 5.0</p>
	<p><b>Everything I Left Behind (Men with Badges Book 1)</b> by JC Emery</p> <p>★★★★☆ 4.2</p>		<p><b>The Shadow Rises (Witch-Hunter Book 1)</b> by K.S. Marsden</p> <p>★★★★☆ 3.8</p>

## Add Book Box

We allow users to add books into the database. Users are only allowed to add books after their login.

### Add Book

Can't find the book?

[+ Add a new record](#)

Add a new book

\* Title

\* Author

\* Cover Image URL

\* Category

Select a category

\* Price

\$ 0

Description

Cancel

Add

## User Login & Signup

Users are able to create or login to their accounts. The interface applies form validation that checks the syntax format of input.

Login

Sign Up

Email address

Password

Login

Login

Sign Up

Name

Email address



Password

Confirm password


Sign Up



# Search Result Page




Most Commented




Look inside

**Bah, Humbug! (Christmas Street Romance #1)**  
by Heather Horrocks  
★★★★☆ 4.2  
394 reviews



Look inside

**A Bride for Tom (Nebraska Historical Romances Book 2)**  
by Ruth Ann Nordin  
★★★★☆ 4.0  
205 reviews




Look inside

**Rise of the Fallen: A Dark Paranormal Romance (All the King's Men Book 1)**  
by Donya Lynne  
★★★★☆ 4.3  
199 reviews

## Search Filter


The filter includes sorting by the number of comments and sorting by ratings.

Most Commented




Look inside

**Play**  
by Kylie Scott  
★★★★★ 4.7  
481 reviews



Look inside

**Bah, Humbug! (Christmas Street Romance #1)**  
by Heather Horrocks  
★★★★☆ 4.2  
394 reviews




Look inside

**Dane: Vol 1 (MacKenzie's of Montana)**  
by Liliana Hart  
★★★★☆ 3.8  
394 reviews

Top Rated

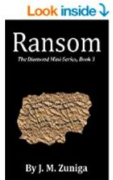
Most Commented

Top Rated




Look inside

**Christmas Cantata (The Liturgical Mysteries Book 10)**  
by Mark Schweizer  
★★★★★ 5.0  
8 reviews



Look inside

**Ransom (The Diamond Mini-Series, Book #3)**  
by J. M. Zuniga  
★★★★★ 5.0  
7 reviews



Look inside

**Not So Lucky**  
by J. M. Zuniga  
★★★★★ 5.0  
6 reviews

17

## View More Button

The button at the bottom of the search page allows users to see more searching results.





by


★★★★★ 4.7

40 reviews

View more

## Book Detail Page

 Title / Author / ASIN 




**The Christmas Cantata (The Liturgical Mysteries Book 10)**  
by Mark Schweizer  
★★★★★ 5.0  
8 reviews


Look inside

kindle edition


People also viewed




An Improper Holiday  
by K.A. Mitchell  
★★★★★ 4.2




Right Side Talking  
by Bonnie Rozanski  
★★★★★ 3.6



Pope Bob  
by Bill Dodds  
★★★★★ 4.2



The Tourist  
by Clare London  
★★★★★ 4.5



Hull's Landing (A Short Novel)  
by James Melzer

Your review

★★★★★

Headline or summary of your review (required)

Write your review here (required)

## Add Review Box

Users are only allowed to add reviews after their login. Users can add reviews with a summary and choose to give a rating. Users can also edit their own reviews and give likes to other reviews.

Your review

★ ★ ★ ★ ★

Headline or summary of your review (required)

Write your review here (required)

Submit

## Review Filter

The filter includes sorting by helpfulness(number of likes) and sorting by time.

Community reviews

Most Helpful

J

Jackie-the-Greyhound 2011-11-14

★★★★★ Another fun visit to St. Germaine, where one can read lines like:  
"I could prob'ly sell another dozen if I could get any more of them praying Santas, but I can't. I order them from Japan. That's why that baby Jesus is wearing a kimono."I must admit I am a big fan of the Liturgical Mysteries series--generally a nice taste of small town living, a funny cast of characters, throw in some religion (Episcopal variety), music and a dose of murder to figure out and these are a great read. This one is a little different, there is indeed a mystery but no murder (and no revolving door for the clergy) this time around, but reads more like an extended version of a slice of life involving our favorite characters.Though the books don't have to be ... [more](#)

7

E

Ethan Winning "Ethan Winning" 2012-03-01

★★★★★ No bodies, but a wonderful yarn with great characters  
This was my first Schweizer novel, but it won't be my last. However, a word of caution: almost all reviews state that in every Mark Schweizer novel, a body turns up. I kept waiting, but nobody. Plenty of busybodies, all fleshed out very well by the author.What we have is a mystery, in this case about a cantata, but it could have been a piece of history, a secret Declaration of In Dependents, or secrets to fly fishing. It doesn't matter. It remains a mystery until near the end.For lack of a better word, it's a "comfortable" book. Great bedtime reading that leaves one kind of warm and toasty (well, it does take place at Christmastime) and extremely satisfied with the c... [more](#)

2

J

J. L. Hewes 2011-11-28

★★★★★ Christmas Cantata Best Yet  
The Christmas Cantata doesn't have the usual goofy "Chandler-style" mystery of the previous Schweizer Liturgical series. Instead, Hayden is given an old manuscript cantata and has to solve the mystery of who wrote it. The italic parts are flashbacks to the composer's story. I think it's his best writing so far, though I own and love all of the series.

## Community reviews

Most Recent

Most Helpful

Most Recent

13-11-23

### ★ wonderful love story

I purchased this book as soon as it was published, and this year when it showed up in an book format, well, I had to purchase it again. This may be the best writing from Mr. Schweitzer, and his other works are great! The tenderness shown in this book over- shadows all his other books. This would make a wonderful Christmas movie. Make it so, Mr. Schweitzer, please.

👍 0

A

Avid Mystery Reader 2013-07-15

### ★★★★★ A Different Type of Christmas Story

This is a wonderful story, a mystery, a love story, a story about a life. Beautifully told with respect and humor. Highly recommended.

👍 0

W

Wendy Xyz "wendyf1952fl" 2013-07-03

### ★★★★★ Excellent!

The Best in a wonderful series. I'm only sorry there's only one more book in the series. I hope there will be many more. Read them all.

👍 0

## More Reviews Button

The button at the bottom of the reviews box allows users to view more reviews.

K

KNIVESINORBIT "the orbit" 2014-01-25

### ★★★★★ I need more!!!

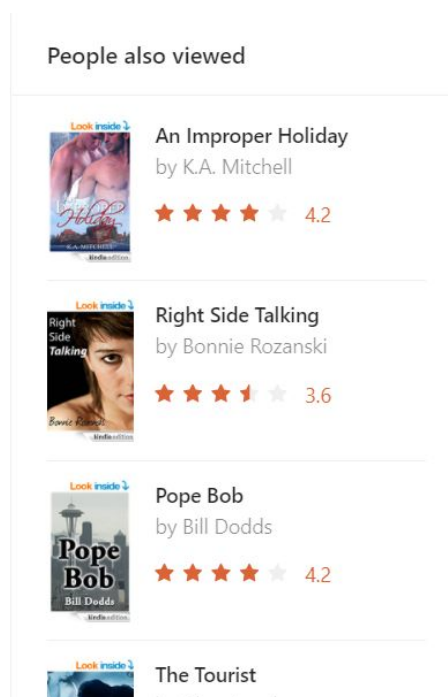
One of my favorite new series to date. The take of the Amish boy Jed and where he actually may have gone and our arrived...Earth or New Pennsylvania...I need to have this be completed. Keep them coming Bunker!!!

👍 1

▼ More reviews

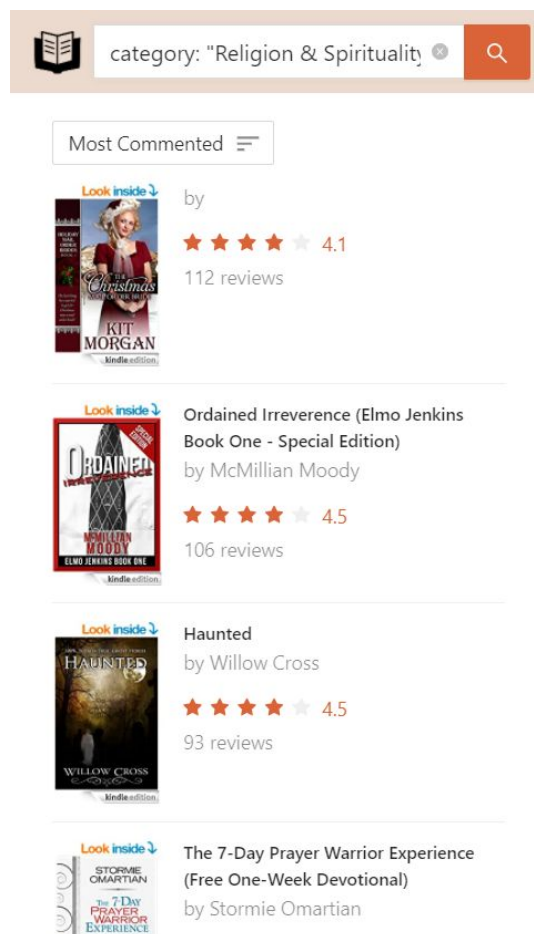
## People Also Viewed Interface

The interface shows a list of books related to the current book.



## Mobile Adaptation

The web application is designed to be adaptive to all devices.



# Analytic Task

Benchmarks of analytic tasks running time including data ingestion process is shown in the [Analytic Task Running Time](#) section.

## TF-IDF

$tf_{ij} idf_i = tf_{ij} \log(\frac{N}{df_i})$  where  $tf_{ij}$  is the frequency of word  $i$  in document  $j$ , and  $df_i$  is the total number of documents containing the word  $i$ . It is the representation of the word importance in the document.

We used Spark MLlib to get term-frequency (TF) and inverse-document-frequency (IDF) separately.

- (1) We dropped the reviews with no texts and transformed each review to a list of words using the Tokenizer, only English words are preserved, punctuations, numbers are removed.
- (2) TF vectors are then generated using CountVectorizer. These are the raw features for each review.
- (3) Spark IDF is an estimator which fits on the dataset and produces an IDFModel. The IDFModel takes the raw features calculated by CountVectorizer and scales them accordingly. An illustration is as below.
- (4) After we get the TF-IDF of each word in every document, we convert the indices to the original word and save the feature vector as a string. Finally, we select the reviewId as key and output the TF-IDF of each review.

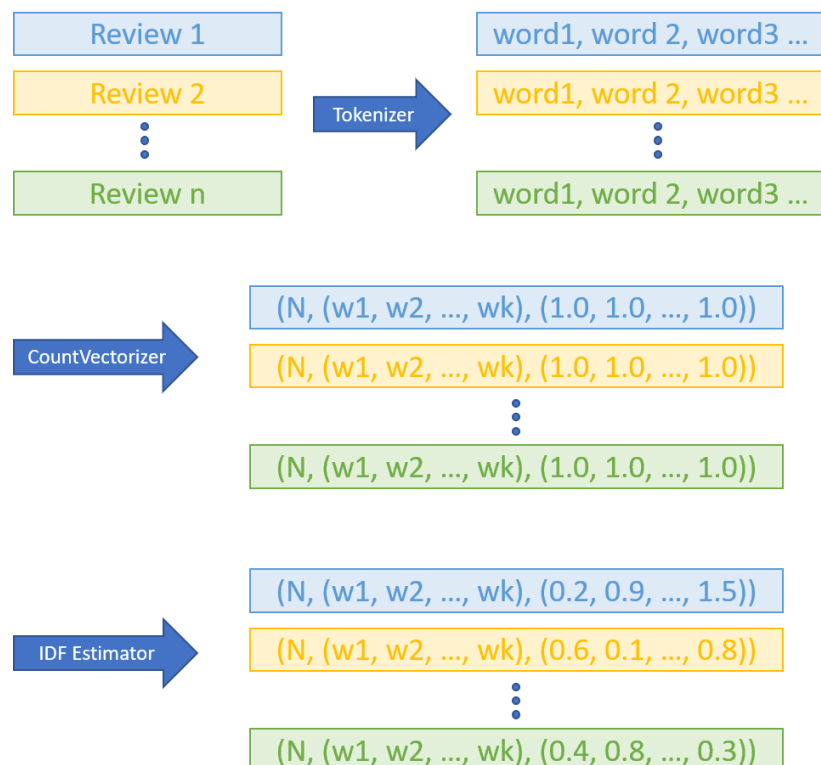


figure 5.1.1 tf-idf task workflow

- (5) The output is written to a csv file. Below is the screenshot of part of the output file. As we expected, common words that appear frequently in most documents receive small TF-IDF values, while rare words have high TF-IDF values.



```

0
the:0.7086997490879355, and:0.4105721155095747, to:0.24839964414029117, a:0.2456023265406929, i:0.5237423742537579,
:1.2104049176194362, this:0.2753618137676199, it:0.46649267884682033, in:1.1371100709057782, was:1.3467815038862805,
book:0.5120272838306481, for:0.6574002622830457, but:0.8226695828947552, read:0.7907605199824121,
so:1.203367429700115, me:1.343166497952599, out:1.437147176350045, good:1.3861936095025638,
would:1.5760496052339623, books:1.676818958777958, or:1.776349060489871, them:1.8135874219256118,
1 reading:1.686462055712262, enjoyed:2.0084029944865547, dont:2.1978557780128103, think:2.250082940382628,
plot:2.7090633698356372, enjoy:2.730502470973159, without:2.9955745356174357, someone:6.39653222616938,
scene:3.8624027549312476, body:4.253469094992268, leaving:4.508308225427349, police:5.019749897264571,
unusual:5.207664016024797, killing:5.294421914013805, movies:5.732342351015947, knock:6.446763551867346,
hitting:6.60123520746369, jaw:6.783107427022495, wash:7.708876902851194, vintage:7.986780785321339,
selfdefense:8.821188035877466, notifying:10.802189504744048, todaystill:12.699309489629929

the:0.5669597992703485, and:0.4105721155095747, to:0.24839964414029117, a:1.2280116327034645, i:0.26187118712687896,
of:2.0830536779525413, :1.2104049176194362, is:1.0598638985750903, this:0.2753618137676199, it:0.46649267884682033,
in:0.5685550354528891, was:2.693563007772561, book:0.5120272838306481, one:1.1771666356100925,
my:1.3164623384102365, an:1.3872036200957922, like:2.792244000476742, at:1.4784847338128881,
good:1.3861936095025638, who:1.6424584505944078, up:1.5923642654259136, its:1.7750871093461882,
well:1.6725007758049137, author:3.3703096458092205, been:2.0563399853283806, little:2.0862766107242203,
also:2.1128092710144544, never:2.3928488203072864, say:2.4445314634319093, made:2.499527061543798,
2 still:2.511896913519733, interesting:2.6010504258353686, why:2.8939494150917437, look:2.836609209994637,
times:2.9391920477528157, hes:3.281331836211006, quite:3.145686789103985, worth:3.0737517181029044,
old:3.2167052979303468, use:3.481397852157429, words:3.581486534187656, id:3.7750968431732908,
language:4.542321274916993, heard:4.667707450801141, introduction:4.922355086307488, born:5.312632135894811,
calling:5.805653135027295, explaining:5.777980090620675, era:5.86843525498375, forgotten:5.868075348555536,
gun:6.234721185939969, dated:6.744336051104363, himthe:6.923723282844745, wolfe:7.482563776775705,
fires:7.2641329403646955, dictionary:7.377926850150897, nero:8.494616870238964, reissue:9.827629864745917,
heater:10.396724396635884, looksee:10.802189504744048

the:0.42521984945276137, and:0.615858173264362, to:0.49679928828058234, a:0.7368069796220786, i:0.5237423742537579,
of:0.8332214711810165, :1.8156073764291545, this:0.2753618137676199, it:0.46649267884682033,
that:0.7075280268368089, was:1.3467815038862805, for:0.6574002622830457, story:1.7358460416806802,
read:2.372281559947236, have:2.117635777081933, are:1.1975696848847681, they:3.084804083130224,
like:1.396122000238371, out:1.437147176350045, had:1.6481655482467696, if:1.5773251187517674,
3 get:1.6645488661861436, can:3.377986987308703, interesting:2.6010504258353686, doesnt:2.7502910468458124,
fun:2.7042612019225, enough:2.7039570795413104, without:2.9955745356174357, old:3.2167052979303468,
free:3.193318875552789, glad:3.3825092065526197, style:3.58386604952343, writer:3.787015828994641,
check:4.380729851784625, language:4.542321274916993, decide:4.745405491515424, fairly:4.907373532691872,
command:6.7145339306727445, convey:6.809508596298448, crude:6.888168496653229, coarse:8.355504067776247,
terminology:12.411627417178149, relaxation:12.411627417178149, englishthat:12.699309489629929

the:0.1417399498175871, i:0.26187118712687896, of:0.41661073559050826, :0.6052024588097181, this:0.2753618137676199,
read:0.7907605199824121, on:1.125746853902397, so:1.203367429700115, one:1.1771666356100925,
really:1.472507312647489, them:1.8135874219256118, am:2.1960426571175846, never:2.3928488203072864,
now:2.407604722395191, any:2.528993293837187, until:2.8593121190420856, id:3.7750968431732908,
4 hooked:4.058071427351738, mysteries:5.13974999362223, amy:6.142531133471888, brewster:10.396724396635884

the:0.1417399498175871, of:0.41661073559050826, :1.8156073764291545, this:0.2753618137676199,
you:1.9775870453441404, me:1.343166497952599, will:1.3924328601287752, like:1.396122000238371,
at:1.4784847338128881, had:1.6481655482467696, if:1.5773251187517674, author:1.6851548229046103,
5 way:1.9122403646293515, through:2.363314675207111, enjoy:2.730502470973159, mystery:3.362013464559331,
least:3.71520076556719, guessing:4.7949746475448345, period:4.983888576645255, pieces:5.169724028838891,
clothing:6.780415635356783, lingo:7.977838847945677

```

figure 5.1.2 first five rows of the tf-idf output file

## Correlation

Pearson correlation is defined as: 
$$r = \frac{\sum_i^n x_i y_i - \sum_i^n x_i \sum_i^n y_i}{\sqrt{n \sum_i^n x_i^2 - (\sum_i^n x_i)^2} \sqrt{n \sum_i^n y_i^2 - (\sum_i^n y_i)^2}}.$$

To get the correlation between the average review length of a book and its price, we first retrieve the data from HDFS. We need to fetch two data reviews.csv with all the pre-processed reviews and books.json with the book metadata. We denote  $x_i$  as the average review length of the reviews for book  $i$ , and the book's price as  $y_i$ .

- (1) For the reviews, we first group the reviews by ASIN and get the average review length. For the books, we first drop the books with negative price values.
- (2) Then we inner join these two dataframes by ASIN to get a combined dataframe with ASIN as key, and average review length and price as attributes.

ASIN	average_review_length	price
asin1	float	float
asin2	float	float
...		
asin3	float	float

figure 5.2.1 inner join result structure

- (3) We applied the MapReduce procedure to calculate correlation effectively. In the mapping stage, we used flatMap to map the combined data with  $(x_i, y_i)$  into five features  $(x_i, x_i^2, y_i, y_i^2, x_i y_i)$ .

```
[
  ("x", float), ("x_square", float), ("y", float), ("y_square", float), ("xy", float),
  ("x", float), ("x_square", float), ("y", float), ("y_square", float), ("xy", float),
  ... ...,
  ("x", float), ("x_square", float), ("y", float), ("y_square", float), ("xy", float)
]
```

*figure 5.2.2 flatMap results structure*

- (4) After that we apply reduceByKey with function  $\lambda x, y : x + y$  to get the summation of each terms in the flatdata, i.e.  $(\sum_i^n x_i, \sum_i^n x_i^2, \sum_i^n y_i, \sum_i^n y_i^2, \sum_i^n x_i y_i)$ .

```
[("x", float), ("x_square", float), ("y", float), ("y_square", float), ("xy", float)]
```

*figure 5.2.3 reduceByKey results structure*

- (5) Finally, we calculate the correlation using the terms above. The output value of correlation between the book price and average review length is **0.02803797087886728**.



# Extra Effort

## Data Cleaning and Pre Processing

We examined the book metadata provided by prof, and found the following problems:

- **Only 44 book records have titles** out of 434,702 books, and they are not actually books, instead, they are kindle stand, kindle cover etc.
- **Each book record has multiple nested categories**, but we only want 1 main category.
- **Review number, average rating is not in the information**, this will make the sorting by review number or rating difficult.

So we decided to clean the data ourselves. In addition, we also want to include the rating information in the book metadata table, for easier usage in the production system, such as sorting books by rating numbers, average rating score etc.

## Clean Book Metadata Datasets

We got two datasets. The kindle book metadata datasets from Amazon(original datasets in the following section), and the kindle kindle book metadata datasets provided by Prof. (Prof datasets in the following section). We use spark to perform the cleaning.

### Cleaning Book Category and Book Title, author

#### Clean Original Datasets

Below is an example of the original datasets, and we do the following cleaning

```
{
  "category": [
    "Kindle Store",
    "Kindle eBooks",
    "Science Fiction & Fantasy"
  ],
  "tech1": "",
  "description": [
    ],
    "fit": "",
    "title": "",
    "also_buy": [
      "B00JSRQV54",
      "B010MH9UYW"
    ],
    "image": [
      ],
    "tech2": "",
    "brand": "Visit Amazon's Dayton Ward Page",
    "feature": [
      ],
    "rank": "1,595,741 Paid in Kindle Store (",
    "also_view": [
      "B003AY9LIU",
      "B000JMKU04",
      "B00SLPZ498",
      "B000FC00EU"
    ],
    "details": {
      },
    "main_cat": "Buy a Kindle",
    "similar_item": "",
    "date": "",
    "price": "",
    "asin": "B000FBJFIC"
  }
```

*figure 6.1.1 Uncleaned Data Structure in Original Dataset*

- **Clean all text fields**, clean up unicode characters and html entities.

- **Clean brand(author) field.** Some of the records have the format “Visit Amazon’s xxx Page”, where xxx is the author, so we only take the xxx as author if it has this format.
- **Clean the rank field,** by taking the first word in the rank field and convert it to integer, -1 if the field does not exist or not in the format shown in the example above.
- **Clean the category field,** by only taking the first category after “Kindle eBooks”
- Combine “also\_buy” and “also\_review” fields to form “recommendation” fields.

The cleaned original dataset records have the following structure:

```
{
  "ASIN": "B00128EFK0",
  "category": "Religion & Spirituality",
  "title": "Approaching the Holy-A Season in God's House",
  "author": "Dennis G. Crump",
  "image": [
  ],
  "description": [
  ],
  "rank": "4646191",
  "recommendation": [
  ],
  "price": ""
}
```

*figure 6.1.2 Cleaned Data Structure in Original Dataset*

## Clean Prof Datasets

Below is an example of the Prof datasets, and we do the following cleaning:

```
{
  "asin": "B000FAST4W",
  "price": 12.46,
  "imageUrl": "http://ecx.images-amazon.com/images/I/51eWyBr8A3L...B02",
  "related": {
    "also_bought": [
      "B004RKH0GS",
      "B004ULMJH2",
      "B00B137JDY",
      "B00FN4NNGY",
      "B007LUGLYO",
      "B007JCNGK",
      "B0044KLPOW"
    ],
    "buy_after_viewing": [
      "B004RKH0GS",
      "B00B137JDY",
      "B00710A8V6",
      "B004478AMA"
    ]
  },
  "categories": [
    [
      "Books",
      "Business & Money",
      "Accounting"
    ],
    [
      "Books",
      "Business & Money",
      "Personal Finance",
      "Budgeting & Money Management"
    ],
    [
      "Kindle Store",
      "Kindle eBooks",
      "Business & Money",
      "Accounting"
    ],
    [
      "Kindle Store",
      "Kindle eBooks",
      "Business & Money",
      "Personal Finance",
      "Budgeting & Money Management"
    ],
    [
      "Kindle Store",
      "Kindle eBooks",
      "Professional & Technical",
      "Accounting & Finance",
      "Accounting"
    ]
  ]
}
```

*figure 6.1.3 Uncleaned Data Structure in Prof Dataset*

- **Clean the category field,** by taking the first category after “Kindle eBooks”, if not exists, the first category after “Books”.

- Combine all fields in related fields, to form a final “related” field.

The cleaned Prof dataset records have the following structure:

```
{
  "asin": "1603420304",
  "price": 7.69,
  "imUrl": "http://ecx.images-amazon.com/images/I/51IEqPrF%2B9L._B02,204,",
  "title_prof": "",
  "author": "",
  "cate": "Cookbooks, Food & Wine",
  "description": "In less time and for less money than it takes to order",
  "related": [ ]
}
```

figure 6.1.3 Cleaned Data Structure in Prof Dataset

### Combine Original and Prof Datasets

Finally, we want to keep the books the same as Prof datasets, so we performed a left join on the Prof datasets, and the original datasets. Here is the final cleaned book structure

```
{
  "asin": "B0000I19AI",
  "price": 8.07,
  "imUrl": "http://ecx.images-amazon.com/images/I/41tIDgGgQjL._B02,",
  "description": "A scholar in ecology and social theory, Redclift",
  "related": "[B0068T28A2]",
  "rank": 2345234,
  "title": "Chewing Gum: The Fortunes of Taste",
  "author": "Michael Redclift",
  "category": "Cookbooks, Food & Wine"
}
```

figure 6.1.4 Combined Data Structure from Prof and Original Datasets

### Stats:

	Prof Datasets	Original Datasets	Final Cleaned Datasets
Total records	434702	491670	<b>434702</b>
With titles	44	488910	<b>190899</b>

table 6.1.1 Stats of the Original, Prof, and Final Datasets

We can see a huge improvement in the number of books with titles, compared to the Prof datasets.

### Including Review Information and Clean Related Field

After that, we further perform the following cleaning and processing

- Remove non-existing asin in the “related” field
- Include rating information, using the review table.
  - Aggregate the review information group by asin, get each book’s rating number, rating total, rating average
  - Do a left join on the cleaned book metadata records, with the aggregated review information.

This is the final full cleaned version of the book metadata information.

```

{
  "asin": "B00DJB6KE2",
  "price": 17.95,
  "imageUrl": "http://ecx.images-amazon.com/images/I/41xgGSXLBWL._B02,204,2",
  "description": "A fine love story that manages to be both charming and",
  "related": [
    "B002VFPS4U"
  ],
  "rank": 263119,
  "title": "Stripped",
  "author": "Jasinda Wilder",
  "category": "Literature & Fiction",
  "review_number": 334,
  "rating_average": 4.2036,
  "rating_total": 1404
}

```

*figure 6.1.5 Final Cleaned Data Structure*

## Decompose Review Datasets

We decompose the original review datasets to review and user table as described previously. It performs the following decomposition:

(reviewerId, reviewerName, asin, summary, reviewText, rating, helpful, unixTimeCreate)

to

Review(reviewId, reviewerId, asin, helpful, rating, summary, reviewText, createdAt, updatedAt) & User(reviewerId, reviewer\_name, email, password, createdAt, updatedAt)

Here are the steps.

- Imported original data to MySQL in a temporary table("temp")
- Created "user" table with schema (reviewerId, name, email, password), where reviewerId is primary key and auto increases.
- Group records in temp by temp.reviewerId and find the username with maximum length, write the users to the user table, temp.reviewerId is stored in user.email field temporarily.
- Create the "review" table, write reviews record in temp table to the final review table, the review.reviewerId, is found by finding user.reviewerId using temp.reviewerId = user.email
- Clean user.email to NULL.

## User System in Production System

In our production system, users are able to sign up a new account or log in with registered email. It is achieved by following:

- Every new signed up user information (reviewerId, reviewer name, email, password hash), would be added in the "user" table in MySQL server.
- Back-end will generate a bearer authentication token upon logging in or register, and pass to front-end, and front-end will use this token as the authentication bearer header in the subsequent API calls, and back-end will decode the token, and know, whether the user has logged in, and who is the user.

## Extra Features in Front End

- **Responsive Front End design for all devices**, eg. PCs and smartphones. We use Ant Design's grid feature, and Media Query of CSS to achieve this.
- **Upvote reviews**: users are able to upvote the reviews which they thought may be useful or helpful. We keep a "helpful" field in the "review" table, the number will increase by 1, when a user upvotes a particular review.
- **Load-More option**: users can view more books or reviews in the same page by clicking the Load-More button at the bottom. In this way, users can view more content without refreshing the current page.
- **Recommendation book system**: while viewing the detail of one book, there is a section on the page aiming to recommend books that other people may also read or have similar features with the current book.

## Extra Effort in Automation Script

- **Parallel execution** while
  - Deploying production system and analytic system.
  - Deploying the 3 servers in our production system.
- Bypassing some AWS instabilities
  - AWS's setup scripts still running after instances begin to run, we wait until it finishes, otherwise, apt-get will be affected.
  - Sometimes the security group does not exist immediately after it is created, we wait for 5s after creating the security group to bypass this AWS issue.
- No external developed libraries used: except for some basic libraries for SSH connection, we did not use any other existing libraries either to build the web server or the Spark Cluster.

## GitHub Full Continuous Integration and Delivery(CD/CI)

We enabled GitHub's CI/CD, to do testing and deployment to our testing servers for every new commit on the `main` branch.

## Appendix

### MongoDB Schema

- **Categories**: storing all book categories in our database.
  - {
  - "`_id`": MongoDB id,
  - "`category`": category name,
  - "`book_number`": book number of this category.
  - }
- **Books**: storing all book metadata in our database.
  - {
  - "`_id`": MongoDB id,
  - "`asin`": book asin,

- "title": book title, default "",
- "author": book author default "",
- "description": book description, default "",
- "rank": book rank, default -1,
- "related": related books, default [],
- "rating\_average": book average rating, default 0,
- "rating\_total": total rating scores,
- "rating\_number": number of ratings
- }
- Index:
  - {"asin": "hashed"}
  - {"title": 1}
  - {"author": 1}
  - {"category": 1}
  - {"rank": 1}
  - {"rating\_average": -1}
  - {"review\_number": -1}
  - {"review\_number": -1, "rating\_average": -1}
  - {"title": "text", "author": "text"}
- **Logs:** storing all user request logs to the system.
  - {
  - "\_id": MongoDB id,
  - "logs": HTTP log in the format of (:method :url :status :res[content-length] - :response-time ms)
  - }

## API List

- **/books:**
  - GET /search
  - POST /
  - GET /trending
  - GET /hot
  - GET /:asin
- **/books/:asin/reviews/**
  - GET /
  - POST /
  - PUT /:reviewid
  - POST /:reviewid/upvote
- **/categories**
  - GET /
  - GET /suggested
  - GET /:category
- **/user**
  - POST /login
  - POST /register
  - GET /me

## Automation Scripts Running Time (min:second)

	Deploy Production System	Deploy Analytics System	Scale Analytics System
Try1	5:52	3:13	5:05
Try2	6:38	3:48	5:10
Try3	6:09	3:54	6:19
Try4	6:11	3:28	4:27
Average	6:12	3:36	5:15

*table 6.1.1 System Deployment Time*

## Analytic Task Running Time

	3 data nodes	4 data nodes	2 data nodes	1 data node
Try1	3:58	3:46	3:32	6:02
Try2	3:33	3:33		
Try3	3:48			
Average	3:46	3:40	3:32	6:02

*table 6.2.1 Analytics Running Time, Data Ingestion Included*

## Analytic Task Resource Utilization

### 3 data nodes

Green line is the name node.

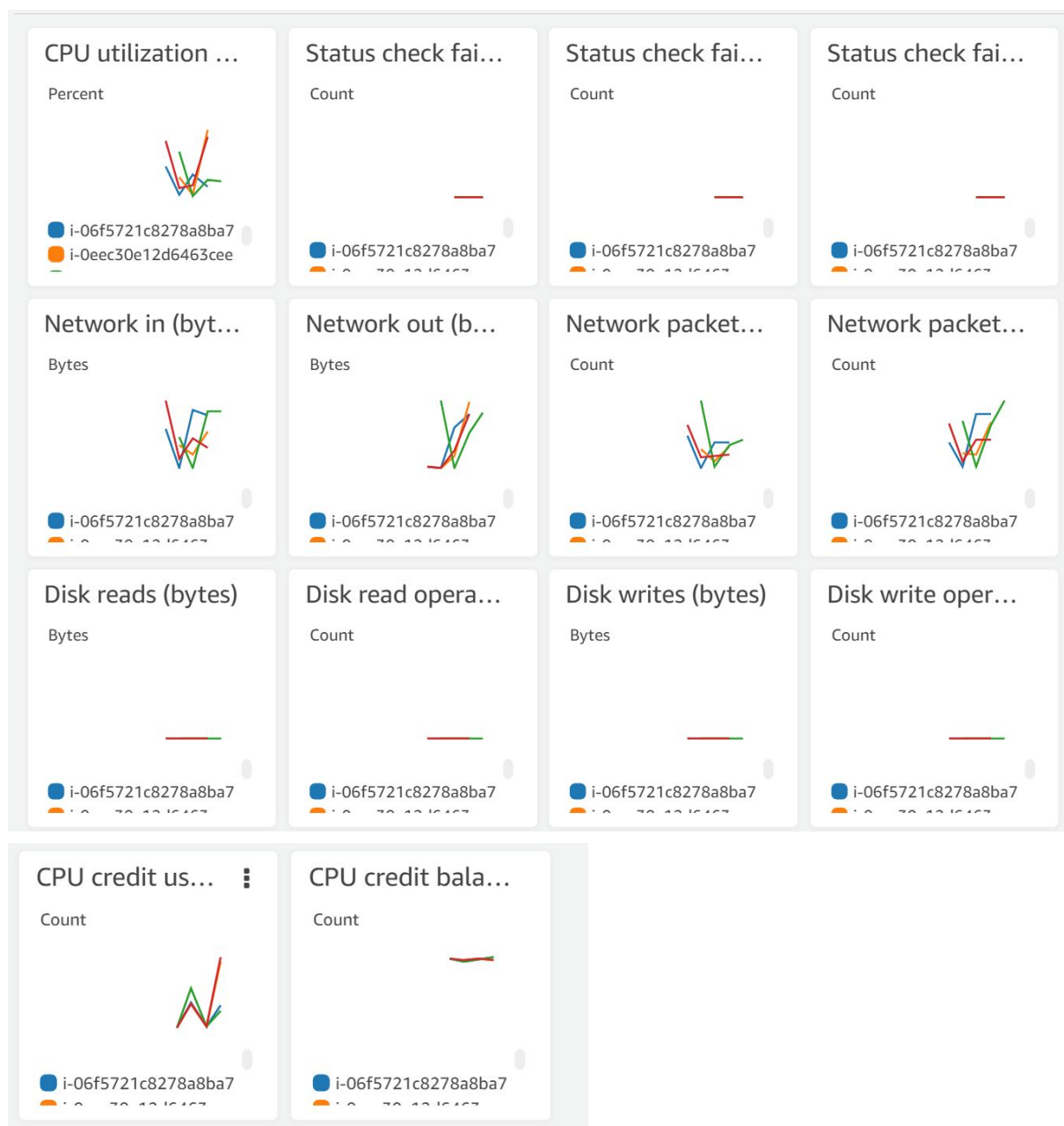


figure 6.3.1 3 data nodes resources utilization

## 4 data nodes

Purple line is the name node.



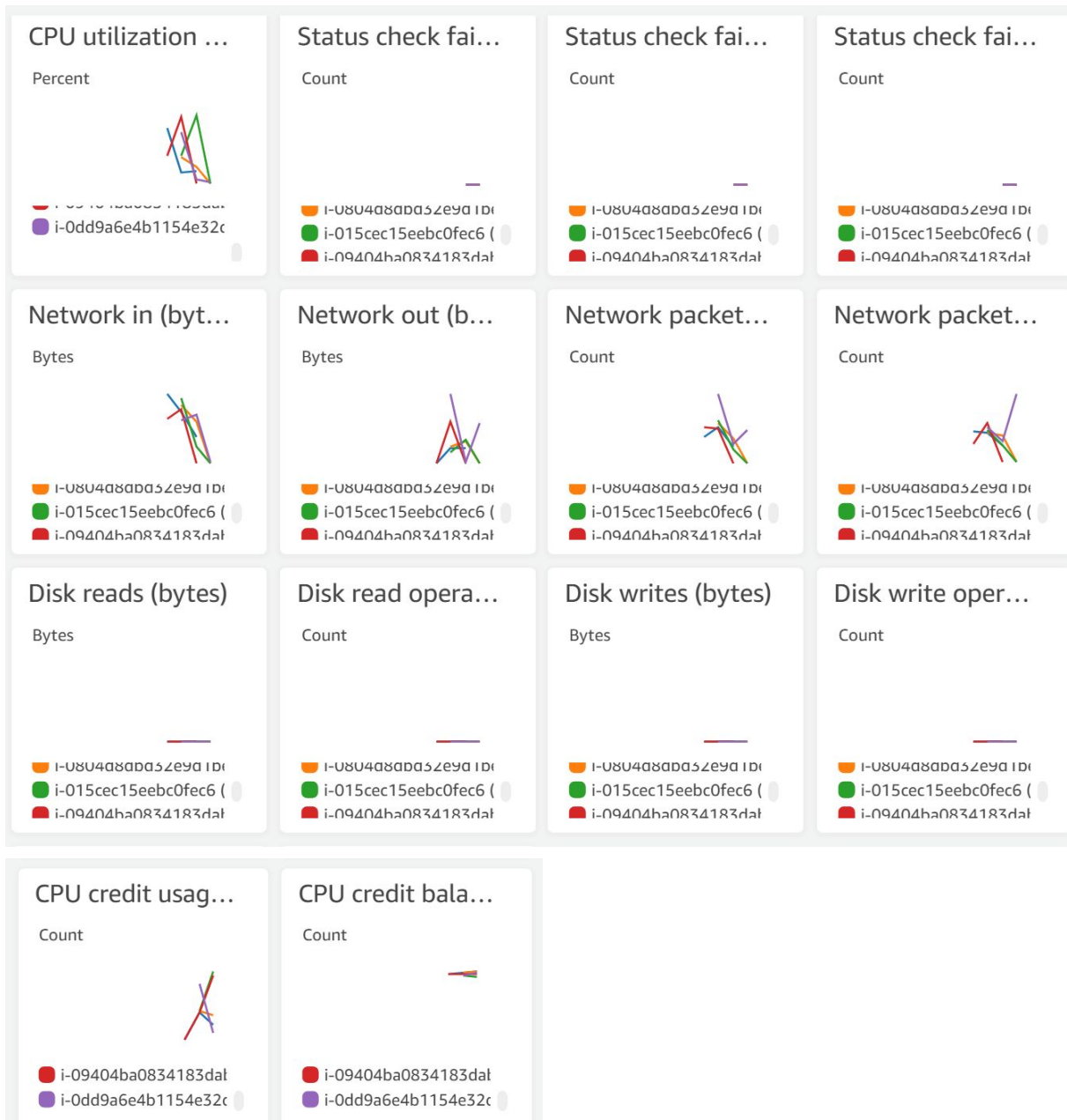


figure 6.3.2 4 data nodes resources utilization