ETL

Are you an aspiring Big Data Engineer or Developer interested in creating Data Pipelines for serving Data Warehouses and Data Analytics platforms? Would you like to learn all about ETL and ELT data pipelines and how to build them using Bash scripting and cutting-edge open-source tools such as Apache Airflow and Apache Kafka? This course may be just right for you.

ETL stands for Extract, Transform, and Load. It refers to the process of curating data from multiple sources and preparing the data for integration and loading into a destination platform such as a data warehouse or analytics environment. ELT is similar but loads the data in its raw format, reserving the transformations for people to apply themselves in a 'self-serve analytics' destination environment. Both methods are typical examples of data pipeline deployments.

In this course, you will explore the fundamental principles and techniques behind ETL and ELT processes. You will learn how to construct a basic ETL data pipeline from scratch using Bash shell-scripting. You will also learn about the tools, technologies, and use cases for the two main paradigms within data pipeline engineering: batch and streaming data pipelines. You will further cement this knowledge by exploring and applying two popular open-source data pipeline tools: Apache Airflow and Apache Kafka.

You will learn all about Apache Airflow and use it to build, put into production, and monitor a basic batch ETL workflow. You will implement this data pipeline using Airflow's central construct of a DAG (directed acyclic graph), consisting of simple Bash tasks and their dependencies.

You will also learn about Apache Kafka and use it to get hands-on experience with streaming data pipelines, implementing Kafka's message producers and consumers, and creating a Kafka weather topic.

This video and hands-on, lab-based course is made up of 5 modules: Data Processing Techniques; ETL & Data Pipelines: Tools and Techniques; Building Data Pipelines using Apache Airflow; Building Streaming Pipelines using Apache Kafka; and a Final Assignment. "ETL and Pipelines" will provide you with a great hands-on introduction to the latest technologies in data pipeline engineering.

Highlights:

- ETL stands for Extract, Transform, and Load
- Loading means writing the data to its destination environment
- Cloud platforms are enabling ELT to become an emerging trend
- The key differences between ETL and ELT include the place of transformation, flexibility, Big Data support, and time-to-insight
- There is an increasing demand for access to raw data that drives the evolution from ETL, which is still used, to ELT, which enables ad-hoc, self-serve analytics
- Data extraction often involves advanced technology including database querying, web scraping, and APIs
- Data transformation, such as typing, structuring, normalizing, aggregating, and cleaning, is about formatting data to suit the application
- Information can be lost in transformation processes through filtering and aggregation
- Data loading techniques include scheduled, on-demand, and incremental
- Data can be loaded in batches or streamed continuously

ETL Techniques

ETL stands for Extract, Transform, and Load, and refers to the process of curating data from multiple sources, conforming it to a unified data format or structure, and loading the transformed data into its new environment.

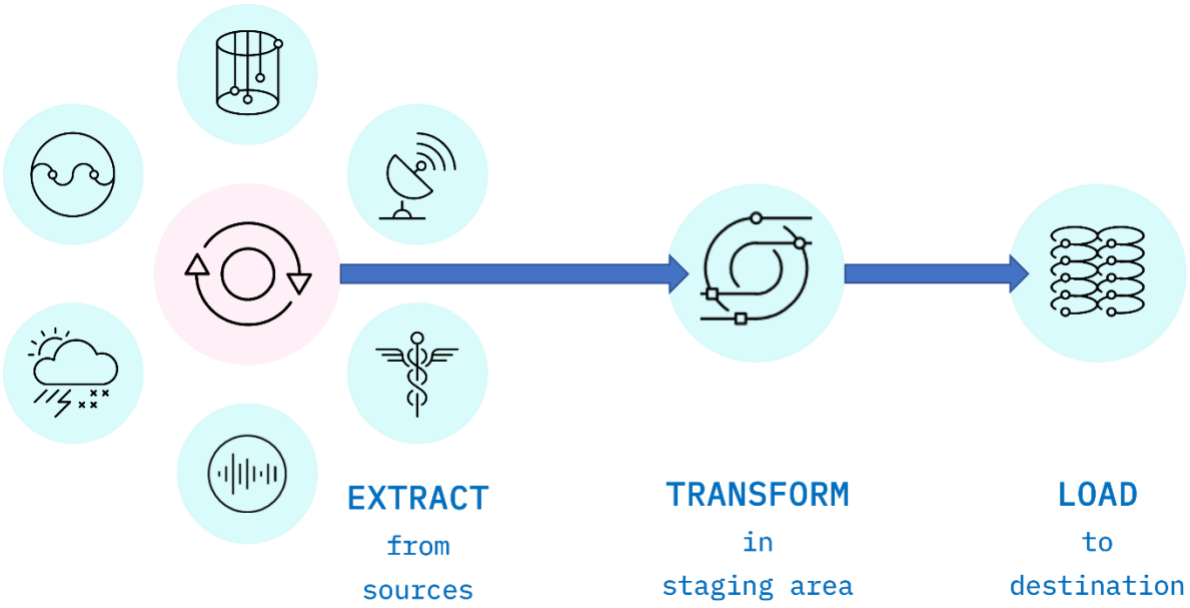


Fig. 1. ETL is an acronym used to describe the main processes behind a data pipeline design methodology that stands for Extract-Transform-Load. Data is extracted from disparate sources to an intermediate staging area where it is integrated and prepared for loading into a destination such as a data warehouse.

Extract

Data extraction is the first stage of the ETL process, where data is acquired from various source systems. The data may be completely raw, such as sensor data from IoT devices, or perhaps it is unstructured data from scanned medical documents or company emails. It may be streaming data coming from a social media network or near real-time stock market buy/sell transactions, or it may come from existing enterprise databases and data warehouses.

Transform

The transformation stage is where rules and processes are applied to the data to prepare it for loading into the target system. This is normally done in an

intermediate working environment called a “staging area.” Here, the data are cleaned to ensure reliability and conformed to ensure compatibility with the target system.

Many other transformations may be applied, including:

- Cleaning: fixing any errors or missing values
- Filtering: selecting only what is needed
- Joining: merging disparate data sources
- Normalizing: converting data to common units
- Data Structuring: converting one data format to another, such as JSON, XML, or CSV to database tables
- Feature Engineering: creating KPIs for dashboards or machine learning
- Anonymizing and Encrypting: ensuring privacy and security
- Sorting: ordering the data to improve search performance
- Aggregating: summarizing granular data

Load

The load phase is all about writing the transformed data to a target system. The system can be as simple as a comma-separated file, which is essentially just a table of data like an Excel spreadsheet. The target can also be a database, which may be part of a much more elaborate system, such as a data warehouse, a data mart, data lake, or some other unified, centralized data store forming the basis for analysis, modeling, and data-driven decision making by business analysts, managers, executives, data scientists, and users at all levels of the enterprise.

In most cases, as data is being loaded into a database, the constraints defined by its schema must be satisfied for the workflow to run successfully. The schema, a set of rules called integrity constraints, includes rules such as uniqueness, referential integrity, and mandatory fields. Thus such requirements imposed on the loading phase help ensure overall data quality.

ETL Workflows as Data Pipelines

Generally, an ETL workflow is a well thought out process that is carefully engineered to meet technical and end-user requirements.

Traditionally, the overall accuracy of the ETL workflow has been a more important requirement than speed, although efficiency is usually an important factor in minimizing resource costs. To boost efficiency, data is fed through a *data pipeline* in smaller packets (see Figure 2). While one packet is being extracted, an earlier packet is being transformed,

and another is being loaded. In this way, data can keep moving through the workflow without interruption. Any remaining bottlenecks within the pipeline can often be handled by parallelizing slower tasks.

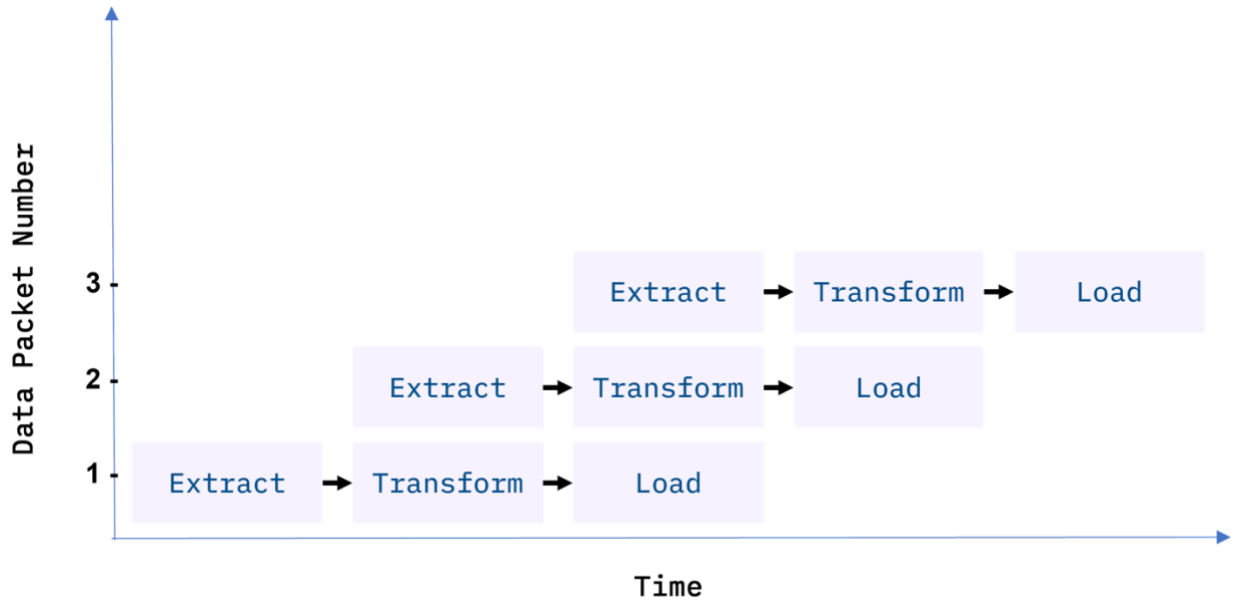


Fig 2. Data packets being fed in sequence, or "piped" through the ETL data pipeline. Ideally, by the time the third packet is ingested, all three ETL processes are running simultaneously on different packets.

With conventional ETL pipelines, data is processed in *batches*, usually on a repeating schedule that ranges from hours to days apart. For example, records accumulating in an Online Transaction Processing System (OLTP) can be moved as a daily batch process to one or more Online Analytics Processing (OLAP) systems where subsequent analysis of large volumes of historical data is carried out.

Batch processing intervals need not be periodic and can be triggered by events, such as

- when the source data reaches a certain size, or
- when an event of interest occurs and is detected by a system, such as an intruder alert, or
- on-demand, with web apps such as music or video streaming services

Staging Areas

ETL pipelines are frequently used to integrate data from disparate and usually siloed systems within the enterprise. These systems can be from different vendors, locations, and divisions of the company, which can add significant operational complexity. As an example, (see Figure

3) a cost accounting OLAP system might retrieve data from distinct OLTP systems utilized by the separate payroll, sales, and purchasing departments.

ETL pipelines are frequently used to integrate data from disparate and usually *siloed* systems within the enterprise. These systems can be from different vendors, locations, and divisions of the company, which can add significant operational complexity. As an example, (see Figure 3) a cost accounting OLAP system might retrieve data from distinct OLTP systems utilized by the separate payroll, sales, and purchasing departments.

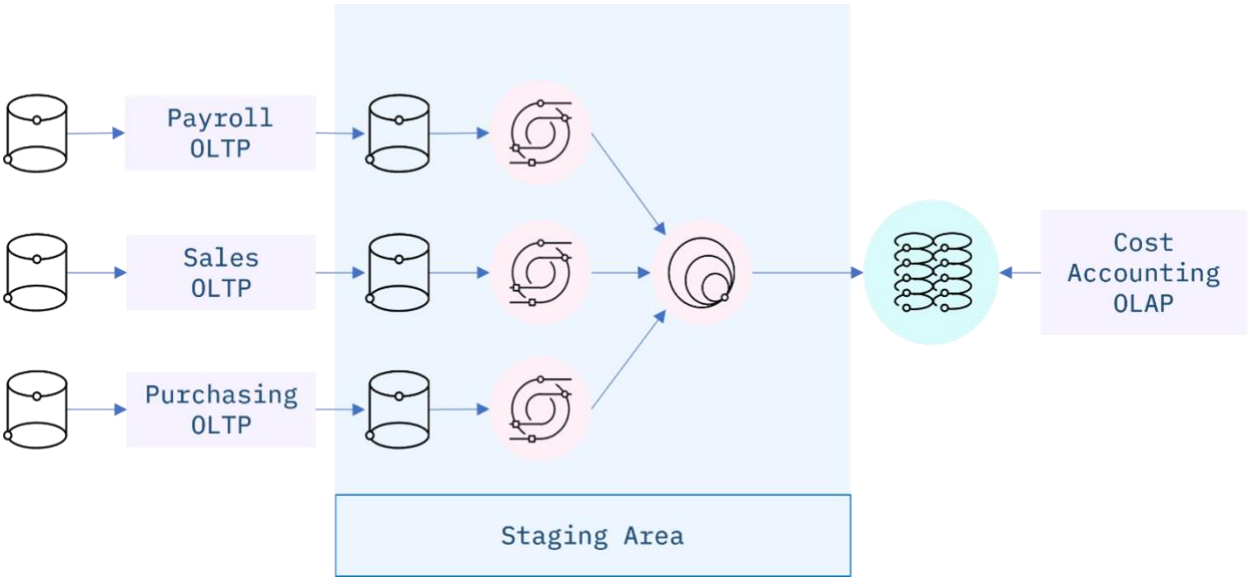


Fig 3. An ETL data integration pipeline concept for a Cost Accounting OLAP, fed by disparate OLTP systems within the enterprise. The staging area is used in this example to manage change detection of new or modified data from the source systems, data updates, and any transformations required to conform and integrate the data prior to loading to the OLAP.

ETL Workflows as DAGs

ETL workflows can involve considerable complexity. By breaking down the details of the workflow into individual tasks and dependencies between those tasks, one can gain better control over that complexity. Workflow orchestration tools such as Apache Airflow do just that.

Airflow represents your workflow as a directed acyclic graph (DAG). A simple example of an Airflow DAG is illustrated in Figure 4. Airflow tasks can be expressed using predefined templates, called operators. Popular operators include Bash operators, for running Bash code, and Python operators for running Python code, which makes them extremely versatile for deploying ETL pipelines and many other kinds of workflows into production.

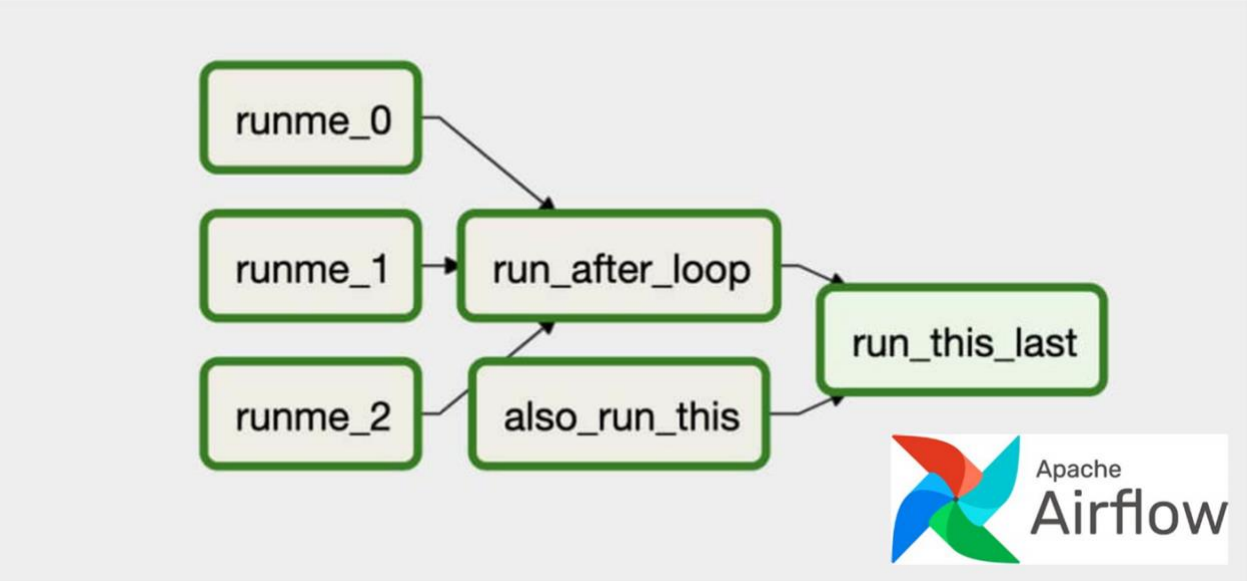


Fig 4. An Apache Airflow DAG representing a workflow. The green boxes represent individual tasks, while the arrows show dependencies between tasks. The three tasks on the left, 'runme_j' are jobs that run simultaneously along with the 'also_run_this' task. Once the 'runme_j' tasks complete, the 'run_after_loop' task starts. Finally, 'run_this_last' engages once all tasks have finished successfully.

Popular ETL tools

There are many ETL tools available today. Modern enterprise grade ETL tools will typically include the following features:

- Automation: Fully automated pipelines
- Ease of use: ETL rule recommendations
- Drag-and-drop interface: "o-code" rules and data flows
- Transformation support: Assistance with complex calculations
- Security and Compliance: Data encryption and HIPAA, GDPR compliance

Some well-known ETL tools are listed below, along with some of their key features. Both commercial and open-source tools are included in the list.

talend

Talend Open Studio

- Supports big data, data warehousing, and profiling
- Includes collaboration, monitoring, and scheduling
- Drag-and-drop GUI for ETL pipeline creation
- Automatically generates Java code

- Integrates with many data warehouses
- Open-source

AWS Glue

- ETL service that simplifies data prep for analytics
- Suggests schemas for storing your data
- Create ETL jobs from the AWS Console

IBM InfoSphere DataStage

- A data integration tool for designing, developing, and running ETL and ELT jobs
- The data integration component of IBM InfoSphere Information Server
- Drag-and-drop graphical interface
- Uses parallel processing and enterprise connectivity in a highly scalable platform

Alteryx

- Self-service data analytics platform
- Drag-and-drop accessibility to ETL tools
- No SQL or coding required to create pipelines

Apache Airflow and Python

- Versatile “configuration” as code data pipeline platform
- Open-sourced by Airbnb
- Programmatically author, schedule, and monitor workflows
- Scales to Big Data
- Integrates with cloud platforms

The Pandas Python library

- Versatile and popular open-source programming tool
- Based on data frames – table-like structures
- Great for ETL, data exploration, and prototyping
- Doesn’t readily scale to Big Data

ETL using Shell Scripting

Temperature Reporting Scenario:

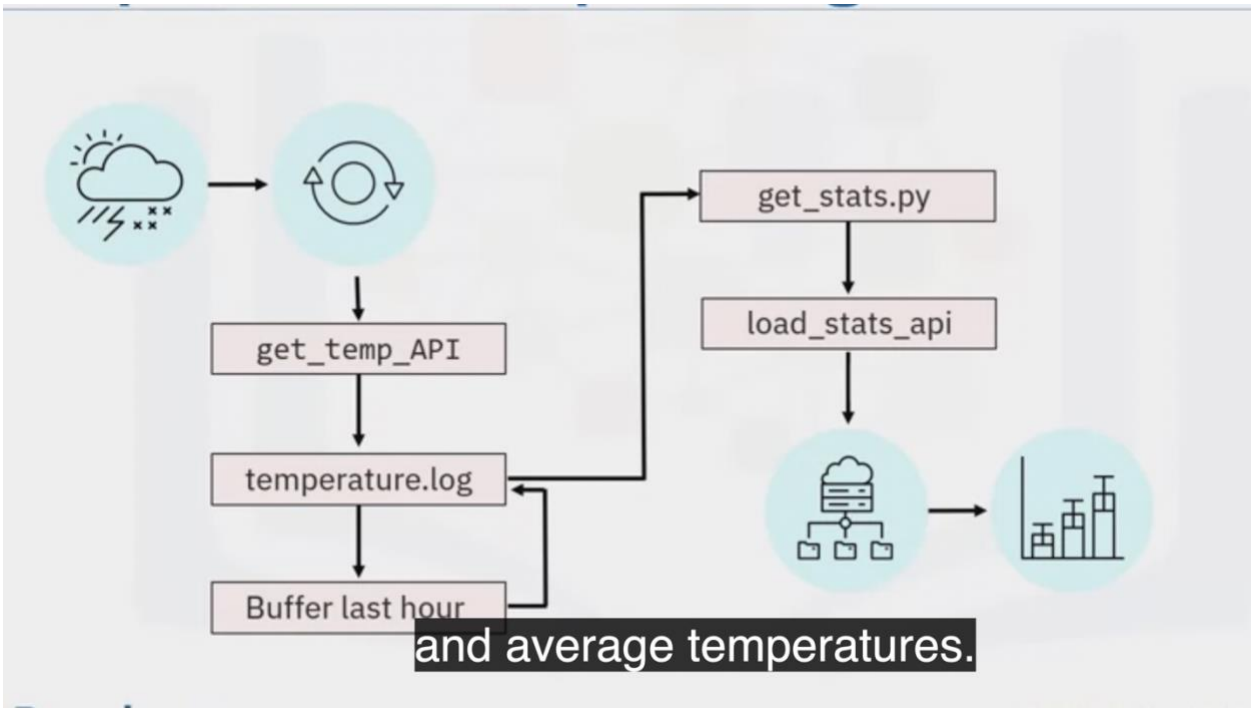
Task: Report Temperature statistics of a remote location

- Hourly average, min. and max. temperature
- From remote temp. sensor
- Update every minute

Given:

- ‘get_temp_API’ – API that reads temperature
- ‘load_stats_API’ – API that loads data for analysis

WORKFLOW:



Create an ETL Shell Script:

$ touch Temperature_ETL.sh
$ gedit Temperature_ETL.sh

In editor: #! /bin/bash

Extract reading with ‘get_temp_API’

Append reading to temperature.log

Buffer last hour of reading

Call get_stats.py to aggregate the results

Load the stats using ‘load_stats_API’

LAB:

Exercise 1 - Extracting data using `cut` command

The filter command `cut` helps us extract selected characters or fields from a line of text.

1.1 Extracting characters.

The command below shows how to extract the first four characters.

```
echo "database" | cut -c1-4
```

You should get the string 'data' as output.

The command below shows how to extract 5th to 8th characters.

```
echo "database" | cut -c5-8
```

You should get the string 'base' as output.

Non-contiguous characters can be extracted using the comma.

The command below shows how to extract the 1st and 5th characters.

```
echo "database" | cut -c1,5
```

You get the output : 'db'

1.2. Extracting fields/columns

We can extract a specific column/field from a delimited text file, by mentioning

- the delimiter using the `-d` option, or
- the field number using the `-f` option.

The /etc/passwd is a ":" delimited file.

The command below extracts user names (the first field) from /etc/passwd.

```
cut -d":" -f1 /etc/passwd
```

The command below extracts multiple fields 1st, 3rd, and 6th (username, userid, and home directory) from /etc/passwd.

```
cut -d":" -f1,3,6 /etc/passwd
```

The command below extracts a range of fields 3rd to 6th (userid, groupid, user description and home directory) from /etc/passwd.

```
cut -d":" -f3-6 /etc/passwd
```

Exercise 2 - Transforming data using `tr`.

`tr` is a filter command used to translate, squeeze, and/or delete characters.

2.1. Translate from one character set to another

The command below translates all lower case alphabets to upper case.

```
echo "Shell Scripting" | tr "[a-z]" "[A-Z]"
```

You could also use the pre-defined character sets also for this purpose:

```
echo "Shell Scripting" | tr "[:lower:]" "[:upper:]"
```

The command below translates all upper case alphabets to lower case.

```
echo "Shell Scripting" | tr  "[A-Z]" "[a-z]"
```

2.2. Squeeze repeating occurrences of characters

The **-s** option replaces a sequence of a repeated characters with a single occurrence of that character.

The command below replaces repeat occurrences of 'space' in the output of `ps` command with one 'space'.

```
ps | tr -s " "
```

In the above example, the space character within quotes can be replaced with the following : `"[:space:]"`.

2.3. Delete characters

We can delete specified characters using the `-d` option.

The command below deletes all digits.

```
echo "My login pin is 5634" | tr -d "[:digit:]"
```

The output will be : 'My login pin is'

Exercise 3 - Start the PostgreSQL database.

On the terminal run the following command to start the PostgreSQL database.

```
start_postgres
```

Note down the access information presented towards the end of these messages, especially the **CommandLine:**.

A sample commandline displayed looks as given below.

```
`psql --username=postgres --host=localhost`
```

Running this command from the shell prompt will start the interactive `psql` client which connects to the PostgreSQL server.

Exercise 4 - Create a table

In this exercise we will create a table called '**users**' in the PostgreSQL database. This table will hold the user account information.

The table 'users' will have the following columns:

1. uname
2. uid
3. home

Step 1: Connect to the database server

Use the connection string saved in the previous exercise to connect to the PostgreSQL server.Run the command below to login to PostgreSQL server.

```
psql --username=postgres --host=localhost
```

You will get the psql prompt: 'postgres=#'

Step 2: Connect to a database.

We will use a database called **template1** which is already available by default.

To connect to this database, run the following command at the 'postgres=#' prompt.

```
\c template1
```

You will get the following message.

```
You are now connected to database "template1" as user "postgres".
```

Also, your prompt will change to 'template1=#'.

Step 3: Create the table. Run the following statement at the 'template1=#' prompt:

```
create table users(username varchar(50),userid int,homedirectory varchar(100)
);
```

If the table is created successfully, you will get the message below.

```
CREATE TABLE
```

Step 4: Quit the psql client

To exit the psql client and come back to the Linux shell, run the following command:

```
\q
```

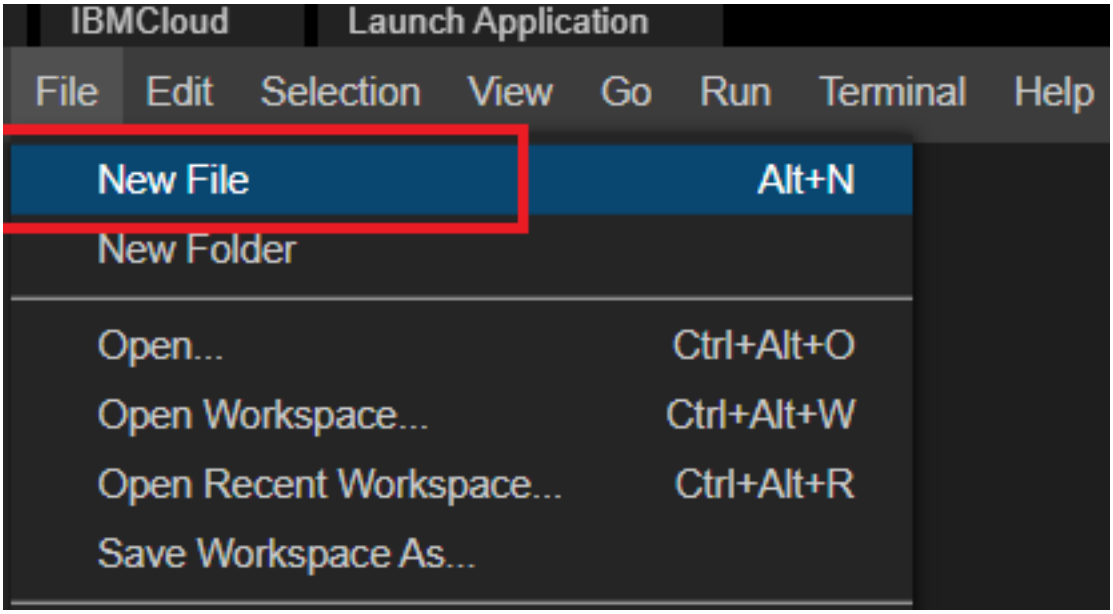
Exercise 5 - Loading data into a PostgreSQL table.

In this exercise, we will create a shell script which does the following.

- Extract the user name, user id, and home directory path of each user account defined in the /etc/passwd file.
- Save the data into a comma separated (CSV) format.
- Load the data in the csv file into a table in PostgreSQL database.

5.1. Create a shell script

Step 1: On the menu on the lab screen, use **File->New File** to create a new file.



Step 2: Give the name as 'csv2db.sh' and click 'OK'.

Step 3: State the objective of the script using comments.

Copy and paste the following lines into the newly created file.

```
# This script

# Extracts data from /etc/passwd file into a CSV file.
```

```
# The csv data file contains the user name, user id and

# home directory of each user account defined in /etc/passwd


# Transforms the text delimiter from ":" to ",".

# Loads the data from the CSV file into a table in PostgreSQL database.
```

Step 4: Save the file using the **File->Save** menu option.

5.2. Extract required user information from /etc/passwd

In this step, we will extract user name (field 1), user id (field 3), and home directory path (field 6) from /etc/passwd file using the `cut` command.

The /etc/passwd has ":" symbol as the column separator.

Copy the following lines and add them to the end of the script.

```
# Extract phase


echo "Extracting data"


# Extract the columns 1 (user name), 2 (user id) and

# 6 (home directory path) from /etc/passwd


cut -d":" -f1,3,6 /etc/passwd
```

Save the file.

Run the script.

```
bash csv2db.sh
```

Verify that the output contains the three fields, that we extracted.

5.3. Redirect the extracted output into a file.

In this step, we will redirect the extracted data into a file named `extracted-data.txt`

Replace the cut command at end of the script with the following command.

```
cut -d":" -f1,3,6 /etc/passwd > extracted-data.txt
```

Save the file.

Run the script.

```
bash csv2db.sh
```

Run the command below to verify that the file `extracted-data.txt` is created, and has the content.

```
cat extracted-data.txt
```

5.4. Transform the data into CSV format

The extracted columns are separated by the original ":" delimiter.

We need to convert this into a "," delimited file.

Add the below lines at the end of the script

```
# Transform phase
echo "Transforming data"
# read the extracted data and replace the colons with commas.


tr ":" "," < extracted-data.txt
```

Save the file.

Run the script.

```
bash csv2db.sh
```

Verify that the output contains ',' in place of ":".

Replace the tr command at end of the script with the command below.

```
tr ":" "," < extracted-data.txt > transformed-data.csv
```

Save the file.

Run the script.

```
bash csv2db.sh
```

Run the command below to verify that the file `transformed-data.csv` is created, and has the content.

```
cat transformed-data.csv
```

5.5. Load the data into the table 'users' in PostgreSQL

To load data from a shell script, we will use the `psql` client utility in a non-interactive manner.

This is done by sending the database commands through a command pipeline to `psql` with the help of `echo` command.

Step 1: Add the copy command

PostgreSQL command to copy data from a CSV file to a table is `COPY`.

The basic structure of the command which we will use in our script is,

```
COPY table_name FROM 'filename' DELIMITERS 'delimiter_character' FORMAT;
```

Now, add the lines below to the end of the script 'csv2db.sh'.

```
# Load phase

echo "Loading data"

# Send the instructions to connect to 'template1' and

# copy the file to the table 'users' through command pipeline.



echo "\c template1;\COPY users  FROM '/home/project/transformed-data.csv' DEL
IMITERS ',' CSV;" | psql --username=postgres --host=localhost
```

Save the file.

Exercise 6 - Execute the final script

Run the script.

```
bash csv2db.sh
```

Run the command below to verify that the table users is populated with the data.

```
echo '\c template1; \\SELECT * from users;' | psql --username=postgres --host
=localhost
```

Congratulations! You have created an ETL script using shell scripting.

Practice exercises

1. Problem:

Copy the data in the file 'web-server-access-log.txt.gz' to the table 'access_log' in the PostgreSQL database 'template1'.

The file is available at the location : "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Bash%20Scripting/ETL%20using%20shell%20scripting/web-server-access-log.txt.gz".

The following are the columns and their data types in the file:

a. timestamp - TIMESTAMP
b. latitude - float
c. longitude - float
d. visitorid - char(37)

and two more columns: accessed*from*mobile (boolean) and browser_code (int)

The columns which we need to copy to the table are the first four coumns : timestamp, latitude, longitude and visitorid.

NOTE: The file comes with a header. So use the 'HEADER' option in the 'COPY' command.

The problem may be solved by completing the following tasks:
Task 1: Start the Postgres server.

Click here for Hint
Click here for Solution
If the server is not already started, run the following command:

```
start_postgres
```

Task 2: Create the table.

Create a table named `access_log` to store the timestamp, latitude, longitude and visitorid.

Click here for Hint
Click here for Solution

Step 1: Connect to the database:

Run the following command at the terminal to connect to Postgres

```
psql --username=postgres --host=localhost
```

Step 2: At the `postgres=#` prompt, run the following command to connect to the database 'template1'.

```
\c template1;
```

Step 3: Once you connect to the database, run the command to create the table called 'access_log':

```
CREATE TABLE access_log(timestamp TIMESTAMP, latitude float, longitude float, visitor_id char(37));
```

Step 4: Once you receive the confirmation message 'CREATE TABLE', quit from psql:

```
\q
```

Task 3. Create a shell script named `cp-access-log.sh` and add commands to complete the remaining tasks to extract and copy the data to the database.

Create a shell script to add commands to complete the rest of the tasks.

Click here for Hint
Click here for Solution
Step 1: On the menu on the lab screen, use **File->New File** to create a new file.

Step 2: Give the name as `cp-access-log.sh` and click 'OK'.

Step 3: State the objective of the script using comments.

Copy and paste the following lines into the newly created file.

```
# cp-access-log.sh

# This script downloads the file 'web-server-access-log.txt.gz'

# from "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Bash%20Scripting/ETL%20using%20shell%20scripting/".



# The script then extracts the .txt file using gunzip.
```

```
# The .txt file contains the timestamp, latitude, longitude

# and visitor id apart from other data.



# Transforms the text delimeter from "#" to "," and saves to a csv file.

# Loads the data from the CSV file into the table 'access_log' in PostgreSQL
database.
```

Step 4: Save the file using the **File->Save** menu option.

Task 4. Download the access log file.

Add the `wget` command to the script to download the file.

Click here for Hint
Click here for Solution
Add the following line to the end of the script.

```
# Download the access log file



wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-
DB0250EN-SkillsNetwork/labs/Bash%20Scripting/ETL%20using%20shell%20scripting/
web-server-access-log.txt.gz"
```

Task 5. Unzip the gzip file.

Run the gunzip command to unzip the .gz file and extract the .txt file.

Click here for Hint
Click here for Solution
Copy the following lines to the end of the script.

```
# Unzip the file to extract the .txt file.

gunzip -f web-server-access-log.txt.gz
```

The **-f** option of gunzip is to overwrite the file if it already exists.

Task 6. Extract required fields from the file.

Extract timestamp, latitude, longitude and visitorid which are the first four fields from the file using the `cut` command.

The columns in the web-server-access-log.txt file is delimited by '#'.

Click here for Hint
Click here for Solution
Step 1: Copy the following lines and add them to the end of the script.

```
# Extract phase


echo "Extracting data"


# Extract the columns 1 (timestamp), 2 (latitude), 3 (longitude) and

# 4 (visitorid)


cut -d# -f1-4 web-server-access-log.txt
```

Step 2: Save the file.

Step 3: Run the script.

```
bash cp-access-log.sh
```

Verify that the output contains all the four fields that we extracted.

Task 7. Redirect the extracted output into a file.

Redirect the extracted data into a file named `extracted-data.txt`

Click here for Hint
Click here for Solution
Step 1: *Replace* the cut command at end of the script with the following command.

```
cut -d# -f1-4 web-server-access-log.txt > extracted-data.txt
```

Step 2: Save the file.

Step 3: Run the script.

```
bash cp-access-log.sh
```

Step 4: Run the command below to verify that the file `extracted-data.txt` is created, and has the content.

```
cat extracted-data.txt
```

Task 8. Transform the data into CSV format.

The extracted columns are separated by the original "#" delimiter.

We need to convert this into a "," delimited file.

Click here for Hint
Click here for Solution
Step 1: Add the lines below at the end of the script.

```
# Transform phase

echo "Transforming data"



# read the extracted data and replace the colons with commas.

tr "#" "," < extracted-data.txt
```

Step 2: Save the file.

Step 3: Run the script.

```
bash cp-access-log.sh
```

Step 4: Verify that the output contains ',' in place of "#".

Now we need to save the transformed data to a .csv file.

Step 5: *Replace* the tr command at end of the script with the command below.

```
tr "#" "," < extracted-data.txt > transformed-data.csv
```

Step 6: Save the file.

Step 7: Run the script.

```
bash cp-access-log.sh
```

Step 8: Run the command below to verify that the file 'transformed-data.csv' is created, and has the content.

```
cat transformed-data.csv
```

Task 9. Load the data into the table `access_log` in PostgreSQL

PostgreSQL command to copy data from a CSV file to a table is `COPY`.

The basic structure of the command is,

```
COPY table_name FROM 'filename' DELIMITERS 'delimiter_character' FORMAT;
```

The file comes with a header. So use the 'HEADER' option in the 'COPY' command.

Invoke this command from the shellscript, by sending it as input to 'psql' filter command.

Click here for Hint
Click here for Solution
Step 1: Add the copy command

Add the lines below to the end of the script 'cp-access-log.sh'.

```
# Load phase

echo "Loading data"



# Send the instructions to connect to 'template1' and

# copy the file to the table 'access_log' through command pipeline.



echo "\c template1;\COPY access_log  FROM '/home/project/transformed-data.csv
' DELIMITERS ',' CSV HEADER;" | psql --username=postgres --host=localhost
```

Step 2: Save the file.

Task 10. Execute the final script.

Run the final script.

Click here for Solution
Run the following command at the terminal:

```
bash cp-access-log.sh
```

The bash script can be downloaded from **here**

Task 11. Verify by querying the database.

Click here for Hint
Click here for Solution
Run the command below at the shell prompt to verify that the table accesss_log is populated with the data.

```
echo '\c template1; \\SELECT * from access_log;' | psql --username=postgres --host=localhost
```

You should see the records displayed on screen.

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

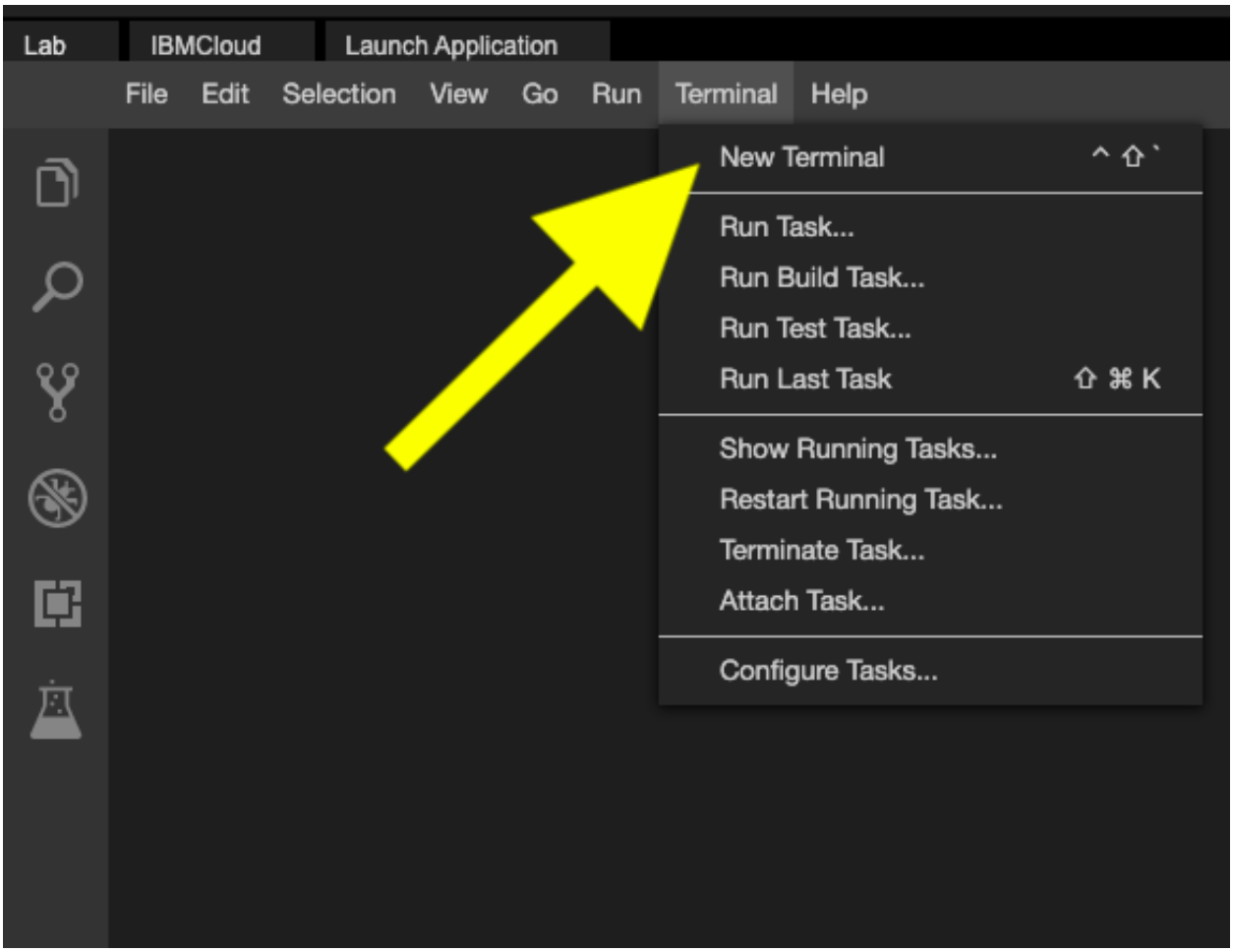
Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-09-06	0.2	Ramesh Sannareddy	Incorporated the beta feedback.
2021-06-07	0.1	Ramesh Sannareddy	Created initial version of the lab

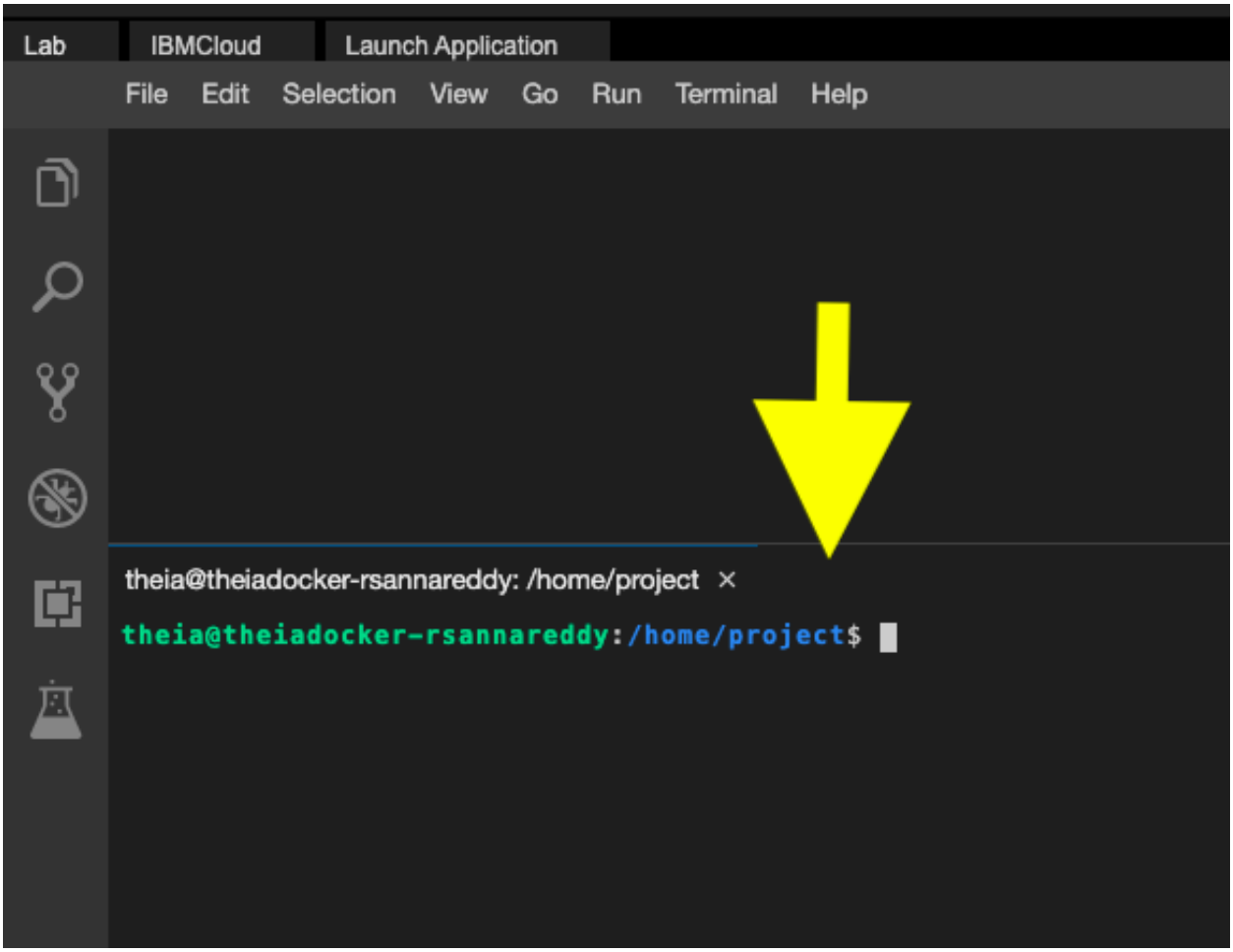
Apache Airflow

Exercise 1 - Start Apache Airflow

Open a new terminal by clicking on the menu bar and selecting **Terminal**->**New Terminal**, as shown in the image below.



This will open a new terminal at the bottom of the screen as in the image below.



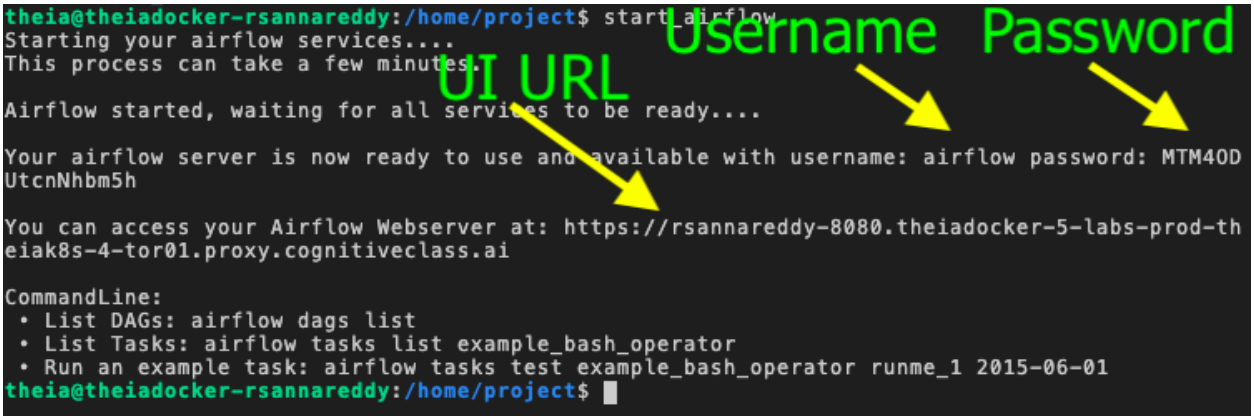
Run the commands below on the newly opened terminal. (You can copy the code by clicking on the little copy button on the bottom right of the codeblock below and paste it wherever you wish.)

Run the command below in the terminal to start Apache Airflow.

```
start_airflow
```

Please be patient, it may take upto 5 minutes for airflow to get started.

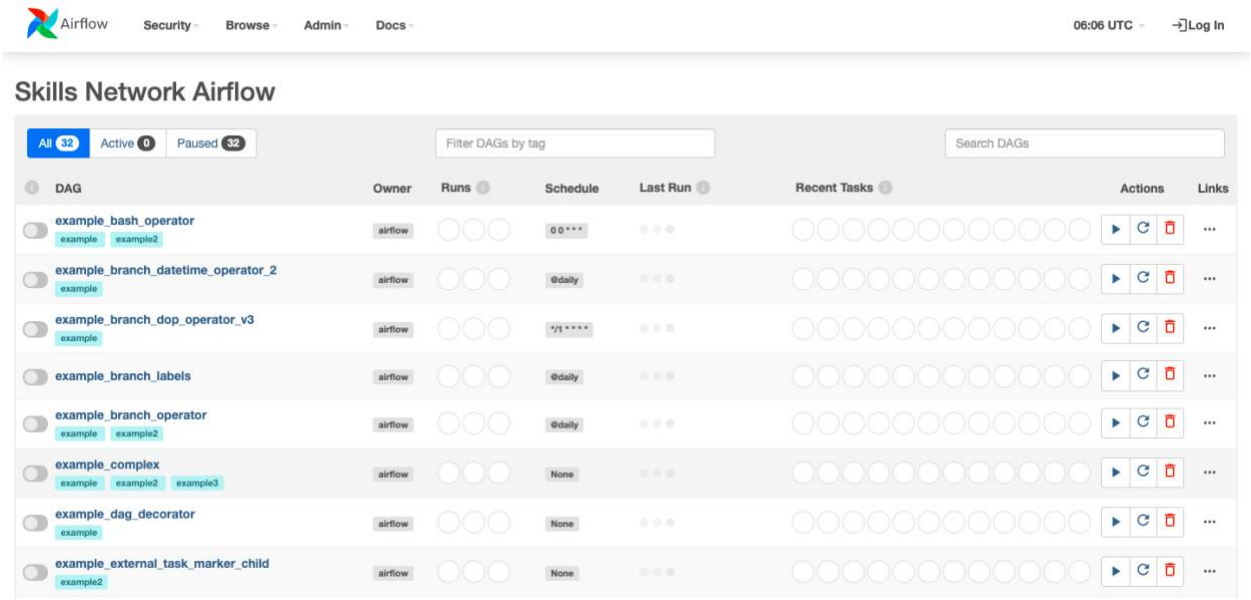
When airflow starts successfully, you should see an output similar to the one below.


theia@theiadocker-rsannareddy:/home/project$ start_airflow
Starting your airflow services....
This process can take a few minutes.
Airflow started, waiting for all services to be ready....
Your airflow server is now ready to use and available with username: airflow password: MTM4OD
UtcnNhbm5h
You can access your Airflow Webserver at: https://rsannareddy-8080.theiadocker-5-labs-prod-th
eiak8s-4-tor01.proxy.cognitiveclass.ai
CommandLine:
• List DAGs: airflow dags list
• List Tasks: airflow tasks list example_bash_operator
• Run an example task: airflow tasks test example_bash_operator runme_1 2015-06-01
theia@theiadocker-rsannareddy:/home/project$
UI URL
Username
Password

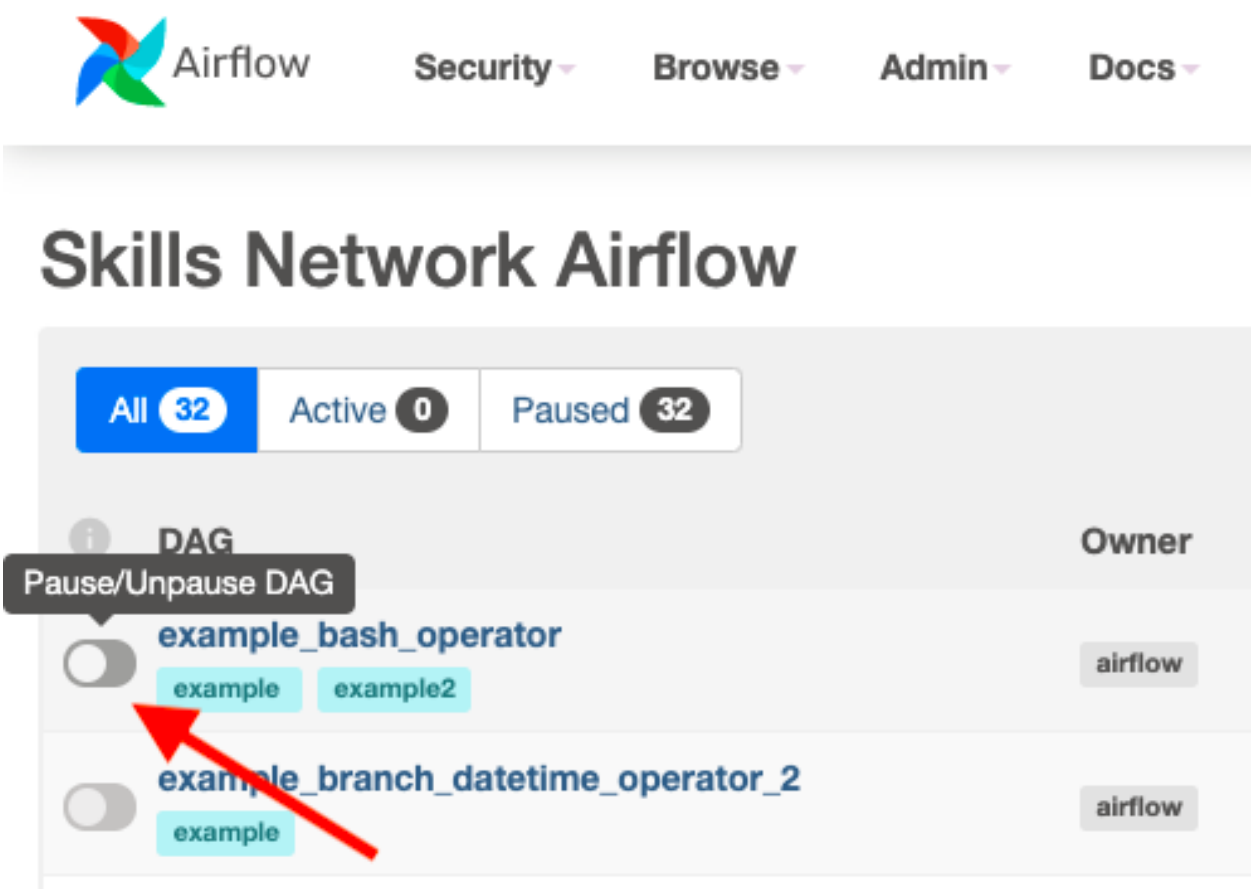
Exercise 2 - Open the Airflow Web UI

Copy the Web-UI URL and paste it on a new browser tab. You can also click on the URL by holding the control key (Command key in case of a Mac).

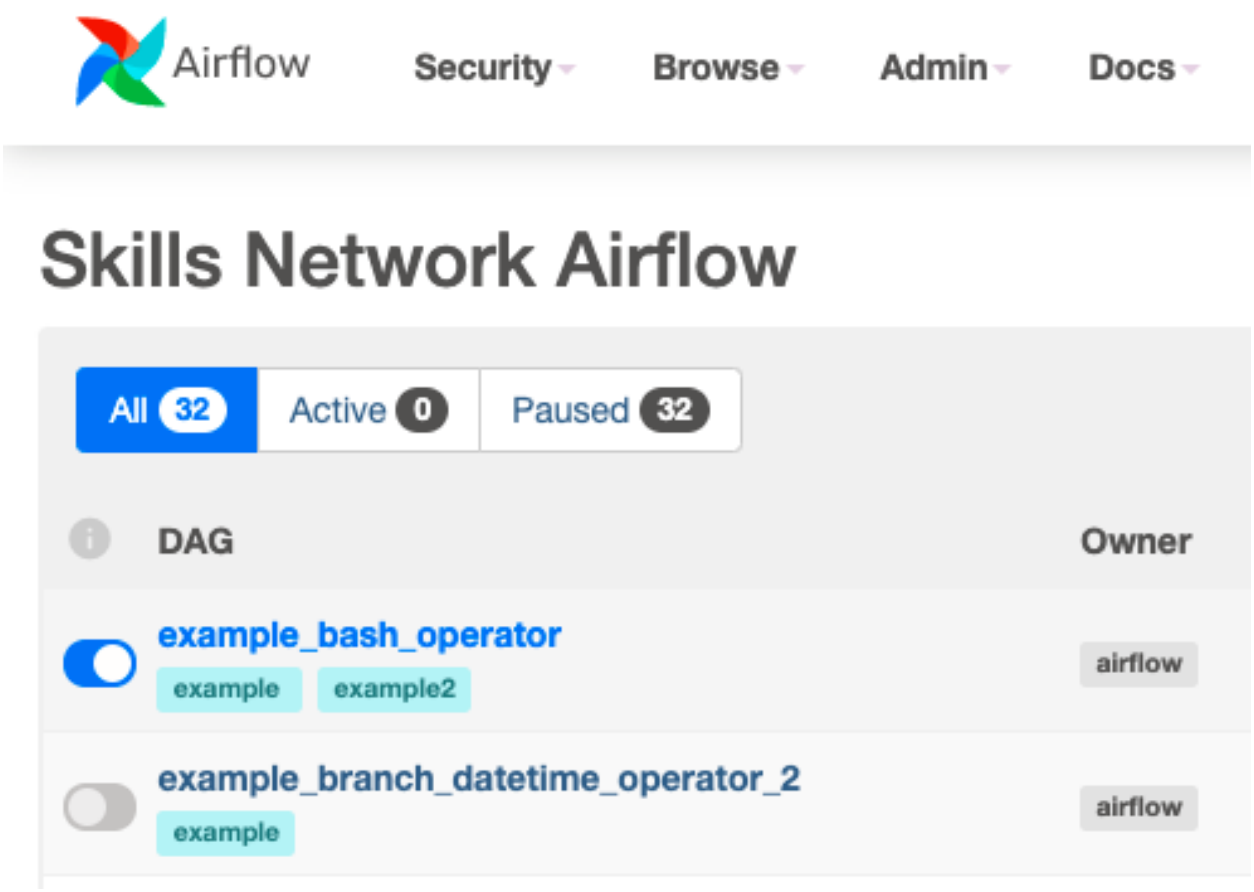
You should land at a page that looks like this:



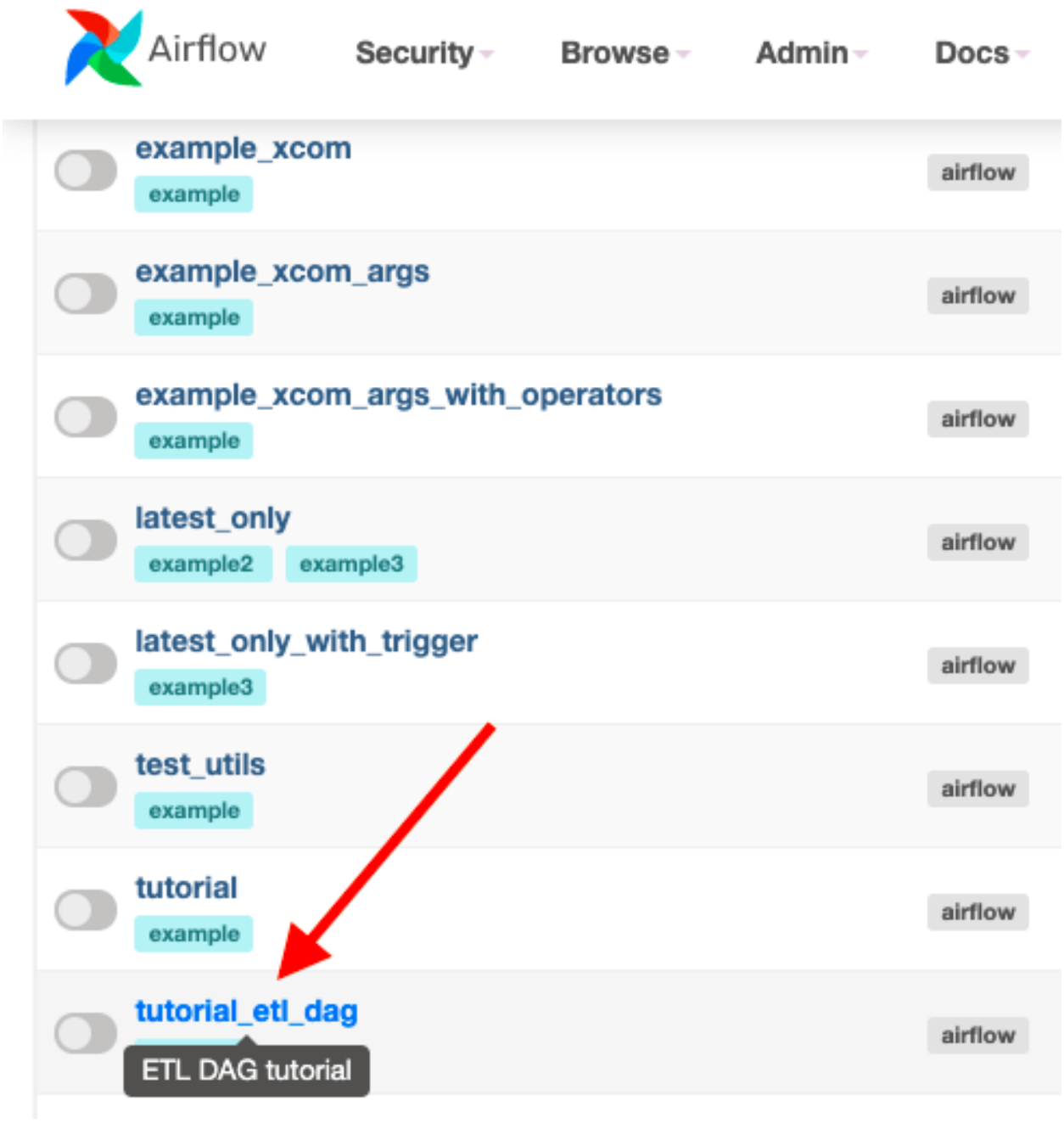
You can Unpause/Pause a DAG using the Unpause/Pause toggle button.



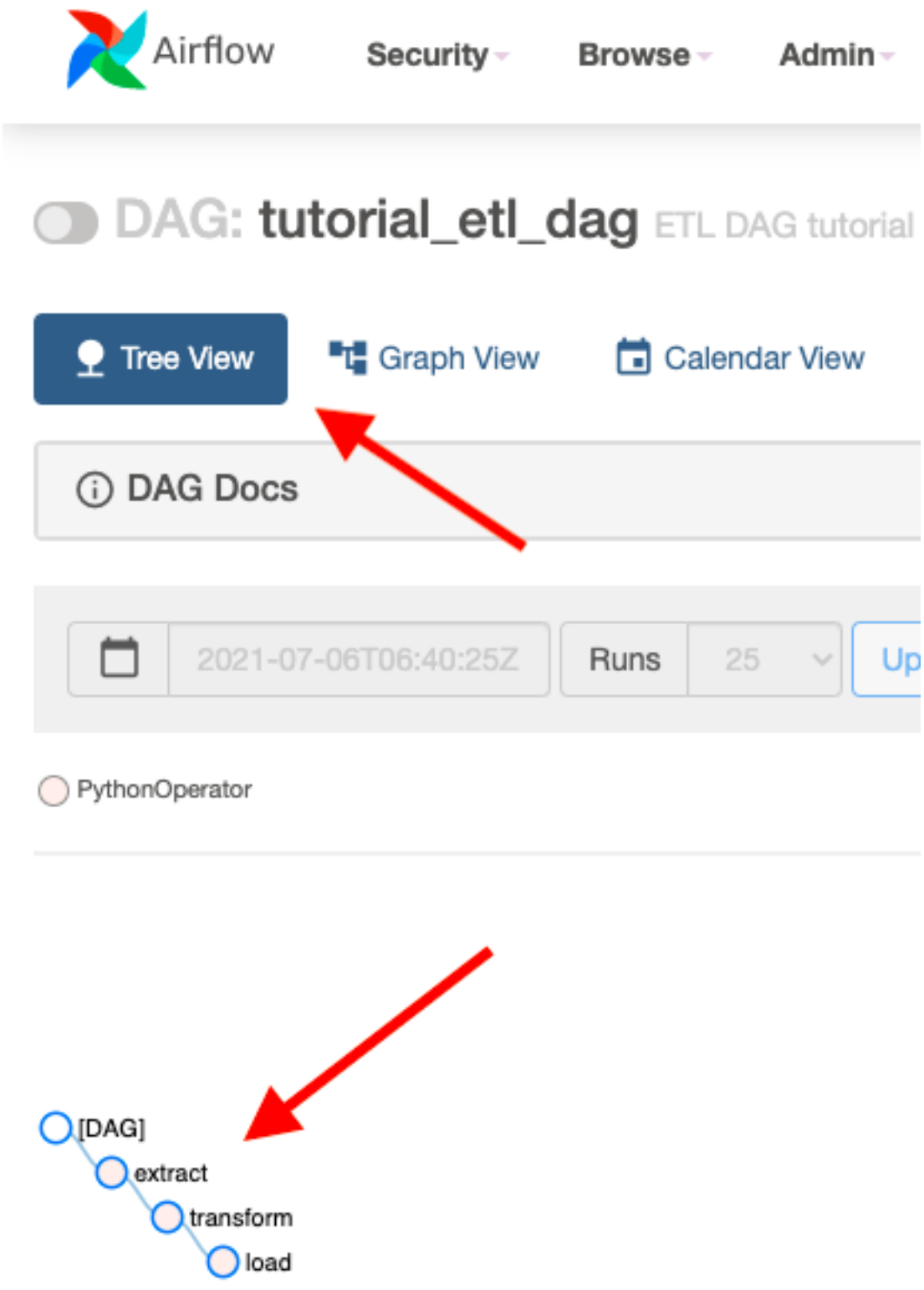
An unpaused DAG looks like this:



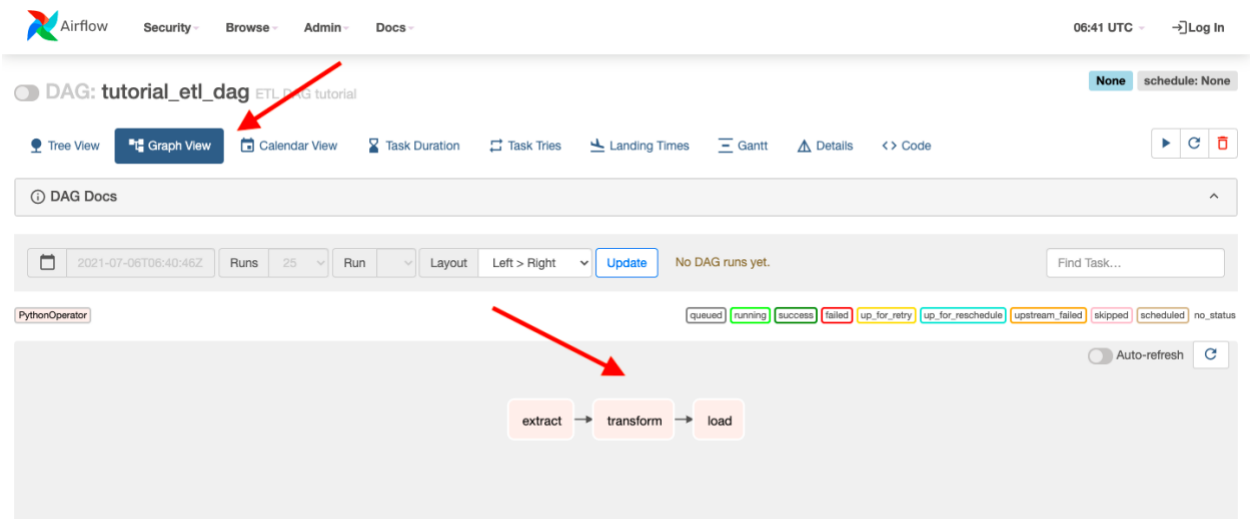
Click on a DAG to explore more about the DAG.



Click on the Tree View button to see the tree view of the DAG.



Click on the Graph View button to see the graph view of the DAG.



Practice exercises

1. Problem:

List tasks for the DAG `example_branch_labels`.
Click here for Hint
Click here for Solution

```
airflow tasks list example_branch_labels
```

2. Problem:

Unpause the DAG `example_branch_labels`.
Click here for Hint
Click here for Solution

```
airflow dags unpause example_branch_labels
```

3. Problem:

Pause the DAG `example_branch_labels`.
Click here for Hint
Click here for Solution

```
airflow dags pause example_branch_labels
```

Build a DAG using Airflow

DAG Python Script Blocks:

- Library Imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

Example: To print greeting and time

- Library Imports:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt
```

- DAG arguments:

```
default_args = {
        'owner' : 'me',
        'start_date' : dt.datetime(2021,7,28),
        'retries': 1,
        'retry_delay': dt. Timedelta(minutes=5),
}
```

- DAG definition:

```
dag=DAG('simple_example',
                description='A simple example DAG',
                default_args=default_args,
                schedule_interval = dt.timedelta(seconds=5),
)
```

- Task definitions:

```
Task1=BashOperator(
        Task_id='print_hello',
        Bash_command='echo \'Greetings. The date and time are\'',
        Dag=dag,
)

Task2=BashOperator(
        Task_id='print_date',
        Bash_command='date',
        Dag=dag,
)
```

- Task pipeline Specifications:

  ```
  task1 >> task2
  ```

Exercise 3 - Explore the anatomy of a DAG

An Apache Airflow DAG is a python program. It consists of these logical blocks.

- Imports

- DAG Arguments
- DAG Definition
- Task Definitions
- Task Pipeline

A typical `imports` block looks like this.

```
# import the libraries


from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG

from airflow import DAG

# Operators; we need this to write tasks!

from airflow.operators.bash_operator import BashOperator

# This makes scheduling easy

from airflow.utils.dates import days_ago
```

A typical `DAG Arguments` block looks like this.

```
#defining DAG arguments


# You can override them on a per-task basis during operator initialization

default_args = {

    'owner': 'Ramesh Sannareddy',

    'start_date': days_ago(0),

    'email': ['ramesh@somemail.com'],

    'email_on_failure': True,

    'email_on_retry': True,

    'retries': 1,

    'retry_delay': timedelta(minutes=5),

}
```

DAG arguments are like settings for the DAG.

The above settings mention

- the owner name,
- when this DAG should run from: days_age(0) means today,
- the email address where the alerts are sent to,
- whether alert must be sent on failure,
- whether alert must be sent on retry,
- the number of retries in case of failure, and
- the time delay between retries.

A typical `DAG definition` block looks like this.

```
# define the DAG

dag = DAG(

    dag_id='sample-etl-dag',

    default_args=default_args,

    description='Sample ETL DAG using Bash',

    schedule_interval=timedelta(days=1),

)
```

Here we are creating a variable named dag by instantiating the DAG class with the following parameters.

`sample-etl-dag` is the ID of the DAG. This is what you see on the web console.

We are passing the dictionary `default_args`, in which all the defaults are defined.

`description` helps us in understanding what this DAG does.

`schedule_interval` tells us how frequently this DAG runs. In this case every day. (`days=1`).

A typical `task definitions` block looks like this:

```
# define the tasks


# define the first task named extract

extract = BashOperator(
```

```
    task_id='extract',
    bash_command='echo "extract"',
    dag=dag,
)



# define the second task named transform
transform = BashOperator(
    task_id='transform',
    bash_command='echo "transform"',
    dag=dag,
)


# define the third task named load

load = BashOperator(
    task_id='load',
    bash_command='echo "load"',
    dag=dag,
)
```

A task is defined using:

- A task_id which is a string and helps in identifying the task.
- What bash command it represents.
- Which dag this task belongs to.

A typical `task pipeline` block looks like this:

```
# task pipeline
extract >> transform >> load
```

Task pipeline helps us to organize the order of tasks.

Here the task `extract` must run first, followed by `transform`, followed by the task `load`.

Exercise 4 - Create a DAG

Let us create a DAG that runs daily, and extracts user information from */etc/passwd* file, transforms it, and loads it into a file.

This DAG has two tasks `extract` that extracts fields from */etc/passwd* file and `transform_and_load` that transforms and loads data into a file.

```
# import the libraries


from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG

from airflow import DAG

# Operators; we need this to write tasks!

from airflow.operators.bash_operator import BashOperator

# This makes scheduling easy

from airflow.utils.dates import days_ago


#defining DAG arguments


# You can override them on a per-task basis during operator initialization

default_args = {

    'owner': 'Ramesh Sannareddy',

    'start_date': days_ago(0),

    'email': ['ramesh@somemail.com'],

    'email_on_failure': False,

    'email_on_retry': False,

    'retries': 1,
```

```
    'retry_delay': timedelta(minutes=5),
}


# defining the DAG


# define the DAG
dag = DAG(
    'my-first-dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(days=1),
)


# define the tasks


# define the first task


extract = BashOperator(
    task_id='extract',
    bash_command='cut -d":" -f1,3,6 /etc/passwd > extracted-data.txt',
    dag=dag,
)



# define the second task
transform_and_load = BashOperator(
    task_id='transform',
    bash_command='tr ":" "," < extracted-data.txt > transformed-data.csv',
```

```
    dag=dag,
)



# task pipeline
extract >> transform_and_load
```

Copy the code above and save it into a file named `my_first_dag.py`

Exercise 5 - Submit a DAG

Submitting a DAG is as simple as copying the DAG python file into `dags`folder in the `AIRFLOW_HOME` directory.

Open a terminal and run the command below to submit the DAG that was created in the previous exercise.

```
cp my_first_dag.py $AIRFLOW_HOME/dags
```

Verify that our DAG actually got submitted.

Run the command below to list out all the existing DAGs.

```
airflow dags list
```

Verify that `my-first-dag` is a part of the output.

```
airflow dags list|grep "my-first-dag"
```

You should see your DAG name in the output.

Run the command below to list out all the tasks in `my-first-dag`.

```
airflow tasks list my-first-dag
```

Practice exercises

1. Problem:

Write a DAG named `ETL_Server_Access_Log_Processing`.
***Task 1**: Create the imports block.*
***Task 2**: Create the DAG Arguments block. You can use the default settings*
***Task 3**: Create the DAG definition block. The DAG should run daily.*
***Task 4**: Create the download task.*
download task must download the server access log file which is available at the URL: **https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Apache%20Airflow/Build%20a%20DAG%20using%20Airflow/web-server-access-log.txt**

***Task 5**: Create the extract task.*
The server access log file contains these fields.

a. `timestamp` - TIMESTAMP
b. `latitude` - float
c. `longitude` - float
d. `visitorid` - char(37)
e. `accessed_from_mobile` - boolean
f. `browser_code` - int

The `extract` task must extract the fields `timestamp` and `visitorid`.

***Task 6**: Create the transform task.*
The `transform` task must capitalize the `visitorid`.

***Task 7**: Create the load task.*
The `load` task must compress the extracted and transformed data.

***Task 8**: Create the task pipeline block.*
The pipeline block should schedule the task in the order listed below:

1. download
2. extract
3. transform
4. load

***Task 10**: Submit the DAG.*
***Task 11**. Verify if the DAG is submitted*

Click here for Hint
Click here for Solution
Select File -> New File from the menu and name it as `ETL_Server_Access_Log_Processing.py`.

Add to the file the following parts of code to complete the tasks given in the problem.

Task 1: Create the imports block.

```
# import the libraries


from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG

from airflow import DAG

# Operators; we need this to write tasks!

from airflow.operators.bash_operator import BashOperator

# This makes scheduling easy

from airflow.utils.dates import days_ago
```

Task 2: Create the DAG Arguments block. You can use the default settings.

```
#defining DAG arguments


# You can override them on a per-task basis during operator initialization

default_args = {

    'owner': 'Ramesh Sannareddy',

    'start_date': days_ago(0),

    'email': ['ramesh@somemail.com'],

    'email_on_failure': False,

    'email_on_retry': False,

    'retries': 1,

    'retry_delay': timedelta(minutes=5),

}
```

Task 3: Create the DAG definition block. The DAG should run daily.

```
# defining the DAG


```

```
# define the DAG

dag = DAG(

    'etl-log-processsing-dag',

    default_args=default_args,

    description='My first DAG',

    schedule_interval=timedelta(days=1),

)
```

Task 4: Create the download task.

```
# define the tasks


# define the task 'download'


download = BashOperator(

    task_id='download',

    bash_command='wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Apache%20Airflow/Build%20a%20DAG%20using%20Airflow/web-server-access-log.txt"',

    dag=dag,

)
```

Task 5: Create the extract task.

The extract task must extract the fields `timestamp` and `visitorid`.

```
# define the task 'extract'


extract = BashOperator(

    task_id='extract',

    bash_command='cut -f1,4 -d"#" web-server-access-log.txt > extracted.txt',

    dag=dag,
```

```
)
```

Task 6: Create the transform task.

The transform task must capitalize the `visitorid`.

```
# define the task 'transform'


transform = BashOperator(
    task_id='transform',
    bash_command='tr "[a-z]" "[A-Z]" < extracted.txt > capitalized.txt',
    dag=dag,
)
```

Task 7: Create the load task.

The `load` task must compress the extracted and transformed data.

```
# define the task 'load'


load = BashOperator(
    task_id='load',
    bash_command='zip log.zip capitalized.txt' ,
    dag=dag,
)
```

Task 8: Create the task pipeline block.

```
# task pipeline


download >> extract >> transform >> load
```

Task 9: Submit the DAG.

```
cp  ETL_Server_Access_Log_Processing.py $AIRFLOW_HOME/dags
```

Task 10: Verify if the DAG is submitted.

```
airflow dags list
```

Verify that the DAG `etl-log-processsing-dag` is listed.

Airflow Monitoring and Logging

Exercise 2 - Submit a dummy DAG

For the purpose of monitoring, let's create a dummy DAG with three tasks.

Task1 does nothing but sleep for 1 second.

Task2 sleeps for 2 seconds.

Task3 sleeps for 3 seconds.

This DAG is scheduled to run every 1 minute.

Step 2.1. Using Menu->`File`->`New File` create a new file named `dummy_dag.py`.

Step 2.2. Copy and paste the code below into it and save the file.

```
# import the libraries


from datetime import timedelta

# The DAG object; we'll need this to instantiate a DAG

from airflow import DAG

# Operators; we need this to write tasks!

from airflow.operators.bash_operator import BashOperator

# This makes scheduling easy

from airflow.utils.dates import days_ago


#defining DAG arguments


# You can override them on a per-task basis during operator initialization

default_args = {
```

```
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG
dag = DAG(
    'dummy_dag',
    default_args=default_args,
    description='My first DAG',
    schedule_interval=timedelta(minutes=1),
)

# define the tasks

# define the first task

task1 = BashOperator(
    task_id='task1',
    bash_command='sleep 1',
    dag=dag,
)
```

```
# define the second task
task2 = BashOperator(
    task_id='task2',
    bash_command='sleep 2',
    dag=dag,
)


# define the third task
task3 = BashOperator(
    task_id='task3',
    bash_command='sleep 3',
    dag=dag,
)


# task pipeline
task1 >> task2 >> task3
```

Submitting a DAG is as simple as copying the DAG python file into `dags`folder in the `AIRFLOW_HOME` directory.

Step 2.3. Open a terminal and run the command below to submit the DAG that was created in the previous exercise.

```
cp dummy_dag.py $AIRFLOW_HOME/dags
```

Step 2.4. Verify that our DAG actually got submitted.

Run the command below to list out all the existing DAGs.

```
airflow dags list
```

Verify that `dummy_dag` is a part of the output.

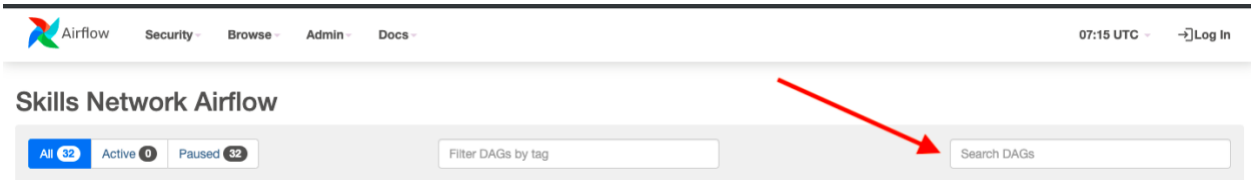
Step 2.5. Run the command below to list out all the tasks in `dummy_dag`.

```
airflow tasks list dummy_dag
```

You should see 3 tasks in the output.

Exercise 3 - Search for a DAG

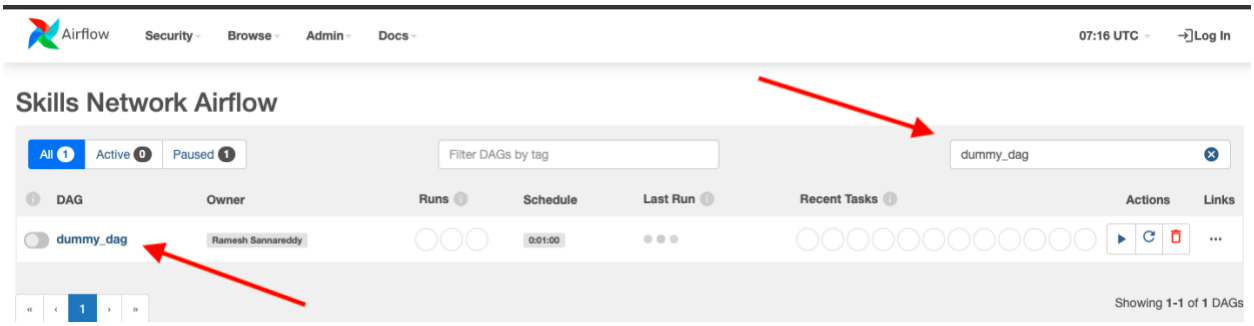
In the Web-UI, identify the `Search DAGs` text box as shown in the image below.



Type `dummy_dag` in the text box and press enter.

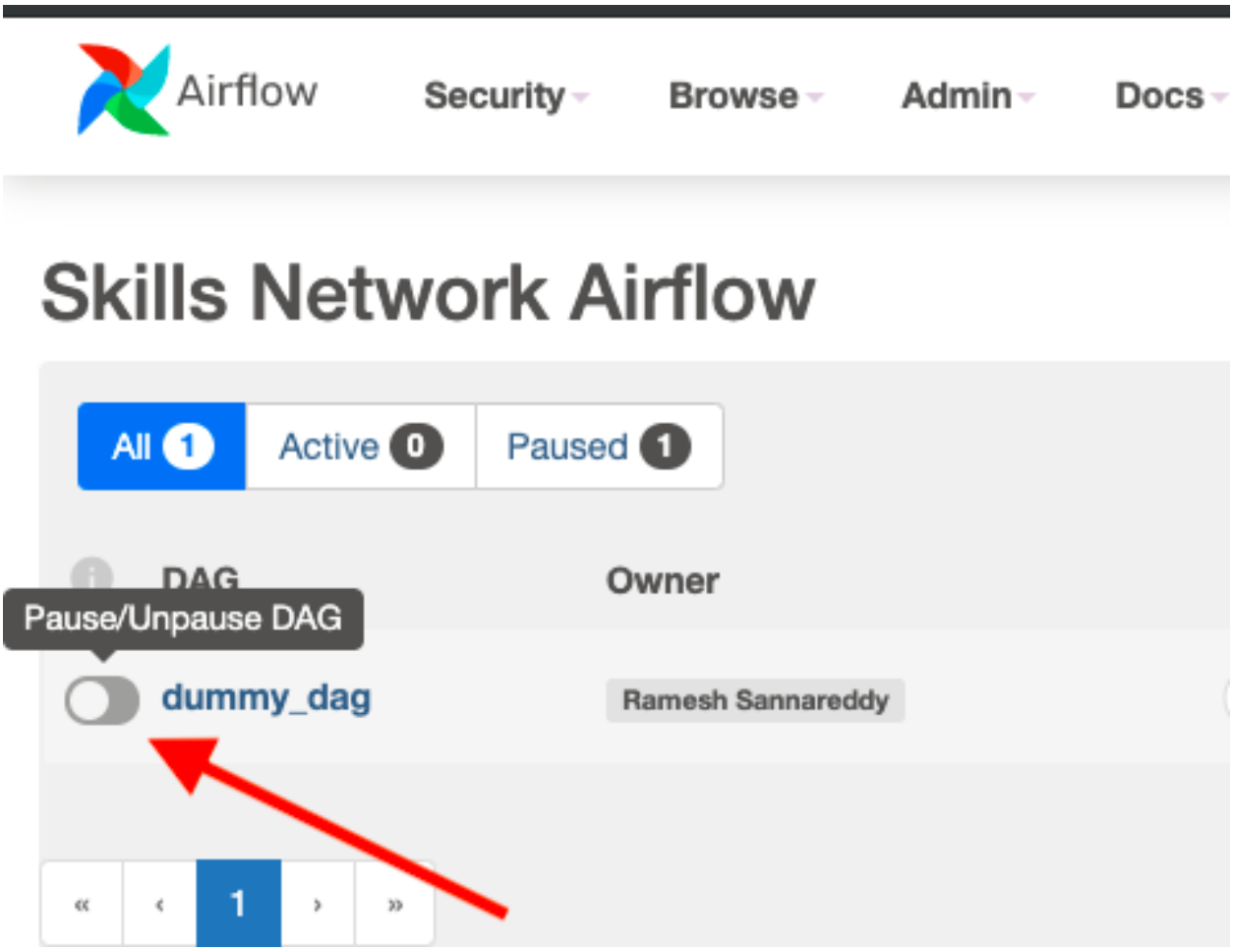
Note: It may take a couple of minutes for the dag to appear here. If you do not see your DAG, please give it a minute and try again.

You should see the `dummy_dag` listed as seen in the image below:

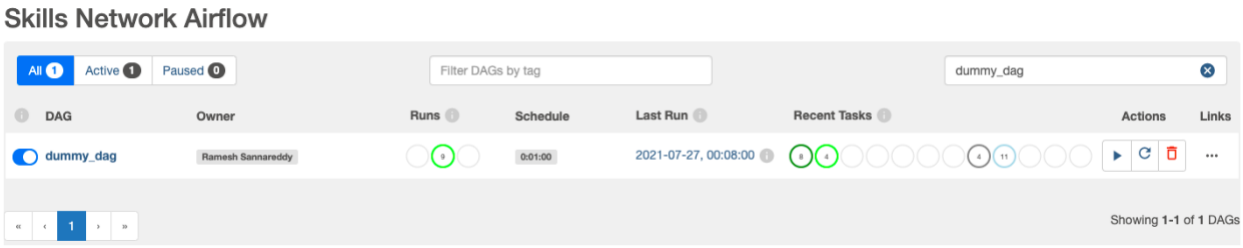


Exercise 4 - Pause/Unpause a DAG

Unpause the DAG using the Pause/Unpause button.



You should see the status as shown in the image below after you unpause the DAG.

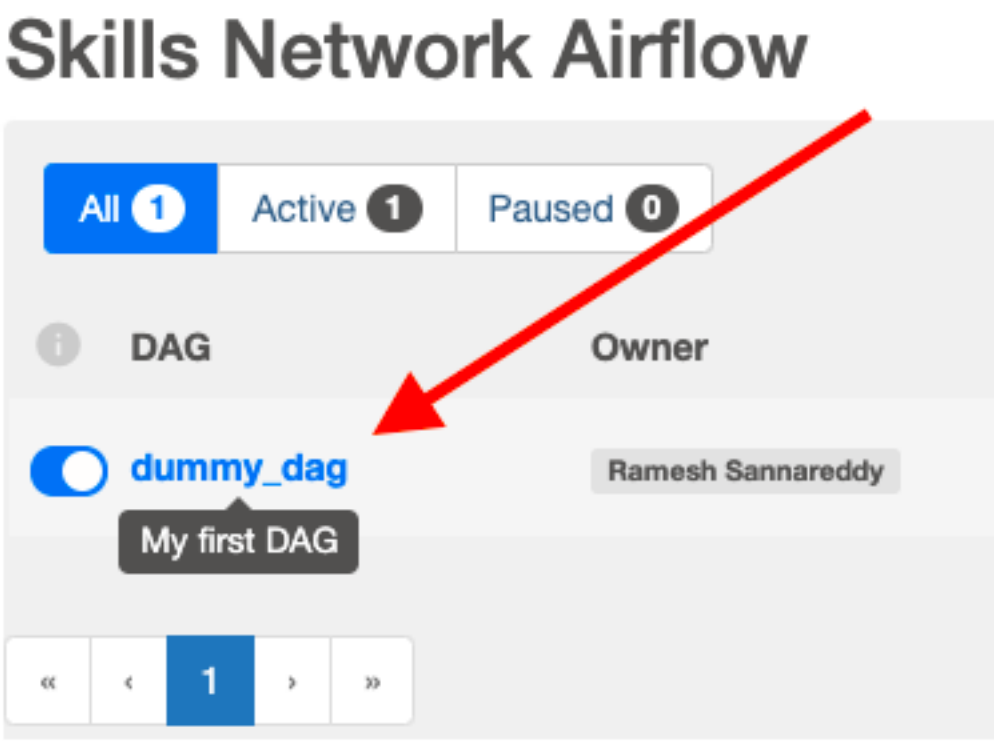


You can see the following details in this view.

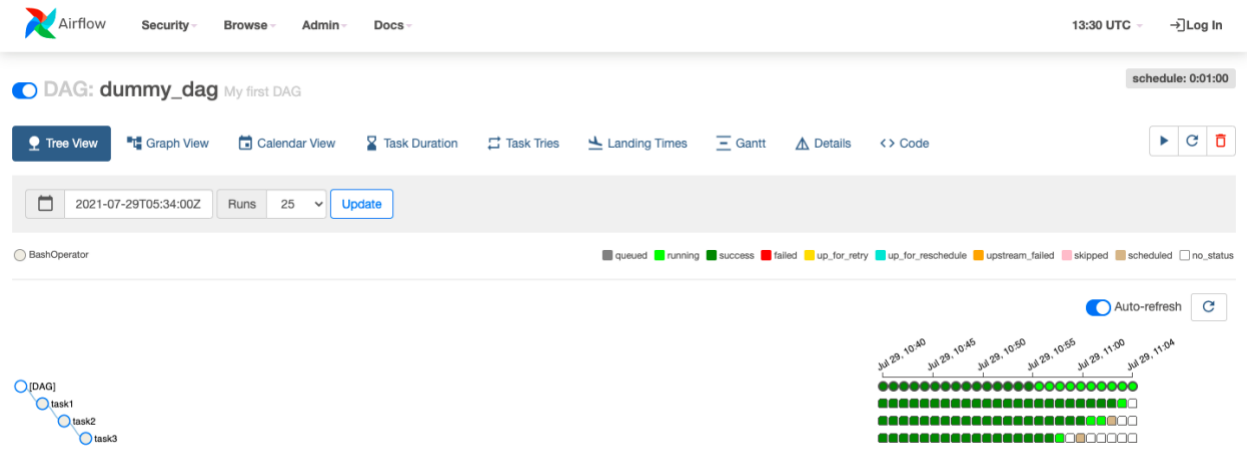
- Owner of the DAG
- How many times this DAG has run.
- Schedule of the DAG
- Last run time of the DAG
- Recent task status.

Exercise 5 - DAG - Detailed view

Click on the DAG name as shown in the image below to see the detailed view of the DAG.

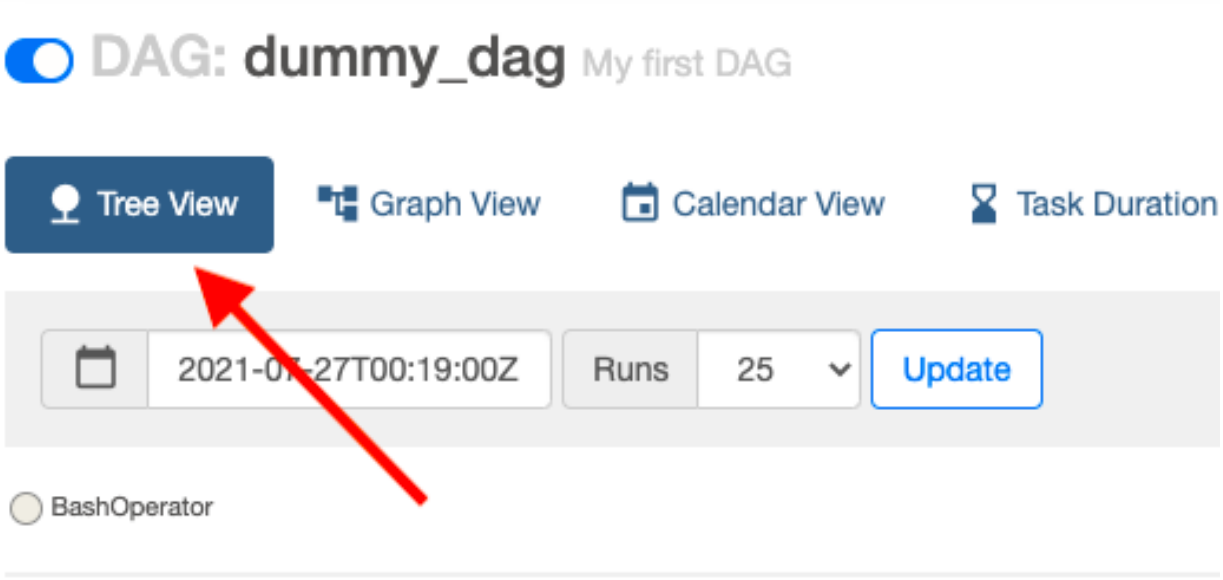


You will land a page that looks like this.



Exercise 6 - Explore tree view of DAG

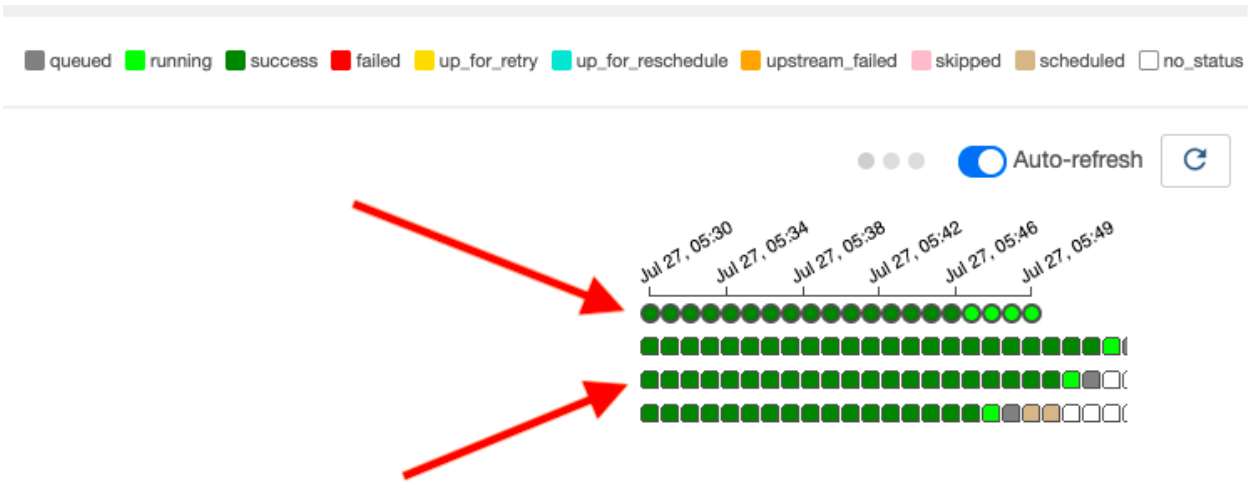
Click on the `Tree View` button to open the Tree view.



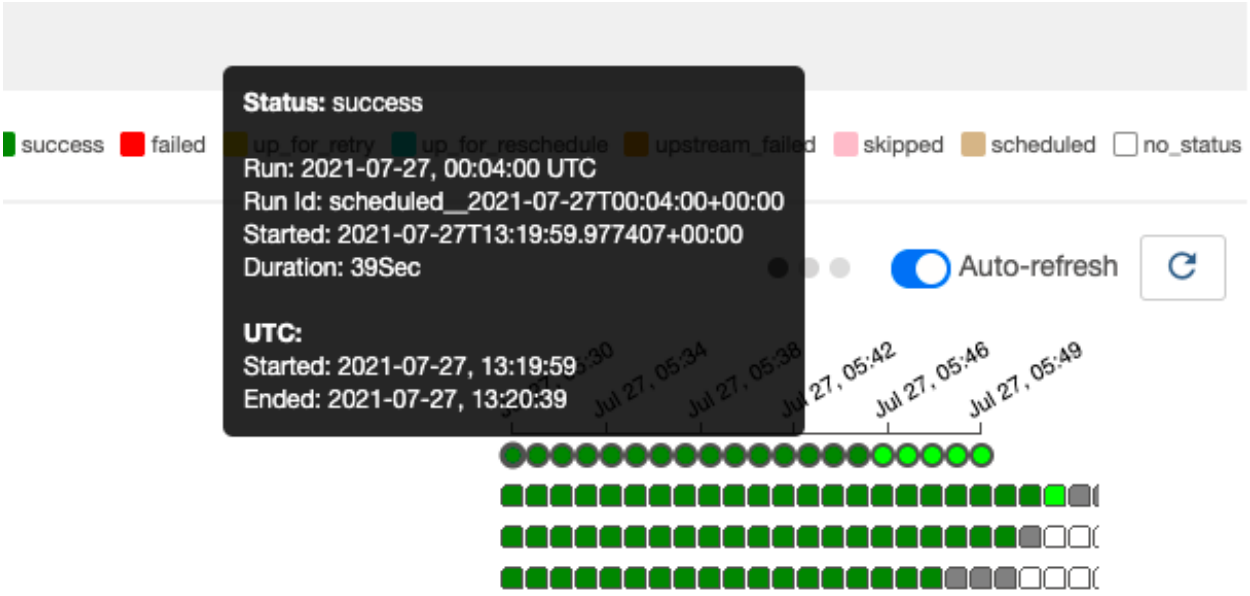
Click on the `Auto Refresh` button to switch on the auto refresh feature.

The tree view shows your DAG tasks in the form of a tree as seen in the image above.

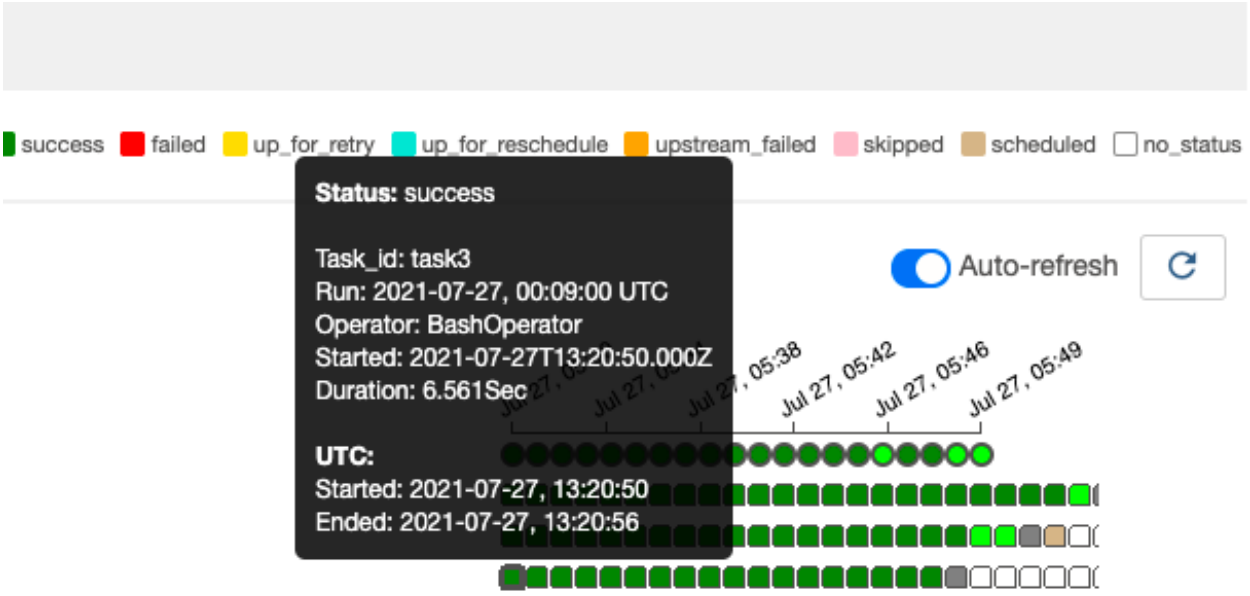
It also shows the DAG run and task run status as seen below.



The circles in the image below represent a single DAG run and the color indicates the status of the DAG run. Place your mouse on any circle to see the details.



The squares in the image below represent a single task within a DAG run and the color indicates its status. Place your mouse on any square to see the task details.



Exercise 7 - Explore graph view of DAG

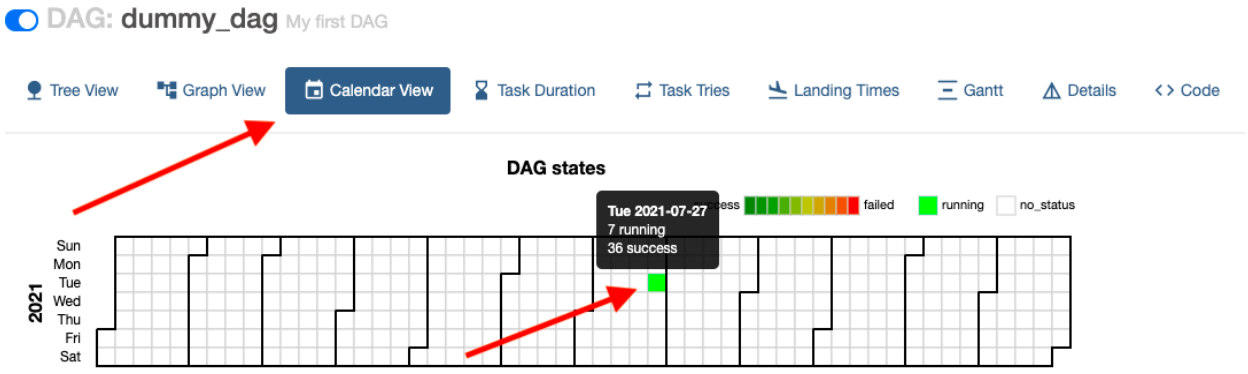
Click on the `Graph View` button to open the graph view.

Click on the `Auto Refresh` button to switch on the auto refresh feature.

The graph view shows the tasks in a form of a graph. With the auto refresh on, each task status is also indicated with the color code.

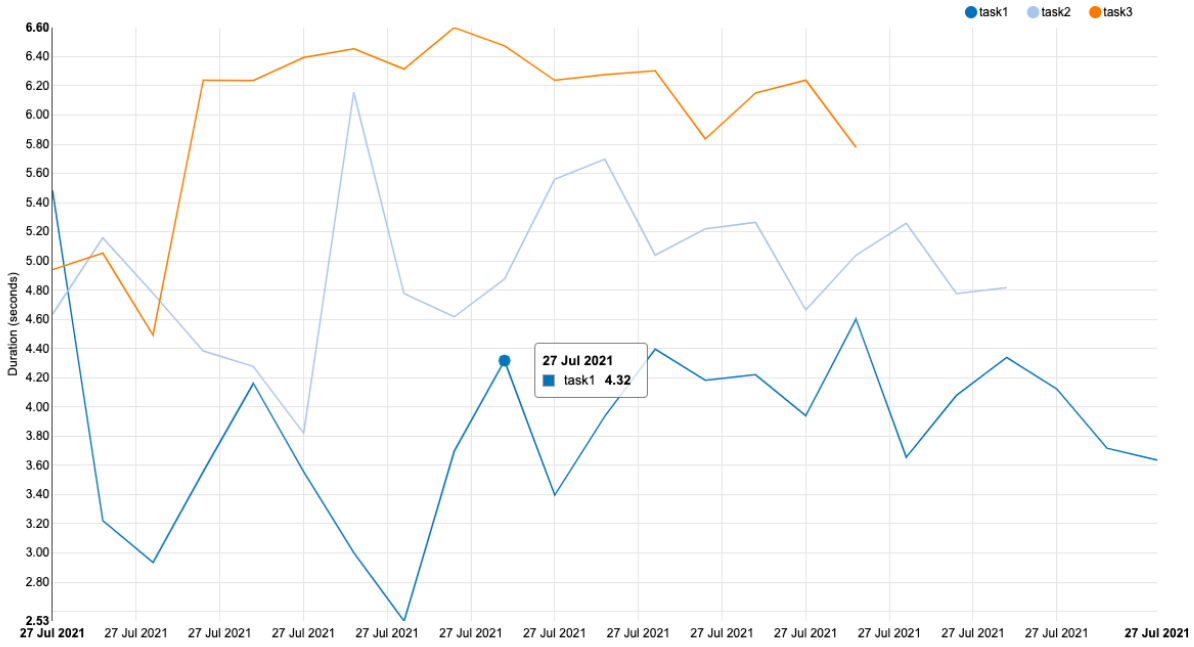
Exercise 8 - Calender view

The calender view gives you an overview of all the dates when this DAG was run along with its status as a color code.



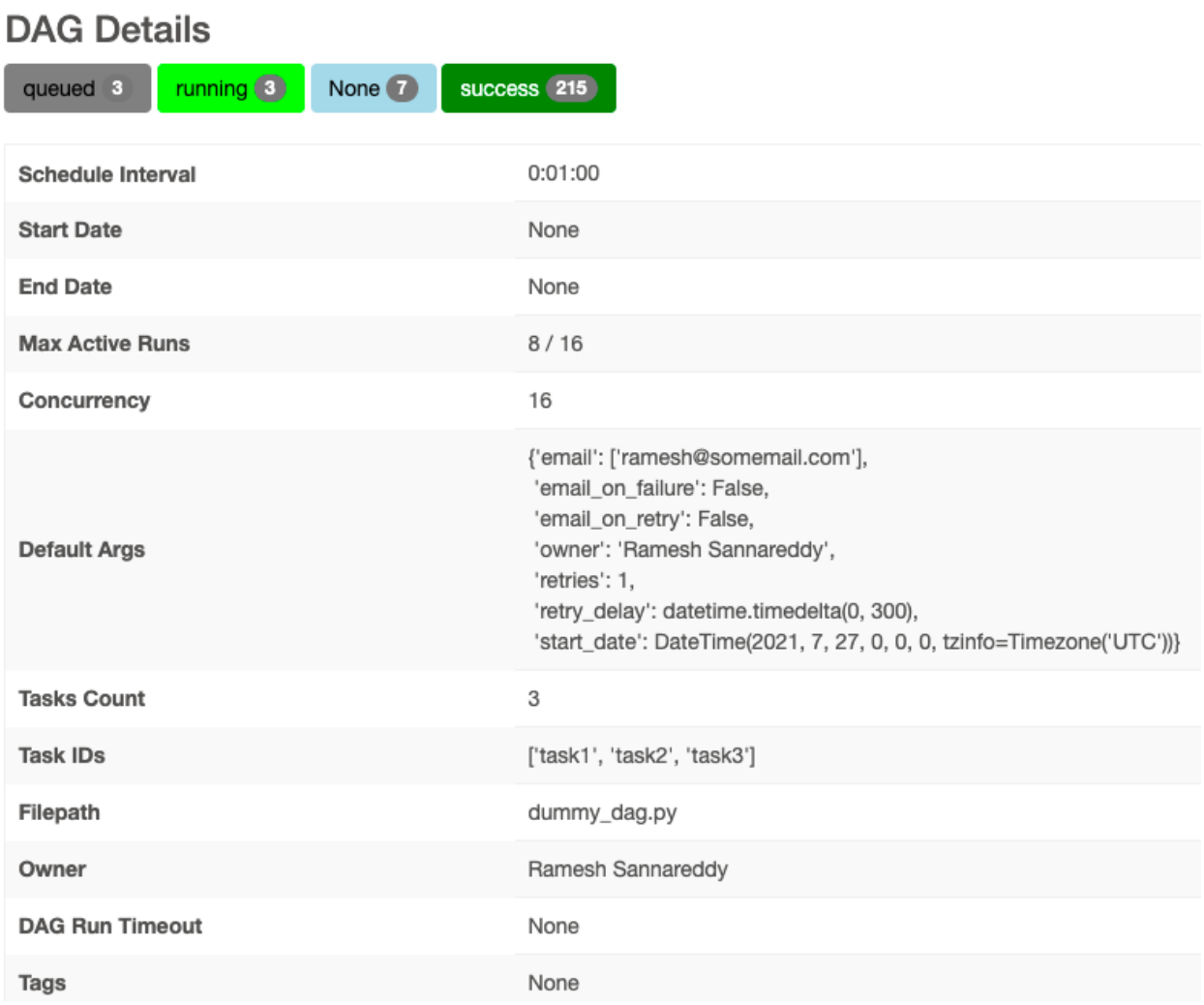
Exercise 9 - Task Duration view

The Task Duration view gives you an overview of how much time each task took to execute, over a period of time.



Exercise 10 - Details view

The Details view give you all the details of the DAG as specified in the code of the DAG.



Exercise 11 - Code view

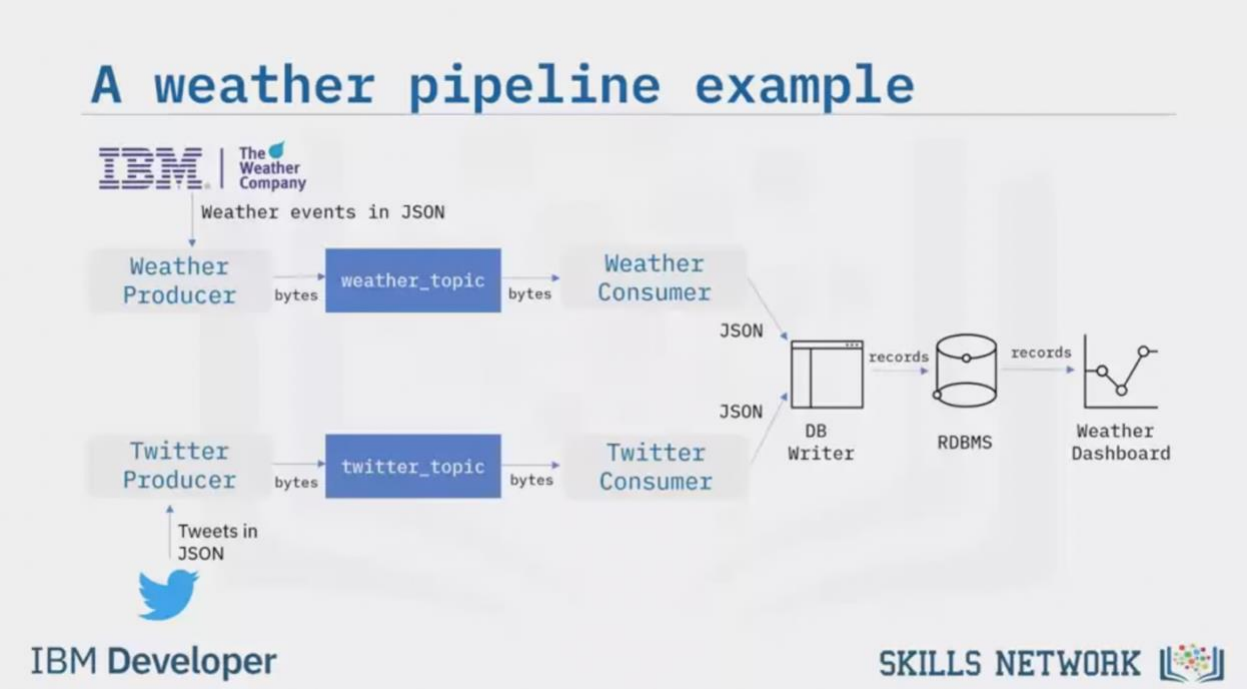
The Code view lets you view the code of the DAG.

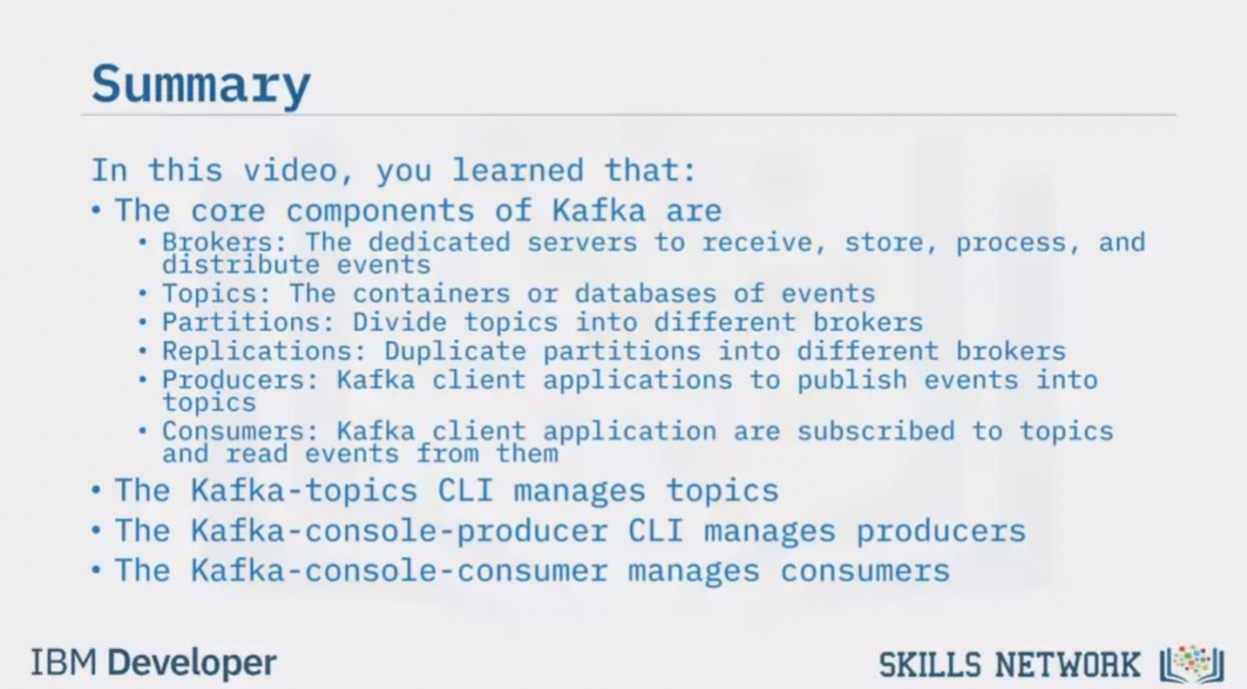
DAG: dummy_dag My first DAG

Tree View | Graph View | Calendar View | Task Duration | Task Tries | Landing Times | Gantt | Details | Code

```python
# import the libraries

from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to write tasks!
from airflow.operators.bash_operator import BashOperator
# This makes scheduling easy
from airflow.utils.dates import days_ago

#defining DAG arguments

# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'Ramesh Sannareddy',
    'start_date': days_ago(0),
    'email': ['ramesh@somemail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# defining the DAG

# define the DAG
dag = DAG(
    'dummy_dag',
```

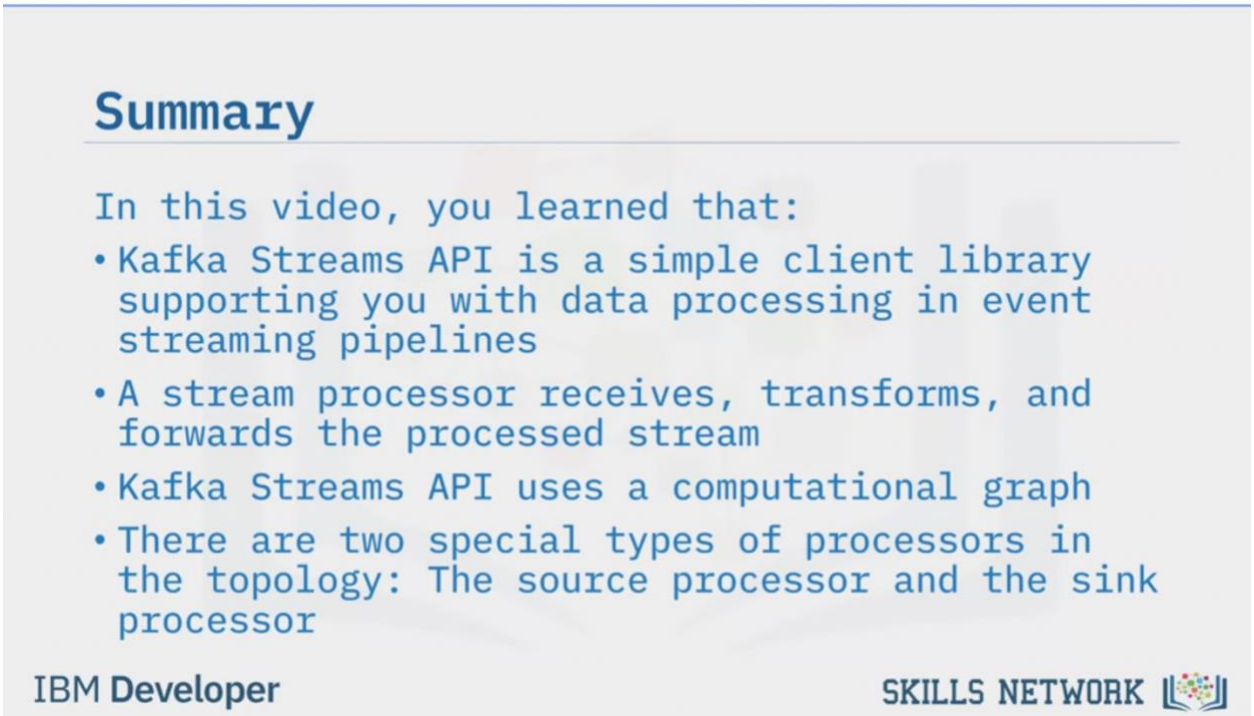
Apache Kafka





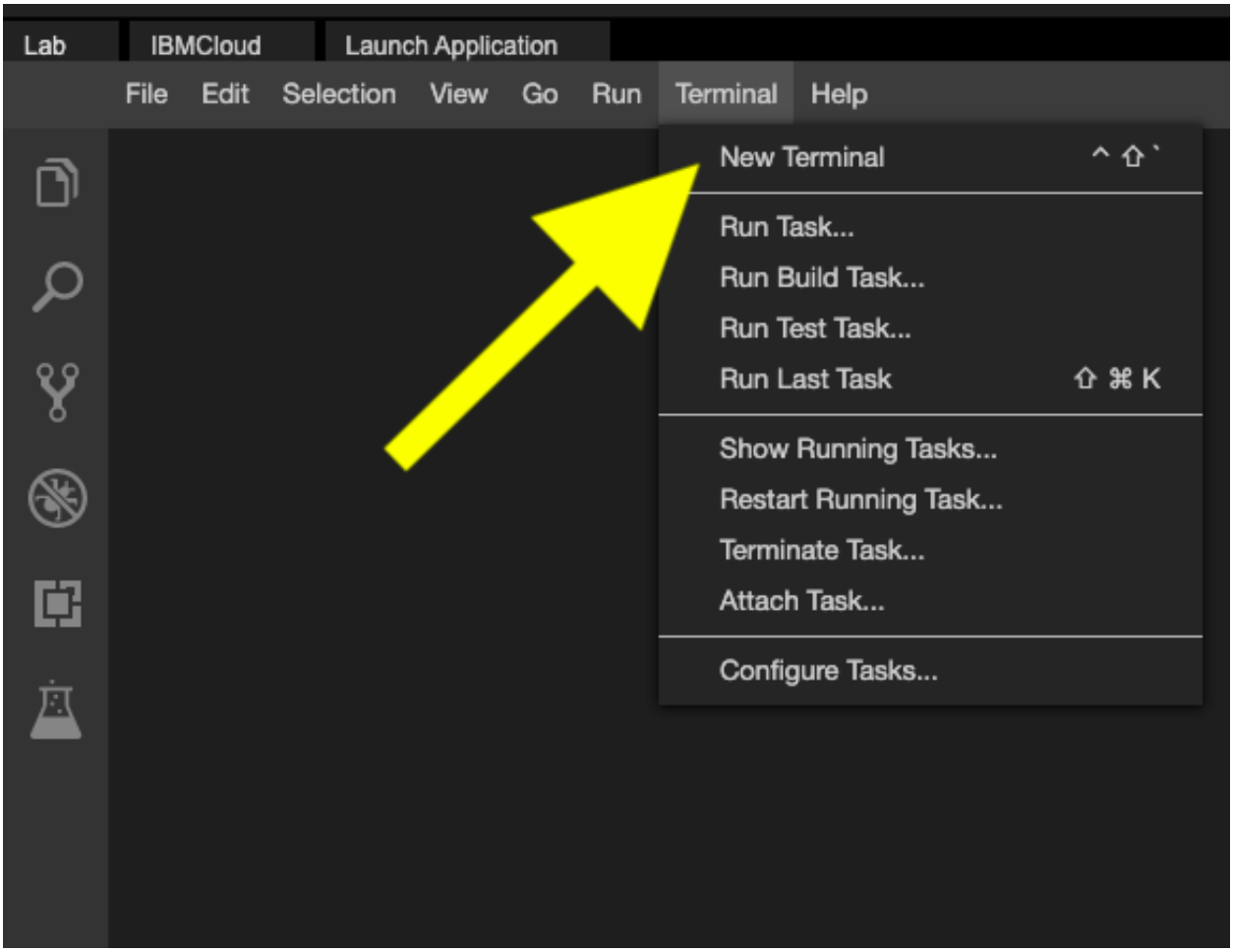
Kafka Streaming

Notes Discuss

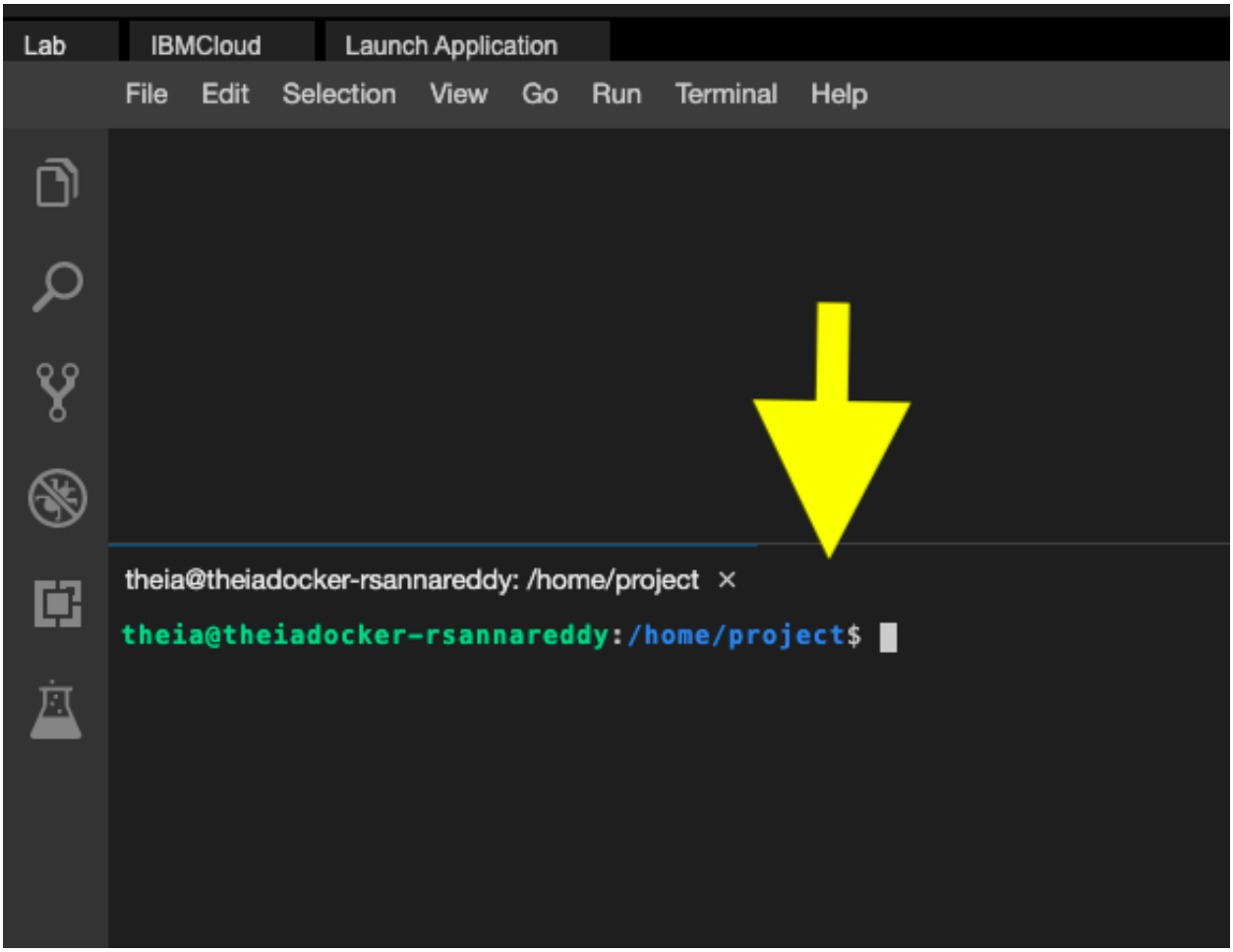


Exercise 1 - Download and extract Kafka

Open a new terminal, by clicking on the menu bar and selecting **Terminal**->**New Terminal**, as shown in the image below.



This will open a new terminal at the bottom of the screen.



Run the commands below on the newly opened terminal. (You can copy the code by clicking on the little copy button on the bottom right of the codeblock below and then paste it, wherever you wish.)

Download Kafka, by running the command below:

```
wget https://archive.apache.org/dist/kafka/2.8.0/kafka_2.12-2.8.0.tgz
```

Extract kafka from the zip file by running the command below.

```
tar -xzf kafka_2.12-2.8.0.tgz
```

This creates a new directory 'kafka_2.12-2.8.0' in the current directory.

Exercise 2 - start ZooKeeper

ZooKeeper is required for Kafka to work. Start the ZooKeeper server.

```
cd kafka_2.12-2.8.0

bin/zookeeper-server-start.sh config/zookeeper.properties
```

mysql --host=127.0.0.1 --port=3306 --user=root --password=Njg4NC1rYXNoeWFw

Exercise 3 - Start the Kafka broker service

Start a new terminal.
Run the commands below. This will start the Kafka message broker service.

```
cd kafka_2.12-2.8.0


bin/kafka-server-start.sh config/server.properties
```

When Kafka starts, you should see an output like this:

Exercise 4 - Create a topic

You need to create a topic before you can start to post messages.

To create a topic named `news`, start a new terminal and run the command below.

```
cd kafka_2.12-2.8.0


bin/kafka-topics.sh --create --topic toll --bootstrap-server localhost:9092
```

You will see the message: 'Created topic news.'

Exercise 5 - Start Producer

You need a producer to send messages to Kafka. Run the command below to start a producer.

```
bin/kafka-console-producer.sh --topic toll --bootstrap-server localhost:9092
```

Once the producer starts, and you get the '>' prompt, type any text message and press enter. Or you can copy the text below and paste. The below text sends three messages to kafka.

```
Good morning

Good day

Enjoy the Kafka lab

Njg4NC1rYXNoeWFw
```

Exercise 6 - Start Consumer

You need a consumer to read messages from kafka.

Open a new terminal.

Run the command below to listen to the messages in the topic `news`.

```
cd kafka_2.12-2.8.0

bin/kafka-console-consumer.sh --topic news --from-beginning --bootstrap-serve
r localhost:9092
```

You should see all the messages you sent from the producer appear here.

You can go back to the producer terminal and type some more messages, one message per line, and you will see them appear here.

Exercise 7 - Explore Kafka directories.

Kafka uses the directory /tmp/kakfa-logs to store the messages.

Explore the folder news-0 inside /tmp/kakfa-logs.

This is where all the messages are stored.

Explore the folder /home/project/kafka_2.12-2.8.0

This folder has the below 3 sub directories.

Practice exercises

1. Problem:

Create a new topic named `weather`.
Click here for Hint
Click here for Solution
Make sure that you are in the 'kafka_2.12-2.8.0' directory. Run the following command:

```
bin/kafka-topics.sh --create --topic weather --bootstrap-server localhost:9092
```

2. Problem:

Post messages to the topic `weather`.
Click here for Hint
Click here for Solution
Make sure that you are in the 'kafka_2.12-2.8.0' directory. Run the following command:

```
bin/kafka-console-producer.sh --topic weather --bootstrap-server localhost:9092
```

Post some test messages.

3. Problem:

Read the messages from the topic `weather`.
Click here for Hint
Click here for Solution

```
bin/kafka-console-consumer.sh --topic weather --from-beginning --bootstrap-server localhost:9092
```

Make sure that the messages you sent from the producer appear here.

Congratulations! You have completed this module. At this point, you know:

- An event stream represents entities' status updates over time
- The main components of an ESP are Event broker, Event storage, Analytic, and Query Engine
- Apache Kafka is a very popular open-source ESP
- Popular Kafka service providers include Confluent Cloud, IBM Event Stream, and Amazon MSK
- The core components of Kafka are brokers, topics, partitions, replications, producers, and consumers
- The Kafka-console-consumer manages consumers
- Kafka Streams API is a simple client library supporting you with data processing in event streaming pipelines
- A stream processor receives, transforms, and forwards the processed stream
- Kafka Streams API uses a computational graph
- There are two special types of processors in the topology: The source processor and the sink processor

Project Overview
Instructions

Now that you are equipped with the knowledge and skills to extract, transform and load data you will use these skills to perform ETL, create a pipeline and upload the data into a database. You will use both Airflow and Kafka in the Hands-on labs.

Scenario

You are a data engineer at a data analytics consulting company. You have been assigned to a project that aims to de-congest the national highways by analyzing the road traffic data from different toll plazas. Each highway is operated by a different toll operator with different IT setup that use different file formats. In the first Hands-on lab your job is to collect data available in different formats and, consolidate it into a single file.

As a vehicle passes a toll plaza, the vehicle's data like `vehicle_id`,`vehicle_type`,`toll_plaza_id` and timestamp are streamed to Kafka. In the second Hands-on lab your job is to create a data pipe line that collects the streaming data and loads it into a database.

Exercise 1 - Prepare the lab environment

Before you start the assignment:

- Start Apache Airflow.
- Download the dataset from the source to the destination mentioned below.

Source : https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignment/tolldata.tgz
Destination : /home/project/airflow/dags/finalassignment

- Create a directory for staging area.

```
mkdir /home/project/airflow/dags/finalassignment/staging
```

Exercise 2 - Create a DAG

Task 1.1 - Define DAG arguments

Define the DAG arguments as per the following details:

Parameter	Value
owner	< You may use any dummy name>
start_date	today
email	< You may use any dummy email>

Parameter	Value
email*on*failure	True
email*on*retry	True
retries	1
retry_delay	5 minutes

Take a screenshot of the task code.

Name the screenshot `dag_args.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.2 - Define the DAG

Create a DAG as per the following details.

Parameter	Value
DAG id	`ETL_toll_data`
Schedule	Daily once
default_args	as you have defined in the previous step
description	Apache Airflow Final Assignment

Take a screenshot of the command you used and the output.

Name the screenshot `dag_definition.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.3 - Create a task to unzip data

Create a task named `unzip_data`.

This task should download data from the url given below and uncompress it into the destination directory.

Take a screenshot of the task code.

Name the screenshot `unzip_data.jpg`. (Images can be saved with either the .jpg or .png extension.)

Read through the file `fileformats.txt` to understand the column details.

Task 1.4 - Create a task to extract data from csv file

Create a task named `extract_data_from_csv`.

This task should extract the fields `Rowid`, `Timestamp`, `Anonymized Vehicle number`, and `Vehicle type` from the `vehicle-data.csv`file and save them into a file named `csv_data.csv`.

Take a screenshot of the task code.

Name the screenshot `extract_data_from_csv.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.5 - Create a task to extract data from tsv file

Create a task named `extract_data_from_tsv`.

This task should extract the fields `Number of axles`, `Tollplaza id`, and `Tollplaza code` from the `tollplaza-data.tsv` file and save it into a file named `tsv_data.csv`.

Take a screenshot of the task code.

Name the screenshot `extract_data_from_tsv.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.6 - Create a task to extract data from fixed width file

Create a task named `extract_data_from_fixed_width`.

This task should extract the fields `Type of Payment code`, and `Vehicle Code` from the fixed width file `payment-data.txt` and save it into a file named `fixed_width_data.csv`.

Take a screenshot of the task code.

Name the screenshot `extract_data_from_fixed_width.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.7 - Create a task to consolidate data extracted from previous tasks

Create a task named `consolidate_data`.

This task should create a single csv file named `extracted_data.csv` by combining data from

- `csv_data.csv`
- `tsv_data.csv`
- `fixed_width_data.csv`

The final csv file should use the fields in the order given below:

`Rowid`, `Timestamp`, `Anonymized Vehicle number`, `Vehicle type`, `Number of axles`, `Tollplaza id`, `Tollplaza code`, `Type of Payment code`, and `Vehicle Code`

Hint: Use the bash `paste` command.

`paste` command merges lines of files.

Example : `paste file1 file2 > newfile`

The above command merges the columns of the files file1 and file2 and sends the output to newfile.

You can use the command `man paste` to explore more.

Take a screenshot of the command you used and the output.

Name the screenshot `consolidate_data.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.8 - Transform and load the data

Create a task named `transform_data`.

This task should transform the vehicle_type field in `extracted_data.csv` into capital letters and save it into a file named `transformed_data.csv` in the staging directory.

Take a screenshot of the command you used and the output.

Name the screenshot `transform.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 1.9 - Define the task pipeline

Define the task pipeline as per the details given below:

Task	Functionality
First task	`unzip_data`
Second task	`extract_data_from_csv`
Third task	`extract_data_from_tsv`
Fourth task	`extract_data_from_fixed_width`
Fifth task	`consolidate_data`
Sixth task	`transform_data`

Take a screenshot of the task pipeline section of the DAG.

Name the screenshot `task_pipeline.jpg`. (Images can be saved with either the .jpg or .png extension.)

```
#defining DAG arguments
default_args = {
    'owner': 'Livanshu Kashyap',
    'start_date': days_ago(0),
```

```
    'email': ['livanshu@gmail.com'],
    'email_on_failure': True,
    'email_on_retry': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# define the DAG
dag = DAG(
    'ETL_toll_data',
    default_args=default_args,
    description='Apache Airflow Final Assignment',
    schedule_interval=timedelta(days=1),
)

#define the task unzip
unzip_data = BashOperator(
    task_id='unzip_data',
    bash_command='wget -c https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-
SkillsNetwork/labs/Final%20Assignment/tolldata.tgz -O -| tar -xz',
    dag=dag,
)

#define the task extract from csv
extract_data_from_csv = BashOperator(
    task_id='extract_data_from_csv',
    bash_command='cut -f 1-4 -d"," vehicle-data.csv >> csv_data.csv',
    dag=dag,
)

#define the task extract from tsv
extract_data_from_tsv = BashOperator(
    task_id='extract_data_from_tsv',
    bash_command='cut -f 5-7 tollplaza-data.tsv >> tsv_data.csv',
    dag=dag,
)
```

```
#define the task extract from fixed width
extract_data_fixed_width = BashOperator(
    task_id='extract_data_fixed_width',
    bash_command='cut -b 59- payment-data.txt >> fixed_width_data.csv',
    dag=dag,
)

#define the task consolidate
consolidate_data = BashOperator(
    task_id='consolidate_data',
    bash_command='paste -d"," csv_data.csv tsv_data.csv fixed_width_data.csv >> extracted_data.csv' ,
    dag=dag,
)

#define the task transform
transform_data = BashOperator(
    task_id='transform_data',
    bash_command='tr "[a-z]" "[A-Z]" < extracted_data.csv > transformed_data.csv',
    dag=dag,
)

#task pipeline
unzip_data >> extract_data_from_csv >> extract_data_from_tsv >> extract_data_fixed_width >> consolidate_data >>
transform_data
```

Scenario

You are a data engineer at a data analytics consulting company. You have been assigned to a project that aims to de-congest the national highways by analyzing the road traffic data from different toll plazas. As a vehicle passes a toll plaza, the vehicle's data like `vehicle_id`,`vehicle_type`,`toll_plaza_id` and timestamp are streamed to Kafka. Your job is to create a data pipe line that collects the streaming data and loads it into a database.

Objectives

In this assignment you will create a streaming data pipe by performing these steps:

- Start a MySQL Database server.
- Create a table to hold the toll data.
- Start the Kafka server.
- Install the Kafka python driver.
- Install the MySQL python driver.
- Create a topic named toll in kafka.
- Download streaming data generator program.
- Customize the generator program to steam to toll topic.
- Download and customise streaming data consumer.
- Customize the consumer program to write into a MySQL database table.
- Verify that streamed data is being collected in the database table.

Exercise 1 - Prepare the lab environment

Before you start the assignment, complete the following steps to set up the lab:

- Step 1: Download Kafka.

```
wget https://archive.apache.org/dist/kafka/2.8.0/kafka_2.12-2.8.0.tgz
```

- Step 2: Extract Kafka.

```
tar -xzf kafka_2.12-2.8.0.tgz
```

- Step 3: Start MySQL server.

```
start_mysql
```

- Step 4: Connect to the mysql server, using the command below. Make sure you use the password given to you when the MySQL server starts. Please make a note or record of the password because you will need it later.

```
mysql --host=127.0.0.1 --port=3306 --user=root --password=Mjk0NDQtcnNhbm5h
```

- Step 5: Create a database named `tolldata`.

At the 'mysql>' prompt, run the command below to create the database.

```
create database tolldata;
```

- Step 6: Create a table named `livetolldata` with the schema to store the data generated by the traffic simulator.

Run the following command to create the table:

```
use tolldata;



create table livetolldata(timestamp datetime,vehicle_id int,vehicle_type char
(15),toll_plaza_id smallint);
```

This is the table where you would store all the streamed data that comes from kafka. Each row is a record of when a vehicle has passed through a certain toll plaza along with its type and anonymized id.

- Step 7: Disconnect from MySQL server.

```
exit
```

- Step 8: Install the python module `kafka-python` using the pip3 command.

```
pip3 install kafka-python
```

This python module will help you to communicate with kafka server. It can used to send and receive messages from kafka.

- Step 8: Install the python module `mysql-connector-python` using the pip3 command.

```
pip3 install mysql-connector-python
```

This python module will help you to interact with mysql server.

Exercise 2 - Start Kafka

Task 2.1 - Start Zookeeper

Start zookeeper server.

Take a screenshot of the command you run.

Name the screenshot `start_zookeeper.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.2 - Start Kafka server

Start Kafka server

Take a screenshot of the command you run.

Name the screenshot `start_kafka.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.3 - Create a topic named `toll`

Create a Kakfa topic named `toll`

Take a screenshot of the command you run.

Name the screenshot `create_toll_topic.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.4 - Download the Toll Traffic Simulator

Download the `toll_traffic_generator.py` from the url given below using 'wget'.

```
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignment/toll_traffic_generator.py
```

Open the code using the theia editor using the "Menu --> File -->Open" option.

Take a screenshot of the task code.

Name the screenshot `download_simulator.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.5 - Configure the Toll Traffic Simulator

Open the `toll_traffic_generator.py` and set the topic to `toll`.

Take a screenshot of the task code with the topic clearly visible.

Name the screenshot `configure_simulator.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.6 - Run the Toll Traffic Simulator

Run the `toll_traffic_generator.py`.

Hint : `python3 <pythonfilename>` runs a python program on the theia lab.

Take a screenshot of the output of the simulator.

Name the screenshot `simulator_output.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.7 - Configure `streaming_data_reader.py`

Download the `streaming_data_reader.py` from the url below using 'wget'.

```
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignment/streaming_data_reader.py
```

Open the `streaming_data_reader.py` and modify the following details so that the program can connect to your mysql server.

`TOPIC`

`DATABASE`

`USERNAME`

`PASSWORD`

Take a screenshot of the code you modified.

Name the screenshot `streaming_reader_code.jpg`. (Images can be saved with either the .jpg or .png extension.)

Task 2.8 - Run `streaming_data_reader.py`

Run the `streaming_data_reader.py`

Take a screenshot of the output of the streaming*data*reader.py.

Name the screenshot `data_reader_output.jpg`. (Images can be saved with either the .jpg or .png extension.)

```
python3 streaming_data_reader.py
```

Task 2.9 - Health check of the streaming data pipeline.

If you have done all the steps till here correctly, the streaming toll data would get stored in the table `livetolldata`.

List the top 10 rows in the table `livetolldata`.

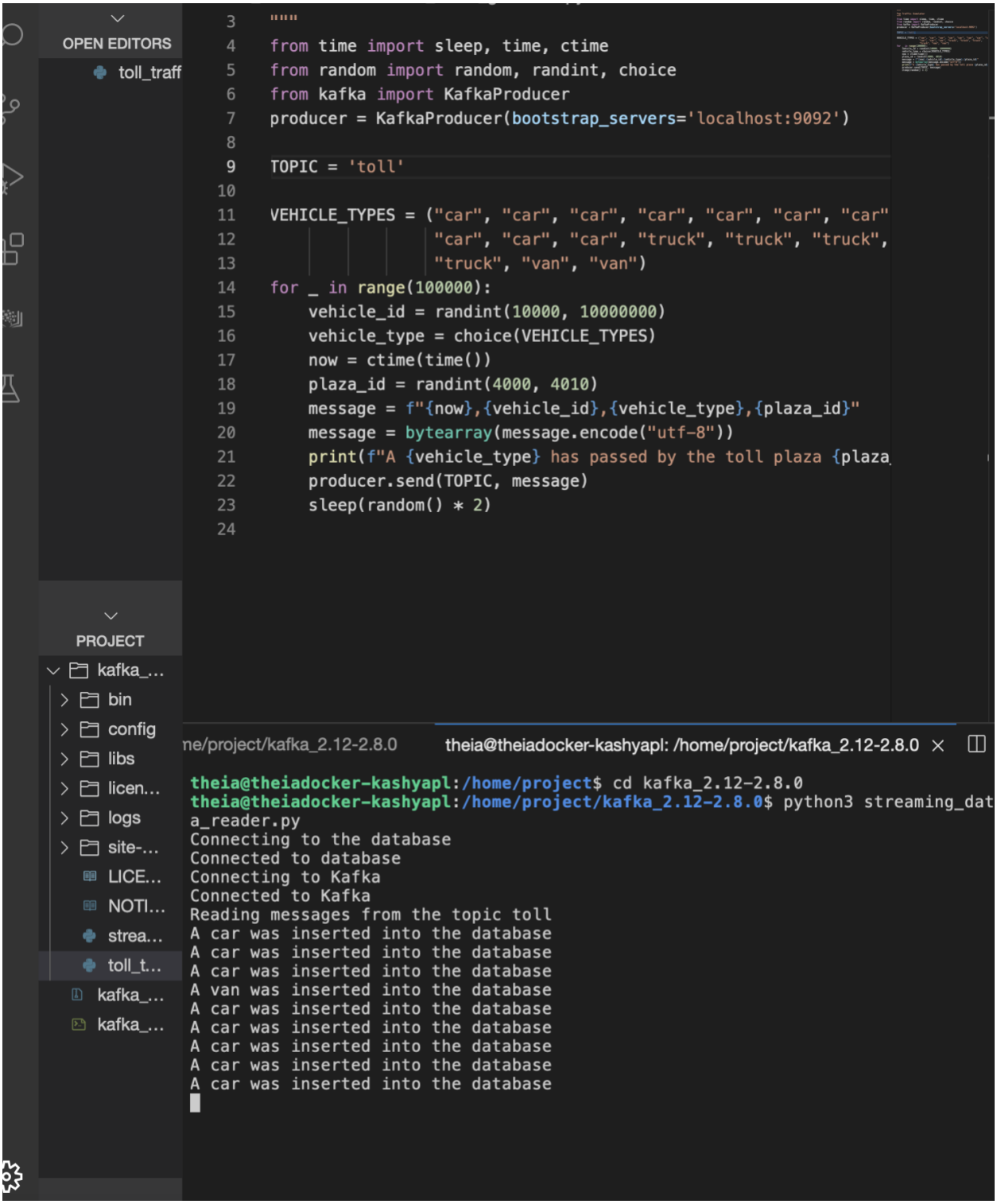
Take a screenshot of the command and the output.

Name the screenshot `output_rows.jpg`. (Images can be saved with either the .jpg or .png extension.)

kafka_2.12-2.8.0 > toll_traffic_generator.py > ...

```python
"""
Top Traffic Simulator
"""
from time import sleep, time, ctime
from random import random, randint, choice
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:9092')

TOPIC = 'toll'

VEHICLE_TYPES = ("car", "car", "car", "car", "car", "car", "car", "car",
                 "car", "car", "car", "truck", "truck", "truck",
                 "truck", "van", "van")
for _ in range(100000):
    vehicle_id = randint(10000, 10000000)
    vehicle_type = choice(VEHICLE_TYPES)
    now = ctime(time())
    plaza_id = randint(4000, 4010)
    message = f"{now},{vehicle_id},{vehicle_type},{plaza_id}"
    message = bytearray(message.encode("utf-8"))
    print(f"A {vehicle_type} has passed by the toll plaza {plaza_id} at {now}.")
    producer.send(TOPIC, message)
    sleep(random() * 2)

```

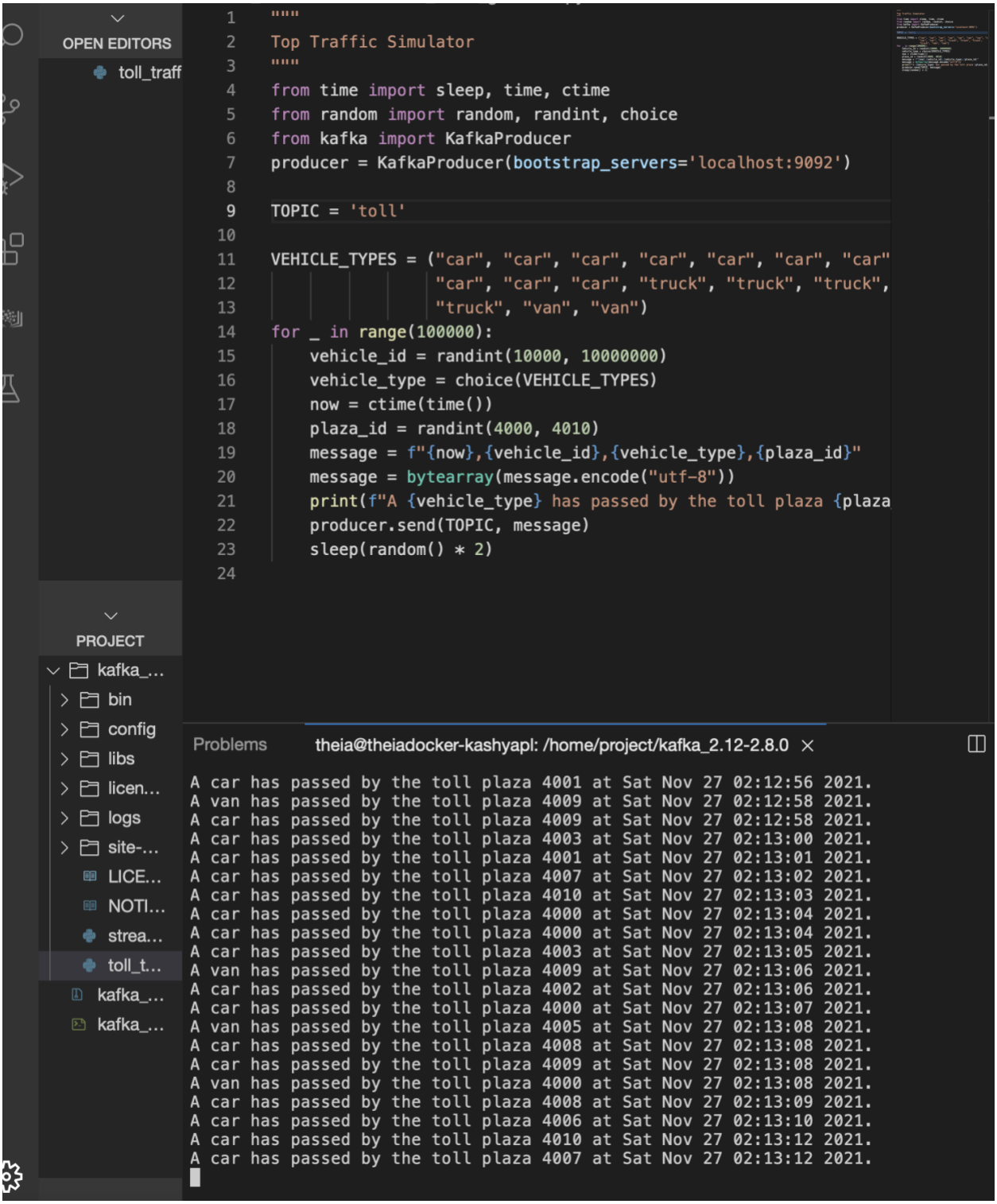
```
theia@theiadocker-kashyapl:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ bin/kafka-topics.sh --create --topic to
ll --bootstrap-server localhost:9092
[2021-11-27 02:49:31,744] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
st/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.client
s.NetworkClient)
[2021-11-27 02:49:31,853] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
st/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.client
s.NetworkClient)
[2021-11-27 02:49:31,955] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
st/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.client
s.NetworkClient)
[2021-11-27 02:49:32,158] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
st/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.client
s.NetworkClient)
[2021-11-27 02:49:32,567] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
st/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.client
s.NetworkClient)
[2021-11-27 02:49:33,383] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localho
```


OPEN EDITORS
toll_traff
PROJECT
kafka_...
bin
config
libs
licen...
logs
site-...
LICE...
NOTI...
strea...
toll_t...
kafka_...
kafka_...
"""
from time import sleep, time, ctime
from random import random, randint, choice
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
TOPIC = 'toll'
VEHICLE_TYPES = ("car", "car", "car", "car", "car", "car", "car"
"car", "car", "car", "truck", "truck", "truck",
"truck", "van", "van")
for _ in range(100000):
vehicle_id = randint(10000, 10000000)
vehicle_type = choice(VEHICLE_TYPES)
now = ctime(time())
plaza_id = randint(4000, 4010)
message = f"{now},{vehicle_id},{vehicle_type},{plaza_id}"
message = bytearray(message.encode("utf-8"))
print(f"A {vehicle_type} has passed by the toll plaza {plaza
producer.send(TOPIC, message)
sleep(random() * 2)
ne/project/kafka_2.12-2.8.0
theia@theiadocker-kashyapl: /home/project/kafka_2.12-2.8.0
theia@theiadocker-kashyapl:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ python3 streaming_dat
a_reader.py
Connecting to the database
Connected to database
Connecting to Kafka
Connected to Kafka
Reading messages from the topic toll
A car was inserted into the database
A car was inserted into the database
A car was inserted into the database
A van was inserted into the database
A car was inserted into the database
A car was inserted into the database
A car was inserted into the database
A car was inserted into the database
A car was inserted into the database

```
431743 after 1 attempt(s)
Caused by: org.apache.kafka.common.errors.TimeoutException: Timed out waiting for a node assignment. Call: createTopics
 (kafka.admin.TopicCommand$)
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignment/toll_traffic_generator.py
--2021-11-27 02:57:46--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignment/toll_traffic_generator.py
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 828 [text/x-python]
Saving to: 'toll_traffic_generator.py'

toll_traffic_generator.p 100%[==============================>]     828  --.-KB/s    in 0s

2021-11-27 02:57:46 (37.4 MB/s) - 'toll_traffic_generator.py' saved [828/828]


Database changed
mysql> SELECT * FROM livetolldata LIMIT 10;
+---------------------+------------+--------------+----------------+
| timestamp           | vehicle_id | vehicle_type | toll_plaza_id |
+---------------------+------------+--------------+----------------+
| 2021-11-27 02:12:52 |     869033 | truck        |          4001 |
| 2021-11-27 02:12:53 |    6521387 | truck        |          4005 |
| 2021-11-27 02:12:55 |     855241 | truck        |          4006 |
| 2021-11-27 02:12:55 |     529782 | truck        |          4005 |
| 2021-11-27 02:12:56 |    3141080 | car          |          4001 |
| 2021-11-27 02:12:58 |    1576035 | van          |          4009 |
| 2021-11-27 02:12:58 |    9234201 | car          |          4009 |
| 2021-11-27 02:13:00 |    5144736 | car          |          4003 |
| 2021-11-27 02:13:01 |     122706 | car          |          4001 |
| 2021-11-27 02:13:02 |    8111727 | car          |          4007 |
+---------------------+------------+--------------+----------------+
10 rows in set (0.03 sec)

mysql>
```

```python
"""
Streaming data consumer
"""
from datetime import datetime
from kafka import KafkaConsumer
import mysql.connector

TOPIC='toll'
DATABASE = 'tolldata'
USERNAME = 'root'
PASSWORD = 'Njg4NC1rYXNoeWFw'

print("Connecting to the database")
try:
    connection = mysql.connector.connect(host='localhost', database=DATABASE, user
except Exception:
    print("Could not connect to database. Please check credentials")
else:
    print("Connected to database")
cursor = connection.cursor()

print("Connecting to Kafka")
consumer = KafkaConsumer(TOPIC)
print("Connected to Kafka")
print(f"Reading messages from the topic {TOPIC}")
for msg in consumer:

    # Extract information from kafka

    message = msg.value.decode("utf-8")

    # Transform the data format to suit the database schema
```

```
heiadocker-kashyapl: /home/project/kafka_2.12-2.8.0    theia@theiadocker-kashyapl: /home/project/kafka_2.12-2.8.0 ×

Saving to: 'streaming_data_reader.py'

streaming_data_reader.py 100%[=================================>]   1.33K  --.-KB/s    in 0s

2021-11-27 01:13:16 (49.4 MB/s) - 'streaming_data_reader.py' saved [1364/1364]

theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ start_mysql
Starting your MySQL database....
This process can take up to a minute.

MySQL database started, waiting for all services to be ready....

Your MySQL database is now ready to use and available with username: root password: Njg4NC1rYXNoeW
w

You can access your MySQL database via:
 • The browser at: https://kashyapl-8080.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitive
ass.ai
 • CommandLine: mysql --host=127.0.0.1 --port=3306 --user=root --password=Njg4NC1rYXNoeWFw
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$
```


OPEN EDITORS
toll_traff
PROJECT
kafka_...
bin
config
libs
licen...
logs
site-...
LICE...
NOTI...
strea...
toll_t...
kafka_...
kafka_...
"""
Top Traffic Simulator
"""
from time import sleep, time, ctime
from random import random, randint, choice
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
TOPIC = 'toll'
VEHICLE_TYPES = ("car", "car", "car", "car", "car", "car", "car"
"car", "car", "car", "truck", "truck", "truck",
"truck", "van", "van")
for _ in range(100000):
vehicle_id = randint(10000, 10000000)
vehicle_type = choice(VEHICLE_TYPES)
now = ctime(time())
plaza_id = randint(4000, 4010)
message = f"{now},{vehicle_id},{vehicle_type},{plaza_id}"
message = bytearray(message.encode("utf-8"))
print(f"A {vehicle_type} has passed by the toll plaza {plaza
producer.send(TOPIC, message)
sleep(random() * 2)
Problems
theia@theiadocker-kashyapl: /home/project/kafka_2.12-2.8.0
A car has passed by the toll plaza 4001 at Sat Nov 27 02:12:56 2021.
A van has passed by the toll plaza 4009 at Sat Nov 27 02:12:58 2021.
A car has passed by the toll plaza 4009 at Sat Nov 27 02:12:58 2021.
A car has passed by the toll plaza 4003 at Sat Nov 27 02:13:00 2021.
A car has passed by the toll plaza 4001 at Sat Nov 27 02:13:01 2021.
A car has passed by the toll plaza 4007 at Sat Nov 27 02:13:02 2021.
A car has passed by the toll plaza 4010 at Sat Nov 27 02:13:03 2021.
A car has passed by the toll plaza 4000 at Sat Nov 27 02:13:04 2021.
A car has passed by the toll plaza 4000 at Sat Nov 27 02:13:04 2021.
A car has passed by the toll plaza 4003 at Sat Nov 27 02:13:05 2021.
A van has passed by the toll plaza 4009 at Sat Nov 27 02:13:06 2021.
A car has passed by the toll plaza 4002 at Sat Nov 27 02:13:06 2021.
A car has passed by the toll plaza 4000 at Sat Nov 27 02:13:07 2021.
A van has passed by the toll plaza 4005 at Sat Nov 27 02:13:08 2021.
A car has passed by the toll plaza 4008 at Sat Nov 27 02:13:08 2021.
A car has passed by the toll plaza 4009 at Sat Nov 27 02:13:08 2021.
A van has passed by the toll plaza 4000 at Sat Nov 27 02:13:08 2021.
A car has passed by the toll plaza 4008 at Sat Nov 27 02:13:09 2021.
A car has passed by the toll plaza 4006 at Sat Nov 27 02:13:10 2021.
A car has passed by the toll plaza 4010 at Sat Nov 27 02:13:12 2021.
A car has passed by the toll plaza 4007 at Sat Nov 27 02:13:12 2021.

```
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ bin/kafka-server-start.sh config/server.properties
JVMJ9VM007W Command-line option unrecognised: -Xlog:gc*:file=/home/project/kafka_2.12-2.8.0/bin/../logs/kafkaServer-gc.log:time,tags:filecount=10,filesize=100M
[2021-11-27 00:26:42,370] INFO Registered kafka:type=kafka.Log4jControllerMBean (kafka.utils.Log4jControllerRegistration$)
[2021-11-27 00:26:42,701] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2021-11-27 00:26:42,785] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2021-11-27 00:26:42,790] INFO starting (kafka.server.KafkaServer)
[2021-11-27 00:26:42,790] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2021-11-27 00:26:42,815] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2021-11-27 00:26:42,822] INFO Client environment:zookeeper.version=3.5.9-83df9301aa5c2a5d284a9940177808c01bc35cef, built on 01/06/2021 20:03 GMT (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,822] INFO Client environment:host.name=theiadocker-kashyapl (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,822] INFO Client environment:java.version=11.0.11 (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,822] INFO Client environment:java.vendor=AdoptOpenJDK (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,822] INFO Client environment:java.home=/usr/lib/jvm/java-jdk-11.0.11+9 (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,822] INFO Client environment:java.class.path=/home/theia/dsdriver/tools/clpplus.jar:/home/theia/dsdriver/tools/jline-0.9.93.jar:/home/theia/dsdriver/tools/antlr-3.3-java.jar:/home/theia/dsdriver/java/db2jcc4.jar:/home/theia/dsdriver/java/sqlj4.zip:/usr/lib/jvm/java-jdk-11.0.11+9/lib:/home/project:/home/project/kafka_2.12-2.8.0/bin/../libs/activation-1.1.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/aopalliance-repackaged-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/argparse4j-0.7.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/audience-annotations-0.5.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/commons-cli-1.4.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/commons-lang3-3.8.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-api-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-basic-auth-extension-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-file-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-json-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-mirror-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-mirror-client-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-runtime-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-transforms-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-api-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-locator-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-utils-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-annotations-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-core-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-databind-2.10.5.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-dataformat-csv-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-datatype-jdk8-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-jaxrs-base-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-jaxrs-json-provider-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-jaxb-annotations-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-paranamer-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-scala_2.12-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.activation-api-1.2.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.annotation-api-1.3.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.inject-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.validation-api-2.0.2.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.ws.rs-api-2.1.6.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.xml.bind-api-2.3.2.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javassist-3.27.0-GA.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javax.servlet-api-3.1.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javax.ws.rs-api-2.1.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jaxb-api-2.3.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-client-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-common-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-container-servlet-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-container-servlet-core-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-hk2-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-media-jaxb-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-server-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-client-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-continuation-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-http-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-io-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-security-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-server-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-servlet-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-servlets-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-util-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-util-ajax-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jline-3.12.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jopt-simple-5.0.4.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka_2.12-2.8.0.jar:/home/project/kafka_2.12-2.8.0-sources.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-clients-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-log4j-appender-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-metadata-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-raft-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-shell-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-examples-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-scala_2.12-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-test-utils-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-tools-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/log4j-1.2.17.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/lz4-java-1.7.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/maven-artifact-3.6.3.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/metrics-core-2.2.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-buffer-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-codec-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-common-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-handler-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-resolver-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-transport-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-transport-native-epoll-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/netty-transport-native-unix-common-4.1.62.Final.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/osgi-resource-locator-1.0.3.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/paranamer-2.8.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/plexus-utils-3.2.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/reflections-0.9.12.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/rocksdbjni-5.18.4.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/scala-collection-compat_2.12-2.3.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/scala-java8-compat_2.12-0.9.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/scala-library-2.12.13.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/scala-logging_2.12-3.9.2.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/scala-reflect-2.12.13.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/slf4j-api-1.7.30.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/slf4j-log4j12-1.7.30.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/snappy-java-1.1.8.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/zookeeper-3.5.9.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/zookeeper-jute-3.5.9.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/zstd-jni-1.4.9-1.jar (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,824] INFO Client environment:java.library.path=/usr/lib/jvm/java-jdk-11.0.11+9/lib/default:/usr/lib/jvm/java-jdk-11.0.11+9/lib:/home/theia/dsdriver/lib:/usr/lib:/usr/lib64 (org.apache.zookeeper.ZooKeeper)
[2021-11-27 00:26:42,824] INFO Client environment:java.io.tmpdir=/tmp (org.apache.zookeeper.ZooKeeper)
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ bin/zookeeper-server-start.sh config/zookeeper.properties
JVMJ9VM007W Command-line option unrecognised: -Xlog:gc*:file=/home/project/kafka_2.12-2.8.0/bin/../logs/zookeeper-gc.log:time,tags:filecount=10,filesize=100M
[2021-11-27 00:22:46,330] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,332] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,342] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,342] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,351] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-11-27 00:22:46,351] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-11-27 00:22:46,351] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-11-27 00:22:46,352] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2021-11-27 00:22:46,364] INFO Log4j 1.2 jmx support found and enabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2021-11-27 00:22:46,386] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,386] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,387] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,387] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-11-27 00:22:46,387] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2021-11-27 00:22:46,391] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2021-11-27 00:22:46,403] INFO Server environment:zookeeper.version=3.5.9-83df9301aa5c2a5d284a9940177808c01bc35cef, built on 01/06/2021 20:03 GMT (org.apache.zookeeper.server.ZooKeeperServer)
[2021-11-27 00:22:46,403] INFO Server environment:host.name=theiadocker-kashyapl (org.apache.zookeeper.server.ZooKeeperServer)
[2021-11-27 00:22:46,403] INFO Server environment:java.version=11.0.11 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-11-27 00:22:46,403] INFO Server environment:java.vendor=AdoptOpenJDK (org.apache.zookeeper.server.ZooKeeperServer)
[2021-11-27 00:22:46,403] INFO Server environment:java.home=/usr/lib/jvm/java-jdk-11.0.11+9 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-11-27 00:22:46,403] INFO Server environment:java.class.path=/home/theia/dsdriver/tools/clpplus.jar:/home/theia/dsdriver/tools/jline-0.9.93.jar:/home/theia/dsdriver/tools/antlr-3.3-java.jar:/home/theia/dsdriver/java/db2jcc4.jar:/home/theia/dsdriver/java/sqlj4.zip:/usr/lib/jvm/java-jdk-11.0.11+9/lib:/home/project:/home/project/kafka_2.12-2.8.0/bin/../libs/activation-1.1.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/aopalliance-repackaged-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/argparse4j-0.7.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/audience-annotations-0.5.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/commons-cli-1.4.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/commons-lang3-3.8.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-api-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-basic-auth-extension-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-file-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-json-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-mirror-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-mirror-client-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-runtime-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/connect-transforms-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-api-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-locator-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/hk2-utils-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-annotations-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-core-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-databind-2.10.5.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-dataformat-csv-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-datatype-jdk8-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-jaxrs-base-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-jaxrs-json-provider-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-jaxb-annotations-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-paranamer-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jackson-module-scala_2.12-2.10.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.activation-api-1.2.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.annotation-api-1.3.5.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.inject-2.6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.validation-api-2.0.2.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.ws.rs-api-2.1.6.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jakarta.xml.bind-api-2.3.2.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javassist-3.27.0-GA.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javax.servlet-api-3.1.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/javax.ws.rs-api-2.1.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jaxb-api-2.3.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-client-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-common-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-container-servlet-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-container-servlet-core-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-hk2-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-media-jaxb-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jersey-server-2.31.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-client-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-continuation-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-http-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-io-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-security-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-server-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-servlet-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-servlets-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-util-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jetty-util-ajax-9.4.39.v20210325.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jline-3.12.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/jopt-simple-5.0.4.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka_2.12-2.8.0.jar:/home/project/kafka_2.12-2.8.0-sources.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-clients-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-log4j-appender-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-metadata-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-raft-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-shell-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-2.8.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/kafka-streams-examples-2.8.0.jar:/home/project/kafka_2.12-2.8
```

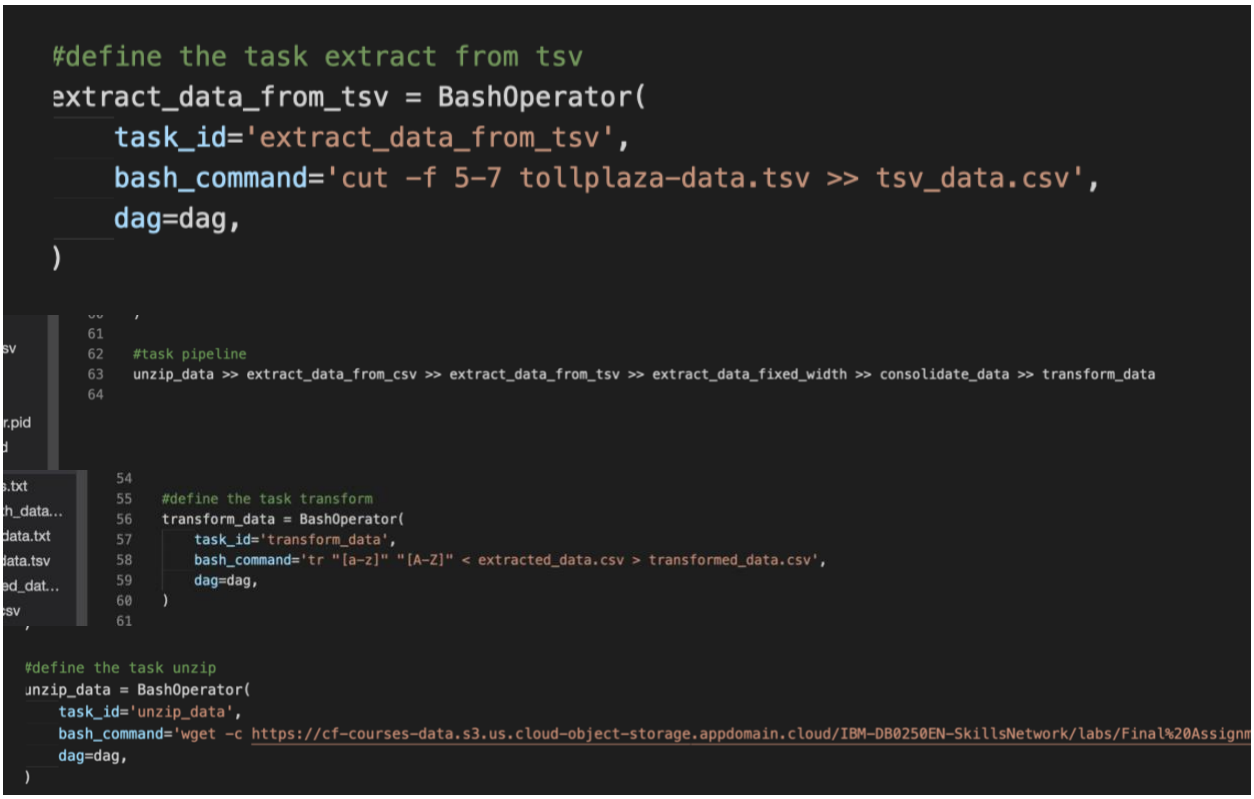
kafka_2.12-2.8.0 > streaming_data_reader.py > ...

```python
"""
Streaming data consumer
"""
from datetime import datetime
from kafka import KafkaConsumer
import mysql.connector

TOPIC='toll'
DATABASE = 'tolldata'
USERNAME = 'root'
PASSWORD = 'Njg4NC1rYXNoeWFw'

print("Connecting to the database")
try:
    connection = mysql.connector.connect(host='localhost', database=DATABASE, user
except Exception:
    print("Could not connect to database. Please check credentials")
else:
    print("Connected to database")
cursor = connection.cursor()

print("Connecting to Kafka")
consumer = KafkaConsumer(TOPIC)
print("Connected to Kafka")
print(f"Reading messages from the topic {TOPIC}")
for msg in consumer:

    # Extract information from kafka

    message = msg.value.decode("utf-8")

```

theia@theiadocker-kashyapl: /home/project/kafka_2.12-2.8.0 × theia@theiadocker-kashyapl: /home/project/kafka_2.12

```
lready exists.
 (kafka.admin.TopicCommand$)
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ python3 toll_traffic_generator.py
A car has passed by the toll plaza 4003 at Sat Nov 27 01:29:57 2021.
Traceback (most recent call last):
  File "toll_traffic_generator.py", line 22, in <module>
    producer.send(TOPIC, message)
  File "/home/theia/.local/lib/python3.6/site-packages/kafka/producer/kafka.py", line 576, in sen
    self._wait_on_metadata(topic, self.config['max_block_ms'] / 1000.0)
  File "/home/theia/.local/lib/python3.6/site-packages/kafka/producer/kafka.py", line 703, in _wa
_on_metadata
    "Failed to update metadata after %.1f secs." % (max_wait,))
kafka.errors.KafkaTimeoutError: KafkaTimeoutError: Failed to update metadata after 60.0 secs.
theia@theiadocker-kashyapl:/home/project/kafka_2.12-2.8.0$ python3 streaming_data_reader.py
Connecting to the database
Connected to database
Connecting to Kafka
Connected to Kafka
Reading messages from the topic toll
```

```
#define the task consolidate
consolidate_data = BashOperator(
    task_id='consolidate_data',
    bash_command='paste -d"," csv_data.csv tsv_data.csv fixed_width_data.csv >> extracted_data.csv' ,
    dag=dag,
)

#defining DAG arguments
default_args = {
    'owner': 'Livanshu Kashyap',
    'start_date': days_ago(0),
    'email': ['livanshu@gmail.com'],
    'email_on_failure': True,
    'email_on_retry': True,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

define the DAG
lag = DAG(
    'ETL_toll_data',
    default_args=default_args,
    description='Apache Airflow Final Assignment',
    schedule_interval=timedelta(days=1),


  #define the task extract from csv
  extract_data_from_csv = BashOperator(
      task_id='extract_data_from_csv',
      bash_command='cut -f 1-4 -d"," vehicle-data.csv >> csv_data.csv',
      dag=dag,
  )

#define the task extract from fixed width
extract_data_fixed_width = BashOperator(
    task_id='extract_data_fixed_width',
    bash_command='cut -b 59- payment-data.txt >> fixed_width_data.csv',
    dag=dag,
)
```



```
#define the task extract from tsv
extract_data_from_tsv = BashOperator(
    task_id='extract_data_from_tsv',
    bash_command='cut -f 5-7 tollplaza-data.tsv >> tsv_data.csv',
    dag=dag,
)

#task pipeline
unzip_data >> extract_data_from_csv >> extract_data_from_tsv >> extract_data_fixed_width >> consolidate_data >> transform_data


#define the task transform
transform_data = BashOperator(
    task_id='transform_data',
    bash_command='tr "[a-z]" "[A-Z]" < extracted_data.csv > transformed_data.csv',
    dag=dag,
)

#define the task unzip
unzip_data = BashOperator(
    task_id='unzip_data',
    bash_command='wget -c https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Final%20Assignm
    dag=dag,
)
```