Cloudera Data Engineering labs

1. Introduction

These labs are based on tutorials published on cloudera.com/tutorials:
Getting Started with Cloudera Data Engineering,
Enrich Data using Cloudera Data Engineering,
Using CLI-API to Automate Access to Cloudera Data Engineering

If you are looking for more exercises please have a look at cloudera.com/tutorials

The labs in this document will give you an introduction to how you can submit Apache Spark jobs using Cloudera Data Engineering (CDE), the serverless auto-scaling service on Cloudera Data Platform (CDP).

The workshop consists of 2 labs.
Lab 1 will create a data warehouse through a Spark job based on CSV files. After creation you will submit an enriching job which will create a new table with enriched data based on information already available in the just created data warehouse.

In Lab 2 you will deploy an Apache Airflow job where you will create a flow based on the scripts used in Lab 1.

If you have set up the Cloudera Data Engineering CLI you can optionally do Lab 3, where you can list, create and start jobs from the CLI from your workstation.

2. Before you begin

You don't need to write your own PySpark scripts. You can download the scripts via this link. Unzip the file.
Ask the instructor for:

- The **Bucket_File_Location** (like s3a://usermarketing-cdp-demo/dl/cars or abfs://data@marketing.dfs.core.windows.net/dl/cars)
- Your **User_Identifier** (like USER01)

You will need to use this information to update the file names of the scripts and the scripts itself to make the created content unique.

First rename the files:

Replace the <Pre-Fix> with your User_Identifier (e.g. USER01-Pre-SetupDW.py) for all 3 files: <Pre-Fix>-Pre-SetupDW.py, <Pre-Fix>-EnrichData_ETL.py and <Pre-Fix>-Airflow-Dag.py.

*During the labs read and write your **User_Identifier** wherever you see **<Pre-Fix>**.*

3. Lab 1: Create and Enrich a Data Warehouse

In this lab you will create a data warehouse with sales, customer and factory data based on CSV files which have been setup in your data lake by your administrator.

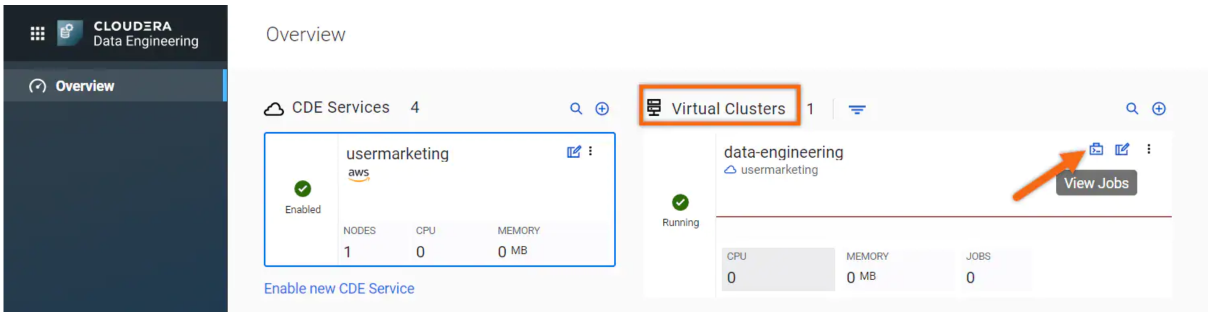
Before we can run the scripts we need to replace some placeholders.
First open the script <Pre-Fix>-SetupDW.py in your favorite editor. You need to find/replace two placeholders with the values you obtained from the instructor.
Replace placeholder <Bucket> (inlcuding brackets) with the Bucket_File_Location obtained in paragraph 2.
Replace placeholder <Pre-Fix> with the User_Identifier obtained in paragraph 2.
Save the script.

Then open the script <Pre-Fix>-EnrichData_ETL.py in your favorite editor. You need to replace the placeholder <Pre-Fix> with the User_Identifier obtained in paragraph 2.
Save the script.

This ensures that you a) are reading the right data and b) are not overwriting tables of other attendees.

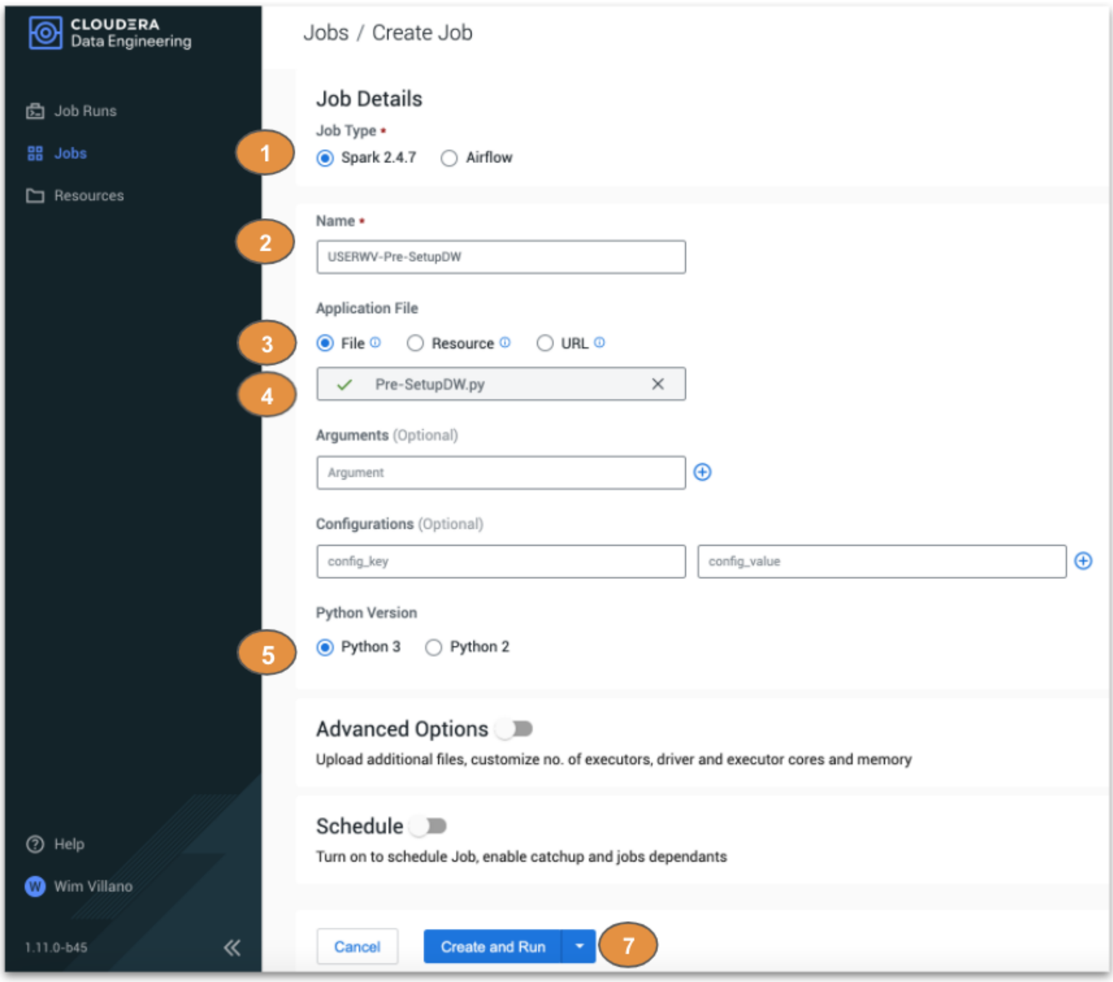
We will be using the GUI to run the job.

With the first job we will create the data warehouse (tables and loading data).
In your assigned virtual cluster, view jobs by selecting .



In the Jobs section, select Create Job to create a new job:

1. Job Type: Spark
2. Name: <Pre-Fix>-Pre-SetupDW
3. Application File: File
4. Upload File: <Pre-Fix>-Pre-SetupDW.py
5. Select Python 3
6. Leave Advanced Options and Scheduling to default settings
7. Hit button Create and Run


Jobs / Create Job

Job Details

Job Type •
[✓ Spark 2.4.7] Airflow

Name •
USERWV-Pre-SetupDW

Application File
[✓ File] Resource URL

Pre-SetupDW.py

Arguments (Optional)

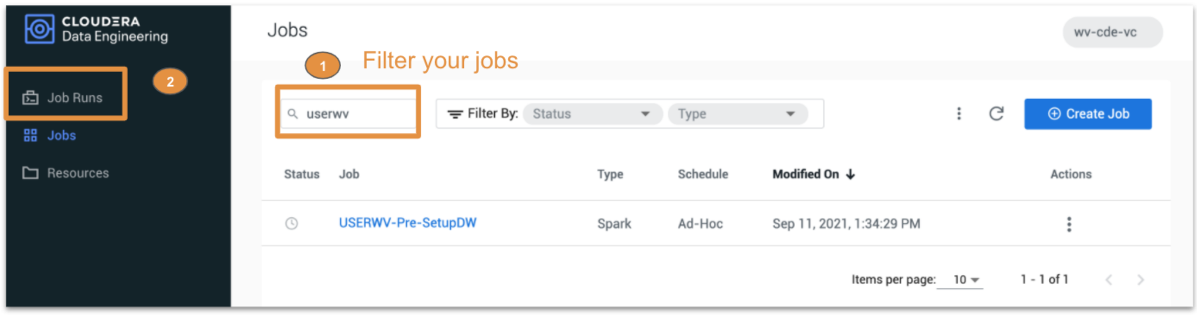
Configurations (Optional)

Python Version
[✓ Python 3] Python 2

Advanced Options
Upload additional files, customize no. of executors, driver and executor cores and memory

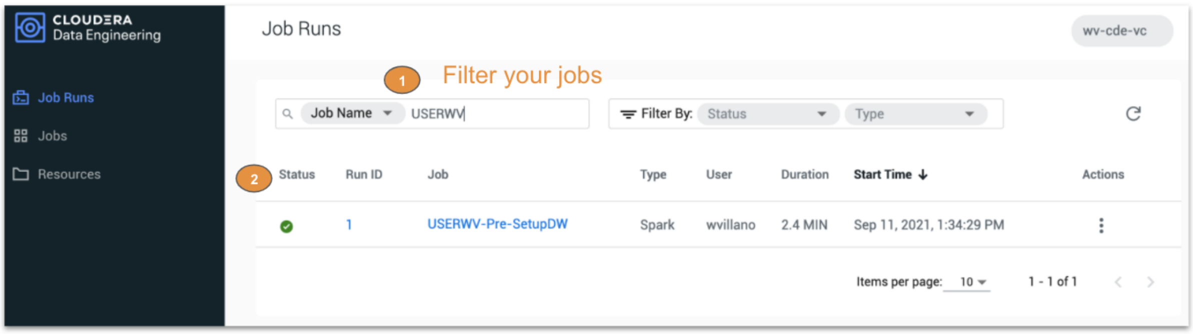
Schedule
Turn on to schedule Job, enable catchup and jobs dependants

Your job appears in the Jobs list. You can filter for your job (1). To see if and how the job is running click on Job Runs (2).


Jobs

Status	Job	Type	Schedule	Modified On	Actions
	USERWV-Pre-SetupDW	Spark	Ad-Hoc	Sep 11, 2021, 1:34:29 PM	

Here you can also filter for your job (1) and check the status. If the job ran successfully a green check will appear (2).



Job Runs

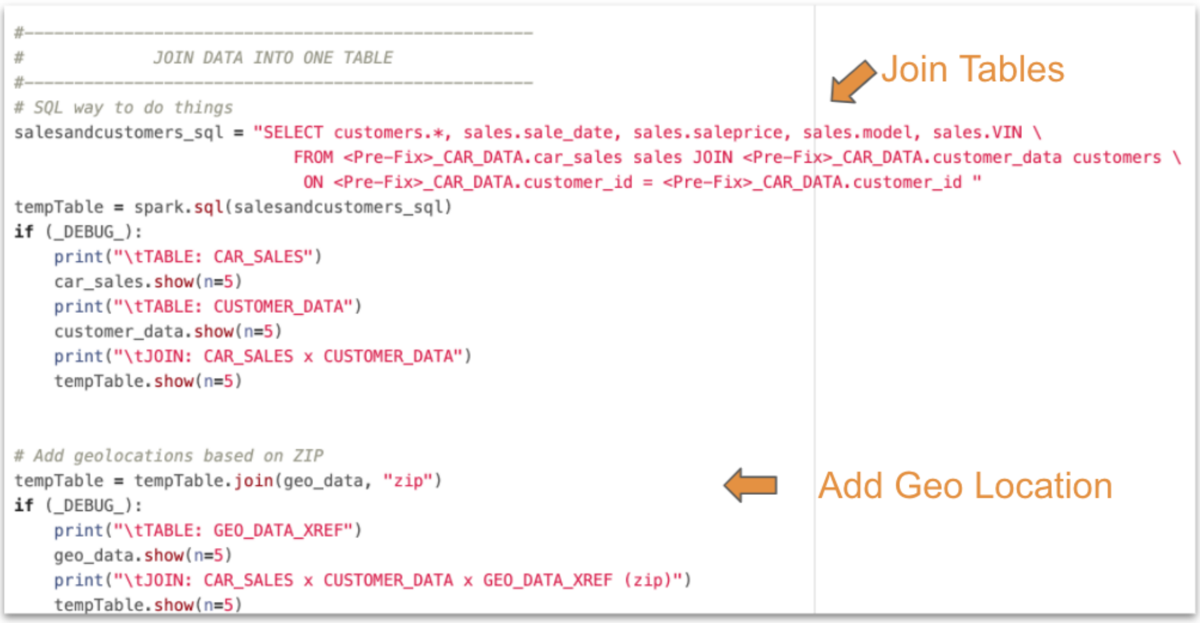
Status	Run ID	Job	Type	User	Duration	Start Time	Actions
✓	1	USERWV-Pre-SetupDW	Spark	wvillano	2.4 MIN	Sep 11, 2021, 1:34:29 PM	⋮

Give it a minute for this job to complete. With the second job we will create a new table with enriched data. To create the next job. Click on Jobs in the left pane and then click Create Job again:

1. Job Type: Spark
2. Name: <Pre-Fix>-EnrichData_ETL
3. Application File: File
4. Upload File: <Pre-Fix>-EnrichData_ETL.py
5. Select Python 3
6. Leave Advanced Options and Scheduling to default settings
7. Hit button Create and Run

You can check again at Job Runs.

This job will take a couple of minutes depending on resources and concurrency. In the meantime you can have a closer look at the Enrich script to see what is happening. Code snippet:



```
#---------------------------------------------------
#               JOIN DATA INTO ONE TABLE
#---------------------------------------------------
# SQL way to do things
salesandcustomers_sql = "SELECT customers.*, sales.sale_date, sales.saleprice, sales.model, sales.VIN \
                          FROM <Pre-Fix>_CAR_DATA.car_sales sales JOIN <Pre-Fix>_CAR_DATA.customer_data customers \
                           ON <Pre-Fix>_CAR_DATA.customer_id = <Pre-Fix>_CAR_DATA.customer_id "
tempTable = spark.sql(salesandcustomers_sql)
if (_DEBUG_):
    print("\tTABLE: CAR_SALES")
    car_sales.show(n=5)
    print("\tTABLE: CUSTOMER_DATA")
    customer_data.show(n=5)
    print("\tJOIN: CAR_SALES x CUSTOMER_DATA")
    tempTable.show(n=5)


# Add geolocations based on ZIP
tempTable = tempTable.join(geo_data, "zip")
if (_DEBUG_):
    print("\tTABLE: GEO_DATA_XREF")
    geo_data.show(n=5)
    print("\tJOIN: CAR_SALES x CUSTOMER_DATA x GEO_DATA_XREF (zip)")
    tempTable.show(n=5)
```

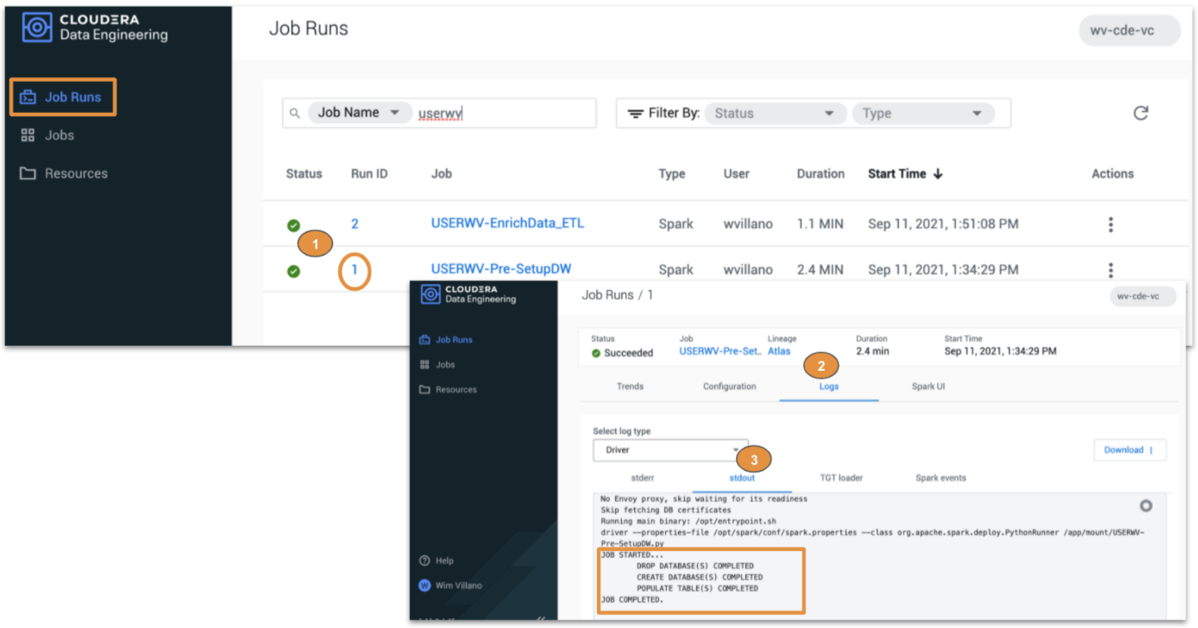
When the job has finished, review the Job Output.

First, let's take a look at the output for <Pre-Fix>-Pre-SetupDW:

Select *Job Runs* tab.

1. Select the Run ID number for your Job name
2. Select Logs
3. Select stdout

The results should look like this:



Status	Run ID	Job	Type	User	Duration	Start Time	Actions
✓	2	USERWV-EnrichData_ETL	Spark	wvillano	1.1 MIN	Sep 11, 2021, 1:51:08 PM	⋮
✓	1	USERWV-Pre-SetupDW	Spark	wvillano	2.4 MIN	Sep 11, 2021, 1:34:29 PM	⋮

Status	Job	Lineage	Duration	Start Time
Succeeded	USERWV-Pre-Set...	Atlas	2.4 min	Sep 11, 2021, 1:34:29 PM

Select log type: Driver

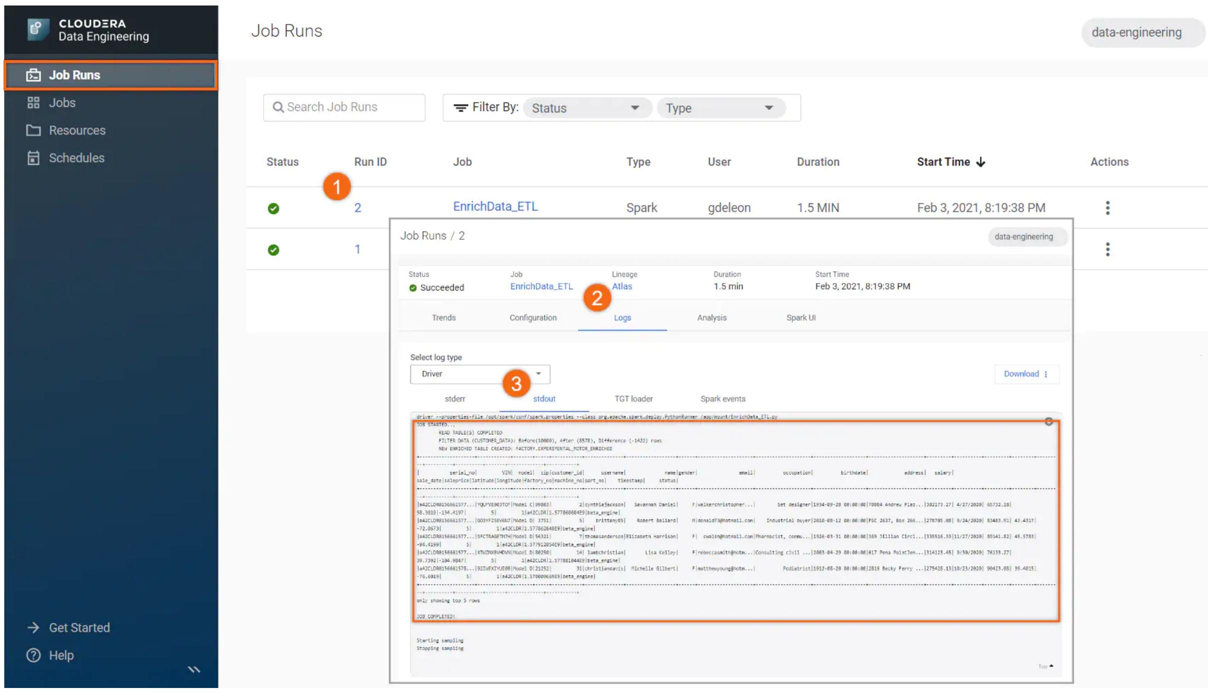
```
No Envoy proxy, skip waiting for its readiness
Skip fetching DB certificates
Running main binary: /opt/entrypoint.sh
driver --properties-file /opt/spark/conf/spark.properties --class org.apache.spark.deploy.PythonRunner /app/mount/USERWV-
Pre-SetupDW.py
JOB STARTED...
        DROP DATABASE(S) COMPLETED
        CREATE DATABASE(S) COMPLETED
        POPULATE TABLE(S) COMPLETED
JOB COMPLETED.
```

Next, let's take a look at the job output for EnrichData_ETL:

Select *Job Runs* tab.

1. Select the Run ID number for your Job name
2. Select Logs
3. Select stdout

The results should look like this:



Job Runs

Status	Run ID	Job	Type	User	Duration	Start Time	Actions
✓	2	EnrichData_ETL	Spark	gdeleon	1.5 MIN	Feb 3, 2021, 8:19:38 PM	⋮
✓	1						⋮

Job Runs / 2

Status	Job	Lineage	Duration	Start Time
Succeeded	EnrichData_ETL	Atlas	1.5 min	Feb 3, 2021, 8:19:38 PM

Trends Configuration Logs Analysis Spark UI

Select log type: Driver

stderr stdout TGT loader Spark events

```
driver --properties-file /opt/spark/conf/spark.properties --class org.apache.spark.deploy.PythonRunner /app/mount/EnrichData_ETL.py
JOB STARTED...
        READ TABLE(S) COMPLETED
        FILTER DATA (CUSTOMER_DATA): Before(10000), After (8578), Difference (-1422) rows
        NEW ENRICHED TABLE CREATED: FACTORY.EXPERIMENTAL_MOTOR_ENRICHED
+-----------------------------------------------------------------------------------------------------------------------------------------
--+----------+---------+---------+----------+
|          serial_no|          VIN|  model|  zip|customer_id|       username|              name|gender|               email|          occupation|          birthdate|            address|   salary|
sale_date|saleprice|latitude|longitude|factory_no|machine_no|part_no|     timestamp|     status|
+-----------------------------------------------------------------------------------------------------------------------------------------
--+----------+---------+---------+----------+
|a42CLDR0156661577...|YQUPVE9OJTOF|Model C|99003|          2|cynthiajackson|   Savannah Daniel|     F|walkerchristopher...|       Set designer|1934-09-28 00:00:00|70884 Andrew Plaz...|302173.27| 4/27/2020| 65732.18|
58.3819|-134.4197|         5|         1|a42CLDR|1.57786068 4E9|beta_engine|
|a42CLDR0156661577...|GO3YFZ5EV6N7|Model D| 3751|          5|     brittany85|    Robert Ballard|     M|donald73@hotmail.com|    Industrial buyer|2010-08-12 00:00:00|PSC 2637, Box 266...|278705.08| 9/24/2020| 83483.91| 43.4317|
-72.0673|         5|         1|a42CLDR|1.57786284 8E9|beta_engine|
|a42CLDR0156661577...|SFCTRAGETH7H|Model D|56321|          7|thomasanderson|Elizabeth Harrison|     F|  cwalsh@hotmail.com|Pharmacist, commu...|1926-03-31 00:00:00|369 Jillian Circl...|335516.33|11/27/2020| 89141.82| 45.5783|
-94.4199|         5|         1|a42CLDR|1.57791285 4E9|beta_engine|
|a42CLDR0156661577...|XTWZNXBVHDVN|Model D|80250|         14| lambchristian|       Lisa Kelley|     F|rebeccasmith@hotm...|Consulting civil ...|2003-04-29 00:00:00|617 Pena PointJen...|314123.45| 9/30/2020| 76133.27|
39.7392|-104.9847|         5|         1|a42CLDR|1.57788104 4E9|beta_engine|
|a42CLDR0156661578...|92IWEXIYUI08|Model D|21252|         31|christiandavis|  Michelle Gilbert|     F|matthewyoung@hotm...|          Podiatrist|1912-08-20 00:00:00|2819 Becky Ferry ...|275428.13|10/23/2020| 90423.08| 39.4015|
-76.6019|         5|         1|a42CLDR|1.57800966 9E9|beta_engine|
+-----------------------------------------------------------------------------------------------------------------------------------------
--+----------+---------+---------+----------+
only showing top 5 rows

JOB COMPLETED!

Starting sampling
Stopping sampling
```

Download ⋮

Top ▲

Summary of lab 1:

In this lab you have created 2 jobs via the GUI. It provides an easy way for developers to run workloads.

In the next lab you will see how you can orchestrate multiple jobs with Apache Airflow.

4. Lab 2: Automating data pipelines with Airflow

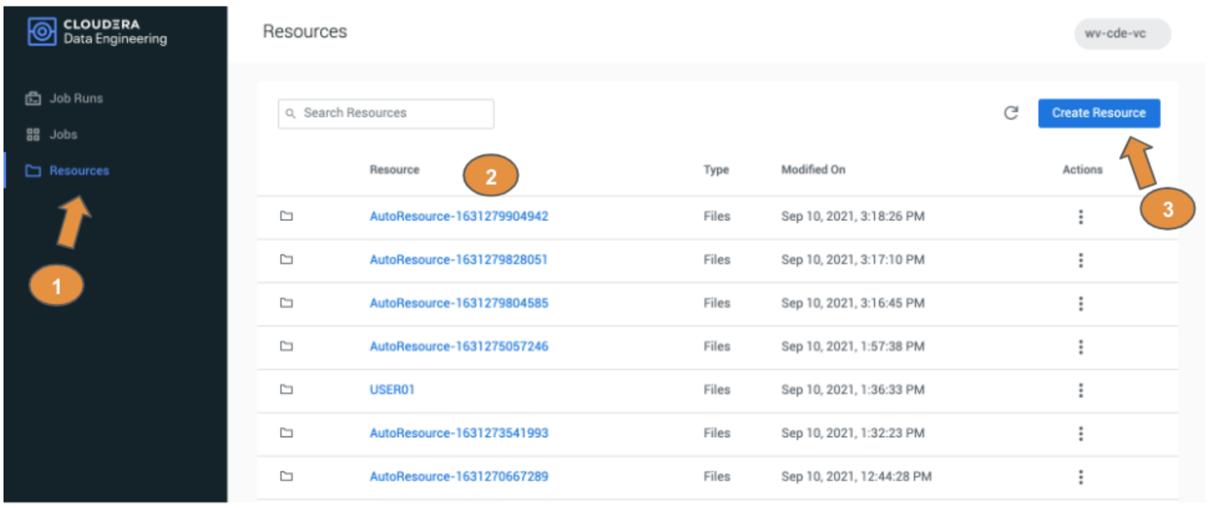
Cloudera Data Engineering (CDE) enables you to automate a workflow or data pipeline using Apache Airflow Python DAG files. Each CDE virtual cluster includes an embedded instance of Apache Airflow. CDE currently (September 2021) supports two Airflow operators; one to run a CDE job and one to access Cloudera Data Warehouse (CDW).

In this lab we will create a pipeline flow of the two jobs created in Lab 1.

We will have a different approach for creating the Airflow job. Now we will make use of a Resource (via the GUI it is implicit of type *Files*). A resource of type *Files* is a named collection of files used by a job. It can be an arbitrary collection of files that a job can reference: the application code for the job, including any necessary configuration files or supporting libraries.
Resources can also be used for more advanced usages like Python virtual environment specifications (type *python-env)* or a custom Docker container image

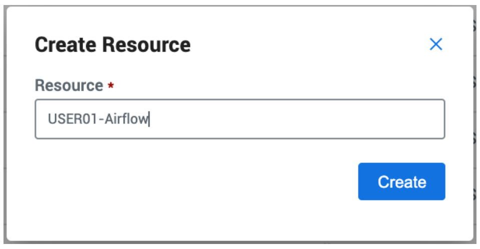
(type *custom-runtime-image*). These need to be created via cde CDE CLI. But we won't make use of that today.

Click on the left pane on Resources (1). You will then also see all automatic created resources (2). When you create a job like in Lab1, the system will automatically create a Resource for the job. If you click on one of the resources you'll see the uploaded Python file. But as you can see it generates a unique name which is difficult to reference. For the Airflow job we will create a new Resource (3). Click on Create Resource.



Resource	Type	Modified On	Actions
AutoResource-1631279904942	Files	Sep 10, 2021, 3:18:26 PM	⋮
AutoResource-1631279828051	Files	Sep 10, 2021, 3:17:10 PM	⋮
AutoResource-1631279804585	Files	Sep 10, 2021, 3:16:45 PM	⋮
AutoResource-1631275057246	Files	Sep 10, 2021, 1:57:38 PM	⋮
USER01	Files	Sep 10, 2021, 1:36:33 PM	⋮
AutoResource-1631273541993	Files	Sep 10, 2021, 1:32:23 PM	⋮
AutoResource-1631270667289	Files	Sep 10, 2021, 12:44:28 PM	⋮

Give the resource a name: <Pre-Fix>-Airflow (with your User_Identifier as <PRe-Fix>) and hit Create:

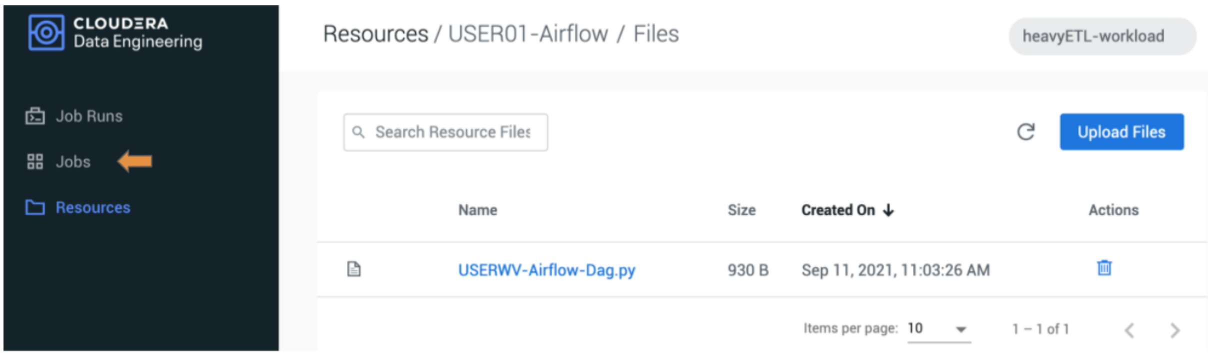


Resource *
USER01-Airflow

Before you can upload the Airflow script you have to change/replace some placeholders in the script <Pre-Fix>-Airflow-Dag.py.

Open the script <Pre-Fix>-Airflow-Dag.py in your favorite editor. You need to find/replace placeholder <Pre-Fix> with the User_Identifier obtained in paragraph 2. Save the script.

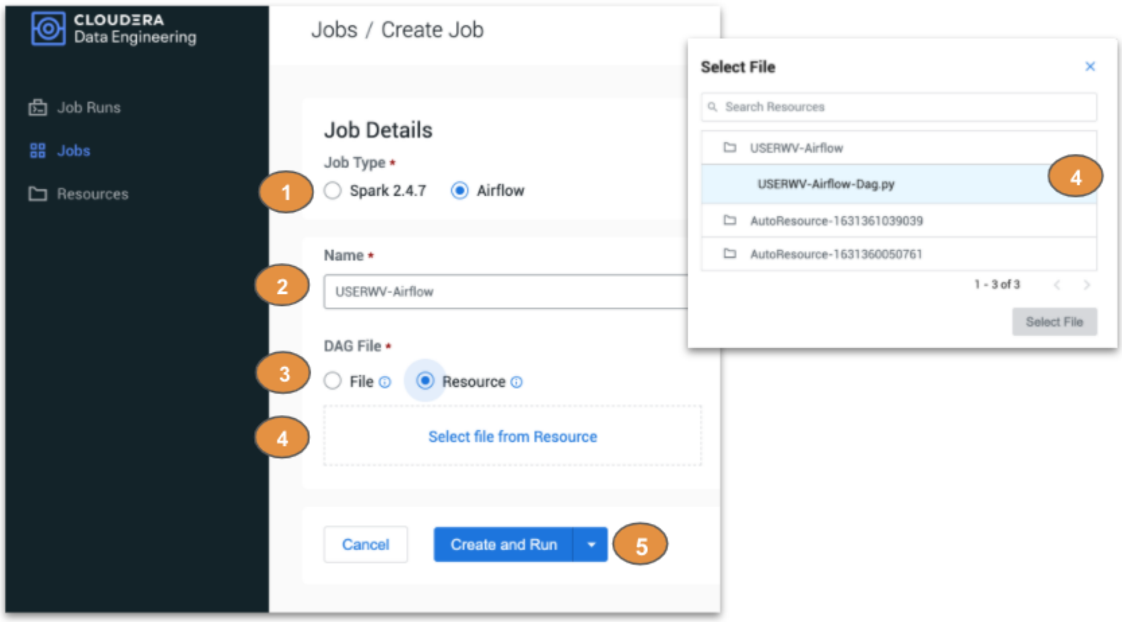
Now you can upload the file <Pre-Fix>-Airflow-Dag.py. Refresh the page after upload. You'll see the file. Click on Jobs in the left pane.



Name	Size	Created On	Actions
USERWV-Airflow-Dag.py	930 B	Sep 11, 2021, 11:03:26 AM	

To create a job for the Airflow task Click the button Create Job with the following information:

1. Job Type: Airflow
2. Name: <Pre-Fix>-Airflow
3. DAG File: Resource
4. Select Uploaded File in your Resource folder
5. Click Create and Run



Job Type: ○ Spark 2.4.7 [✓ Airflow]

Name: USERWV-Airflow

DAG File: ○ File [✓ Resource]

Below a couple of code snippets of the script <Pre-Fix>-Airflow-Dag.py are highlighted :
Cloudera specific operators start with cloudera.cdp. :

```
from cloudera.cdp.airflow.operators.cde_operator import CDEJobRunOperator
from cloudera.cdp.airflow.operators.cdw_operator import CDWOperator
```

The parameter job_name is referencing a job which has already been created. If you followed the naming convention in Lab 1 you are ok:

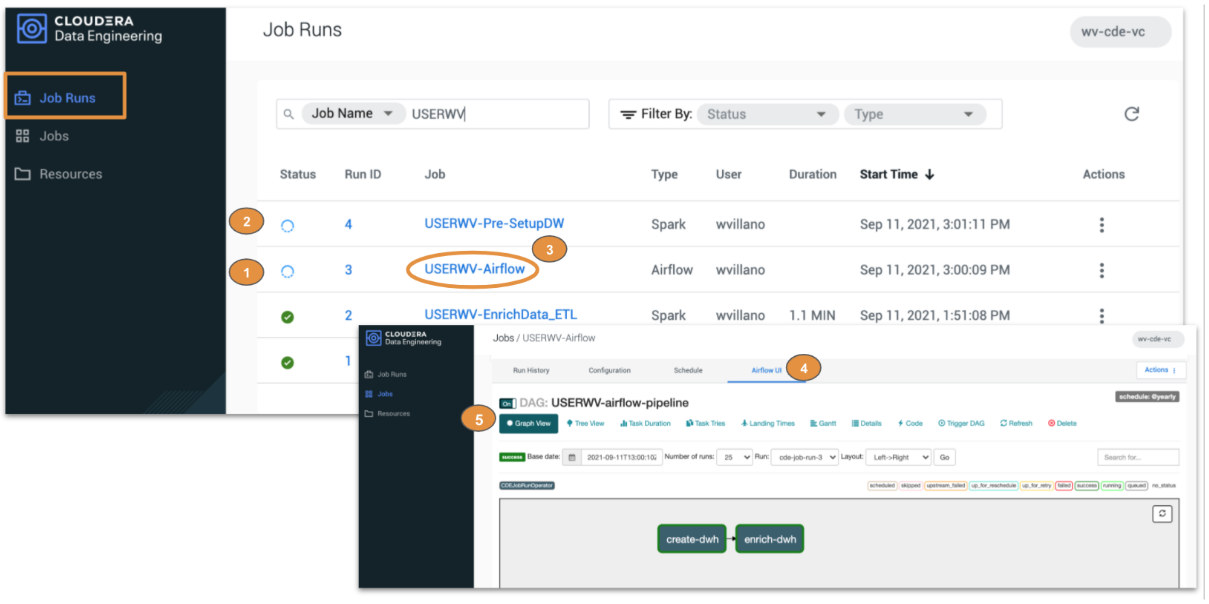
```
create_step1 = CDEJobRunOperator(
    task_id='create-dwh',
    dag=dag,
    job_name='<Pre-Fix>-Pre-SetupDW'
)
```

At the bottom of the script the dependencies between the jobs are defined: first run create_step1 (which is the job with job name <Pre-Fix>-Pre-SetupDW) and then run enrich_step2 (which is the job with job name <Pre-Fix>-EnrichData_ETL).

```
create_step1 >> enrich_step2
```

For more options see Airflow documentation:
https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html#example-pipeline-definition

Now let's view the Airflow Job while running. Click on Job Runs in the left pane. See the <Pre-Fix>-Airflow job running (1) which starts the other jobs (2).
You can click on the Aiflow job name (3) which will take you to the details of the job. Click on the tab Airflow UI (4) and then Graph View (5) to get a graphical overview of this simple pipeline with 2 jobs.



Status	Run ID	Job	Type	User	Duration	Start Time	Actions
	4	USERWV-Pre-SetupDW	Spark	wvillano		Sep 11, 2021, 3:01:11 PM	⋮
	3	USERWV-Airflow	Airflow	wvillano		Sep 11, 2021, 3:00:09 PM	⋮
✓	2	USERWV-EnrichData_ETL	Spark	wvillano	1.1 MIN	Sep 11, 2021, 1:51:08 PM	⋮
✓	1						

Jobs / USERWV-Airflow

Run History | Configuration | Schedule | Airflow UI | Actions

On DAG: USERWV-airflow-pipeline

schedule: @yearly

Graph View | Tree View | Task Duration | Task Tries | Landing Times | Gantt | Details | Code | Trigger DAG | Refresh | Delete

success Base date: 2021-09-11T13:00:10 Number of runs: 25 Run: cde-job-run-3 Layout: Left->Right Go Search for...

CDEJobRunOperator

scheduled skipped upstream_failed up_for_reschedule up_for_retry failed success running queued no_status

This ends Lab2. In this lab you run and examine an Airflow job in which a pipeline was created where the jobs from Lab 1 were sequentially executed.

5. Optional Lab 3: Using CLI to automate access to CDE

For this lab you need to have installed the CDE CLI tool.

Everything we have done from the GUI so far can also be done from a command line with the CDE CLI tool.

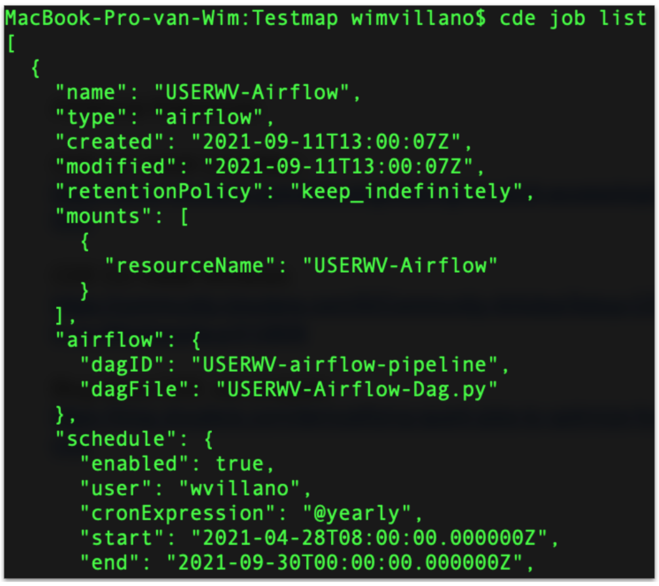
Be sure to go in a terminal to the directory with the *.py scripts used in the previous labs. When prompted to enter your API User Password then enter your Workload-user password from CDP.

On a Mac to start the command is: cde on linux start the command with ./cde

You can get an overview of the jobs currently available:

```
cde job list
```

Which returns a json based output with all the jobs defined:

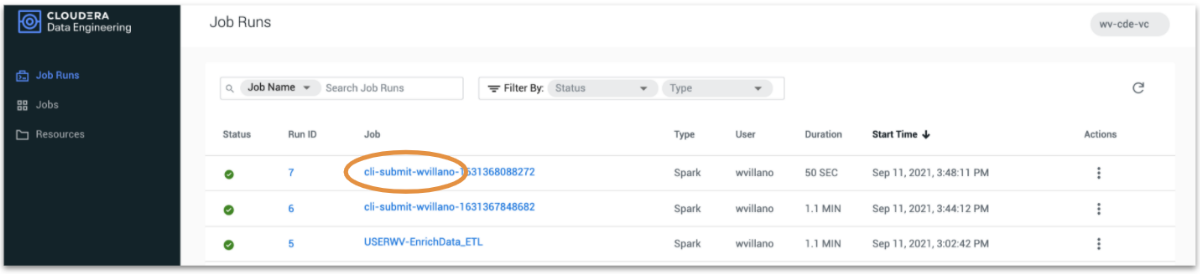


```
MacBook-Pro-van-Wim:Testmap wimvillano$ cde job list
[
  {
    "name": "USERWV-Airflow",
    "type": "airflow",
    "created": "2021-09-11T13:00:07Z",
    "modified": "2021-09-11T13:00:07Z",
    "retentionPolicy": "keep_indefinitely",
    "mounts": [
      {
        "resourceName": "USERWV-Airflow"
      }
    ],
    "airflow": {
      "dagID": "USERWV-airflow-pipeline",
      "dagFile": "USERWV-Airflow-Dag.py"
    },
    "schedule": {
      "enabled": true,
      "user": "wvillano",
      "cronExpression": "@yearly",
      "start": "2021-04-28T08:00:00.000000Z",
      "end": "2021-09-30T00:00:00.000000Z",
```

But you can also immediately submit a job from the command line. For this we will use the script <Pre-Fix>-EnrichedData-ETL.py already used in Lab 1:

```
cde spark submit <Pre-Fix>-EnrichData-ETL.py
```

Which will show up in the Job Run with a special prefix: “cli-submit”



Status	Run ID	Job	Type	User	Duration	Start Time	Actions
✓	7	cli-submit-wvillano-1631368088272	Spark	wvillano	50 SEC	Sep 11, 2021, 3:48:11 PM	⋮
✓	6	cli-submit-wvillano-1631367848682	Spark	wvillano	1.1 MIN	Sep 11, 2021, 3:44:12 PM	⋮
✓	5	USERWV-EnrichData_ETL	Spark	wvillano	1.1 MIN	Sep 11, 2021, 3:02:42 PM	⋮

You can also create a Resource:

```
cde resource create --name "<Pre-Fix>-cli-resource"
```

Upload a script to it. In this case we will use again <Pre-Fix>-EnrichedData-ETL.py:

```
cde resource upload --local-path "<Pre-Fix>-EnrichData_ETL.py"
--name "<Pre-Fix>-cli-resource"
```

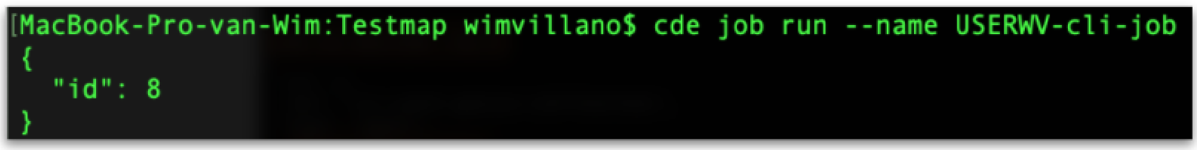
Now you can create a job:

```
cde job create \
--name <Pre-Fix>-cli-job \
--type spark \
--mount-1-resource <Pre-Fix>-cli-resource \
--application-file <Pre-Fix>-EnrichData_ETL.py
```

And run the job:

```
cde job run --name <Pre-Fix>-cli-job
```

The output will be the job id:



```
[MacBook-Pro-van-Wim:Testmap wimvillano$ cde job run --name USERWV-cli-job
{
  "id": 8
}
```

You can use this job-id to get an update of the status of the job:

```
cde run describe --id <job-id>
```

This lab was a short introduction to how you can create and manage jobs from a command line which can help you to automate and reach higher productivity.

For complete overview of CLI commands see:
https://docs.cloudera.com/data-engineering/cloud/cli-access/topics/cde-cli-reference.html

Additional References

CDE CLI install Linux/Mac:
https://docs.cloudera.com/data-engineering/cloud/cli-access/topics/cde-download-cli.html

CDE CLI install Windows:
https://community.cloudera.com/t5/Community-Articles/Setup-CDE-CLI-with-Git-Bash-on-Windows/ta-p/312808

Blog about CDE Job analysis:
https://blog.cloudera.com/demystifying-spark-jobs-to-optimize-for-cost-and-performance/

Different ways to manage dependencies:
https://blog.cloudera.com/managing-python-dependencies-for-spark-workloads-in-cloudera-data-engineering/