

**Información sobre el documento:**

<b>Año: 2022</b>	Especialización en Ciencia de Datos (2021-B)	Materia: Seminario
<b>Entregable N°1</b>	Fecha Entrega: 2022-11-01	Alumna: Carla Olmo colmo@itba.edu.ar

## Enunciado del Trabajo Práctico

### Proyecto del Seminario

- Realizar análisis de datos (**individual o 2 personas máximo**)
- Tomar información de distintas fuentes y generen métricas y/o modelos predictivos
- El resultado debe ser guardado en Postgres o en archivos Parquet.
- Crear visualizaciones para ilustrar el análisis y los resultados.
- Inspirarse en proyectos laborales/personales o para el TFI.

## Objetivo de la Solución

Presentar en forma horaria una proyección de la demanda para las próximas 24 horas.

## Conceptos de la implementación de la solución

Como no trabajé con Docker antes, enfoqué en intentar armar un ambiente que resuelva el ETL de datos y la ejecución periódica de la predicción de la demanda de energía eléctrica, para que me sirva de base para el trabajo final.

En este contexto, enfoqué más en la infraestructura (hacer funcionar los dockers, que interactúen, hacer funcionar Airflow con una imagen diferente por la necesidad de incorporar paquetes de Python que no están en la imagen oficial de Apache, etc) Por esto, no hay un gran trabajo en encontrar el mejor modelo que haga el forecast ni un dashboard que muestre muchos datos. Esto lo dejo para, con más tiempo, el trabajo final.

La solución tiene como tareas macro:

- Obtener en forma horaria el dato de demanda horaria (y temperatura) de energía eléctrica, a nivel total Argentina. Este dato es publicado por una API implementada por la empresa Cammesa. Registrar este dato en una base de datos donde se mantiene la historia.
- Con los datos obtenidos y un modelo pre fiteado, obtener una predicción para las próximas 24 horas.
- Exponer el histórico y la proyección en un dashboard.

La solución se implementó en una serie de containers, permitiendo portabilidad de la solución y reducir tiempo de configuración de los entornos de ejecución de cada bloque de la solución.

## Esquema de Containers

La solución se desarrolló en una máquina Windows 10 con 16 Gb de ram. Para poder ejecutar Docker se instaló WSL y luego Docker. Como IDE se utilizó VS Code.

## Solución PROD

## Soporte DEV

Airflow  
webserver

db

Superset

Jupyter

Airflow  
scheduler

Redis  
superset

Pg-admin

postgres

En el entregable se incluyen tanto los containers necesarios para la solución en ambiente productivo (color azul de fondo) y los que se utilizaron en la etapa de desarrollo (color gris de fondo)

Más detalle sobre los containers al final del documento.

## Cómo ejecutar la solución?

- 1- Clonar el repo [https://github.com/colmo786/tp\\_docker.git](https://github.com/colmo786/tp_docker.git)
- 2- Ir a la carpeta /docker
- 3- Bultear la imagen de Airflow con el comando `./dockers.sh build_airflow`.

Se bulteará la versión 2.4.2 de Airflow, a la que se le agregan librerías para forecasting. Se bultea antes del resto porque agregar el build en el Docker compose general bultea la imagen por cada container que la usa.

<https://stackoverflow.com/questions/40899236/how-to-prevent-docker-compose-building-the-same-image-multiple-times>

```
=> [2/3] RUN pip install --upgrade pip
=> [3/3] RUN pip install tensorflow
=> => # Collecting tensorflow
=> => #   Downloading tensorflow-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (578.0 MB)
```

```
=> [2/3] RUN pip install --upgrade pip
=> [3/3] RUN pip install tensorflow
=> => #   Uninstalling protobuf-3.20.0:
=> => #     Successfully uninstalled protobuf-3.20.0
=> => #   Attempting uninstall: google-auth-oauthlib
=> => #     Found existing installation: google-auth-oauthlib 0.5.3
=> => #   Uninstalling google-auth-oauthlib-0.5.3:
=> => #     Successfully uninstalled google-auth-oauthlib-0.5.3
```

- 4- [Opcional] Bultear la imagen de Jupyterlab con el comando `./dockers.sh build_jupyter`

Se bulteará una versión de Jupyter a la que se le agregan librerías para forecasting. Esta imagen se utilizó para desarrollo y pruebas del código luego volcado en los DAGs.

```
Creating volume "energy_sup
Building jupyter
[+] Building 12.0s (3/6)
```

Nota: se tuvo que instalar la versión sklearn 1.0 que es compatible con Python 3.7. <https://scikit-learn.org/stable/install.html>

- 5- Inicializar Airflow con el comando `./dockers.sh init_airflow`. Esta serie de comandos actualizan la DB de airflow y crean el usuario admin/admin para acceder a esa aplicación.

```
airflow-init_1 | 2.4.2
docker_airflow-init_1 exited with code 0
Stopping docker_postgres_1 ... done
Removing docker_airflow-init_1 ... done
Removing docker_postgres_1 ... done
Removing network docker_default
Removing volume docker_postgres-db-volume
Removing volume docker_local_pgdata
Removing volume docker_superset
If you read 'docker_airflow-init_1 exited with code 0', then you can exec start_all
```

6- Si el comando anterior terminó con code 0, ejecutar `./docker.sh start_all`. Demora unos minutos.

Se pullean el resto de las imágenes, en caso de no existir en el host.

```
Creating pgadmin4 ... done
Creating jupyter ... done
Creating superset ... done
Creating energy_airflow-init_1 ... done
Creating energy_airflow-scheduler_1 ... done
Creating energy_airflow-webserver_1 ... done

Everything is ready, access Superset at http://localhost:/
Airflow at http://localhost:8080/
Jupyterlab at http://localhost:8888/
pgAdmin at http://localhost:5050/
```

En el Docker Desktop quedan 6 containers (de 7) ejecutándose:

<input type="checkbox"/>	energy 7 containers	-	Running (6/7)	-	<a href="#">Open</a>			
<input type="checkbox"/>	redis_superset 16b1d96433ec	redis:latest	Running	-	16 minutes ag			
<input type="checkbox"/>	energy_postgres_1 75a0fd5c73a8	postgres:13	Running	-	16 minutes ag			
<input type="checkbox"/>	local_pgdb 72167d137cd9	postgres:latest	Running	5432	16 minutes ag			
<input type="checkbox"/>	superset 40c6f5168d2a	amancevice/superset:latest	Running	8088	16 minutes ag			
<input type="checkbox"/>	energy_airflow-init_1 112a4e71479a	airflow/built-in:latest	Exited	-				
<input type="checkbox"/>	energy_airflow-scheduler_1 6800cd8e39a3	airflow/built-in:latest	Running	-	21 seconds ag			
<input type="checkbox"/>	energy_airflow-webserver_1 b6fdb429191d	airflow/built-in:latest	Running	8080	15 minutes ag			

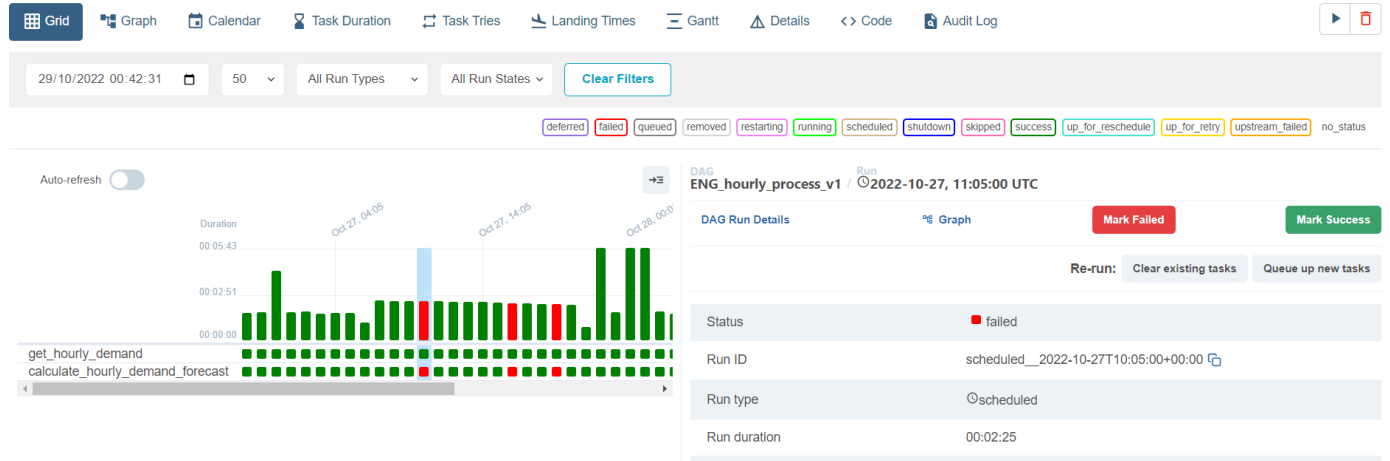
7- Acceder a Airflow en <http://localhost:8080/home>. Usuario: airflow Password: airflow

NOTA: Intenté cambiar el port como para otras aplicaciones pero no me funcionó. Sería mejor poder cambiarlo para que no use el por default, que es utilizado por otras aplicaciones (no del paquete “energy”)

NOTA: dependiendo de los recursos de la máquina, los dockers pueden demorar en terminar de cargarse y en alguna oportunidad se experimentó que Airflow detecta errores en los DAGs (no encuentra la ubicación de tensorflow, o similar) pero con refresh comienza a funcionar.

8- Habilitar el DAG ENG\_hourly\_process\_v1. Tiene como fecha de inicio el día posterior al último dato histórico pre cargado en la base de datos. Ejecutará un backfill (cada hora, a los 5 minutos) hasta la hora en que se habilita el DAG.

DAG	Owner	Runs	Schedule	Last Run	Next Run
ENG_hourly_process_v1	energy		5 * * * * *		2022-10-26, 00:05:00



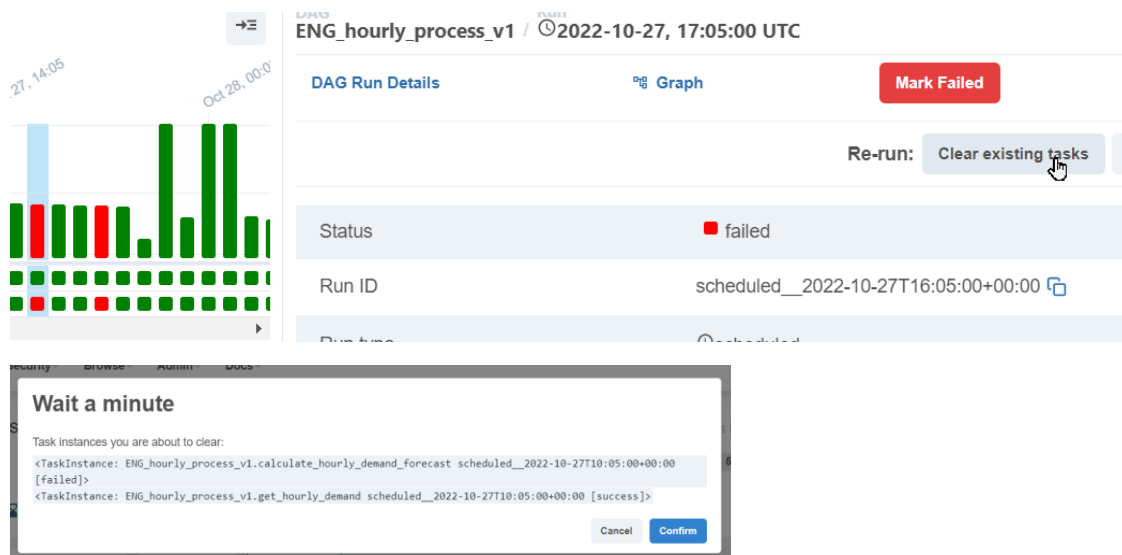
Este proceso incorporará o modificará registros en la base de datos. El registro se inserta o se updatea con la fecha y hora actual (no la de “ejecución” de airflow) NOTA: no me llego a dar cuenta a priori, si esto debe ser así o si conviene usar la fecha de ejecución de Airflow, podría ser para asociar el registro al log de ejecución de la tarea?

hourly_demand integer	hourly_temp double precision	day_of_week integer	is_holiday integer	create_user character (10)	create_date timestamp without time zone	update_user character (10)	update_date timestamp without time zone
16282	19.5	2	0	COLMO	2022-10-26 21:18:08.222296	COLMO	2022-10-26 21:36:29.492749
16784	20.6	2	0	COLMO	2022-10-26 21:18:08.222337	COLMO	2022-10-26 21:36:29.492781
17123	21.9	2	0	COLMO	2022-10-26 21:18:08.222392	COLMO	2022-10-26 21:36:29.492812

## Troubleshooting

Durante el backfill, ee detectaron casos en los que, por falta de recursos, las tareas de los DAGs terminaron con error.

Para volver a ejecutarlas, seleccionar la barra correspondiente a la ejecución con error y presionar el botón `Clear existing tasks` Esto permitirá que se repita la ejecución.



9- Los resultados se muestran en un Dashboard creado en Superset. <http://localhost:8088/login/>

Usuario: admin Password: admin. Dashboard: dashboard\_test.

Presenta la información real de demanda de energía eléctrica total Argentina, publicada por Cammesa, en forma horaria hasta el 26/10/2022 y a partir de esa fecha, publica tanto el dato real como el pronóstico de la siguiente semana.

## Dashboards

OWNER:  CREATED BY:

Title:

☆ [dashboard\\_test](#)

Superset Dashboards Charts SQL Lab Data

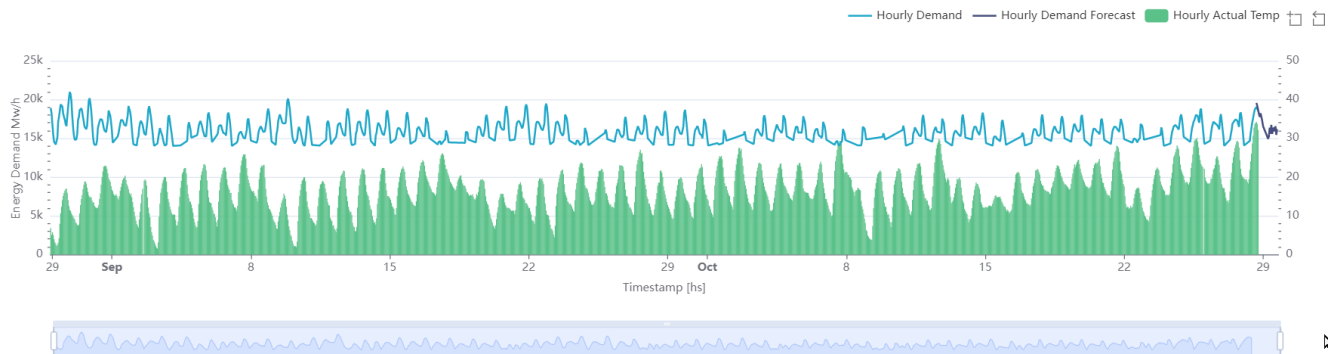
+ - Sett

dashboard\_test ☆ Draft

EDIT DASHBOARD

### Energy App - Hourly Demand Dashboard

Demand Last 2 Months - Forecast 1 Week



10- Para bajar toda la solución, ejecutar `./dockers.sh stop_all`

## Containers de Soporte

### Jupyter Lab

Para acceder a un container con Jupyter Lab conectado a la red de dockers, ejecutar `./dockers.sh start_jupyter`

NOTA: el container de jupyter debe haber sido builteado con anterioridad (punto 4. de la lista)

```

(base) colmo@DESKTOP-0FBKFFQ:~/docker$ ./dockers.sh start_jupyter
Starting up Jupyter lab
Creating network "jupyter_default" with the default driver
Creating jupyter ... done

To access Jupyterlab, please user link with token in logs of container.

(base) colmo@DESKTOP-0FBKFFQ:~/docker$
  
```

Acceder al log del container y obtener el link:

```

jupyter docker-jupyter-scipy-extensible
RUNNING

Entered start.sh with args: jupyter lab
Executing the command: jupyter lab
[I 2022-10-29 01:22:51.873 ServerApp] jupyterlab | extension was successfully linked.
[W 2022-10-29 01:22:51.879 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be
lease.
[W 2022-10-29 01:22:51.879 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp.
release.
[W 2022-10-29 01:22:51.879 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp.
release.
[I 2022-10-29 01:22:51.893 ServerApp] nbclassic | extension was successfully linked.
[I 2022-10-29 01:22:51.906 ServerApp] Writing Jupyter server cookie secret to /home/jovyan/.local/share/jupyter/runtime/jupyter_c
[I 2022-10-29 01:22:51.921 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.10/site-packages/jupyterlab
[I 2022-10-29 01:22:51.921 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2022-10-29 01:22:51.926 ServerApp] jupyterlab | extension was successfully loaded.
[I 2022-10-29 01:22:51.930 ServerApp] nbclassic | extension was successfully loaded.
[I 2022-10-29 01:22:51.931 ServerApp] Serving notebooks from local directory: /home/jovyan
[I 2022-10-29 01:22:51.931 ServerApp] Jupyter Server 1.21.0 is running at:
[I 2022-10-29 01:22:51.931 ServerApp] http://2caced1b41e:8888/lab?token=f3f0934774ad2be3748220296424dbccd16074f633beab0f
[I 2022-10-29 01:22:51.931 ServerApp] or http://127.0.0.1:8888/lab?token=f3f0934774ad2be3748220296424dbccd16074f633beab0f
[I 2022-10-29 01:22:51.931 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2022-10-29 01:22:51.936 ServerApp]

```

Para stoppear el docker: `./dockers.sh stop_jupyter`

### Pg-Admin

Para acceder a un container con pg-Admin conectado a la red de dockers, ejecutar `./dockers.sh start_pgadmin`

Usuario: [user@live.com.ar](mailto:user@live.com.ar) Password: postgres

Se accede a la UI en: <http://localhost:5050/login>



Para stoppear el docker: `./dockers.sh stop_pgadmin`

## Anexo Dockers

### db

Este Docker corre el servidor de la base de datos postgres donde se mantiene la información histórica y los resultados de las proyecciones.

En la primer ejecución del Docker compose, al iniciarse el servicio ejecutará el script `./postgres/scripts/init.sql` que crea la base de datos y la completa con la información histórica.

En el Docker-compose se monta un volumen del host, con lo cual, se asegura que los datos no se pierden si los containers se eliminan. Los datos sólo se perderán en la situación en que se elimina este volumen.

La imagen es postgres oficial.

### pg-admin

Para administrar la base de datos, se instaló en un container la herramienta pgAdmin.

Con este container en la red de containers, se creó el esquema, las tablas y se pudieron realizar análisis sobre los datos. Se mantiene en la solución como soporte a la misma.

Se expone en `https://localhost:5050`. Usuario [user@live.com](mailto:user@live.com) password postgres

### Airflow

Se re buiteó la imagen oficial de Apache, según indicaciones de la página principal <https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html>

Esta imagen tuvo que ser extendida con la instalación de los paquetes scikit-learn y tensorflow.

Se ejecuta con la opción LocalExecutor.

En esta herramienta se definen los flujos de trabajo de ejecución periódica. Se creó 1 DAG con 2 tareas, que se ejecutan en forma horaria, a los 5 min de la hora.

- La primer tarea recolecta los datos de demanda horaria publicados por Cammesa en la API pública <https://api.cammesa.com/demanda-svc/swagger-ui.html> y los almacena en la base de datos postgres "local\_pgdb", en el servicio que corre en el container "db". Todo el código soporte de este DAG (funciones de request a URL, de conexión y acceso a la base de datos, etc) se encuentran registradas en el archivo `energy.py`, en la carpeta de DAGs. Se utilizan las variables de entorno de Airflow para guardar los parámetros de acceso a esta base de datos (ver Docker compose de la solución) Este DAG tiene como fecha de inicio de ejecución el último día de datos almacenados en la historia. De esta manera, al ejecutarse por primera vez el DAG, Airflow hará un backfill hasta la fecha actual.
- La segunda tarea, dependiente de la primera, obtiene una proyección de la demanda para las próximas 24 horas. Este DAG obtiene la historia de la base de datos postgres y registra la proyección en la misma base de datos (de manera que pueda ser consultado por dashboards de Superset) Según recomendación de la página oficial de Airflow, se desalienta el uso de xcom para transferir dataframes entre tareas.

Para conectar el servidor de Airflow con el servidor de Postgres se utiliza la red creada por default por el Docker-compose.

La UI de airflow se expone en `https://localhost:8080`. Usuario: airflow, password: airflow.

Logging de errores: se utiliza directamente el std output de Python (función print) lo que hace que la descripción del error quede en el log del docker. No tuve suficiente tiempo para investigar si existen mejores alternativas a esto.

Así mismo, por lo que se pudo entender del uso de DAGs, cada tarea termina con error únicamente si el operador (en este caso, PythonOperator) sale con error. Para que el operador de Python salga con error, el programa debe haber tenido un error o haberse ejecutado un `raise error`. Con lo cual, se implementan pocos try-except y los que se

implementan, terminan con un raise error luego de haber registrado en el log la descripción del error y parámetros que ayuden a determinar la causa.

## Superset

Se utilizó la imagen provista en el seminario.

Se creó una conexión a la base de datos local\_pgdb, datasets para las tablas y una consulta (saved queries) que integra datos de 2 tablas y se utiliza para el chart del dashboard.

Se publica en <http://localhost:8088/> Usuario: admin, password: admin (se crean al ejecutar el superset-init)

## Jupyterlab

Se instaló un container con la imagen de jupyterlab. Este container se utiliza como soporte para probar en el ambiente los códigos que luego serán utilizados por los DAGs y para ejecutar tareas de creación o carga de datos inicial en la base de datos.

Monta también un volumen del host para poder mantener los notebooks compartidos entre el host y el container y que no se pierdan al eliminar el container.

Para acceder, se debe ejecutar `./dockers.sh start_jupyter` y obtener el link con el token del log del container.