
UNIVERSIDAD AUSTRAL DE CHILE

CENTRO DE DOCENCIA Y CIENCIAS BÁSICAS PARA INGENIERIA



Roble Austral



Manual de uso Apache Airflow y Forms de prácticas

PROFESOR:

Luis Veas {luis.veasc@inf.uach.cl}
Instituto de Informática, Universidad Austral de Chile

ALUMNO:

{vicente.alves@alumnos.uach.cl}

Contenido

Resumen.....	3
Introducción.....	4
Tecnologías.....	5
Instalación y Configuración de Airflow (Máquina Local).....	6
Prerrequisitos.....	6
Pasos para la Instalación.....	6
1. Instalar airflow:.....	6
2. Agregar dependencias adicionales (Base de Datos):.....	6
3. Inicialización de Airflow:.....	6
Como lanzar Apache Airflow.....	7
Instalación y Configuración de Airflow (Contenedor Docker).....	8
Prerrequisitos.....	8
Pasos para la Instalación.....	8
1. Crear un archivo Docker Compose.....	8
2. Preparar el Entorno.....	8
3. Lanzar los Servicios.....	9
4. Acceso a la Interfaz de Airflow.....	9
5. Añadir DAGs.....	9
6. Detener los Servicios.....	9
Instalación y Configuración de NextJS.....	10
Pasos para la Instalación.....	10
Instalación y Configuración de Postgres.....	11
Pasos para la Instalación.....	11
Notas extra:.....	12
Configuración del Proyecto de Prácticas.....	13
Airflow.....	13
NextJs.....	16
Proyecto de Prácticas En Funcionamiento.....	17
Creación de Dags.....	19
1. ¿Qué es un DAG?.....	19
2. Estructura de un DAG en Airflow.....	19
3. Componentes Principales de un DAG.....	20
4. Secuencia de las Tareas.....	21
5. Variables de Airflow.....	21
6. Parámetros del DAG.....	21
7. Manejo de Errores y Retries.....	22
8. SubDAGs.....	22
9. DAGs Dinámicos y Parametrización.....	24
10. Consideraciones de Rendimiento y Escalabilidad.....	24

Resumen

Este manual tiene como objetivo proporcionar una guía detallada para la instalación, configuración y puesta en marcha de las herramientas necesarias para el desarrollo de un proyecto web utilizando tecnologías clave como **Apache Airflow**, **Next.js** y **PostgreSQL**. El documento aborda desde los aspectos más básicos, como la instalación de cada herramienta, hasta la integración de las mismas para el correcto funcionamiento de un sistema de gestión de prácticas.

El manual comienza con la instalación de Apache Airflow, una plataforma de orquestación de flujos de trabajo, para gestionar tareas automatizadas relacionadas con la carga, transformación y análisis de datos. Se detallan los pasos para configurar y ejecutar Airflow, incluyendo la creación de flujos de trabajo (DAGs) y la configuración de variables y conexiones necesarias para interactuar con la base de datos.

En la segunda parte, se aborda la instalación de Next.js, un framework de React, para desarrollar una interfaz web interactiva y dinámica que permita a los estudiantes visualizar sus prácticas y realizar un seguimiento de su progreso. Se incluyen los pasos para configurar el proyecto Next.js, así como la integración con PostgreSQL como sistema de gestión de base de datos.

La tercera sección se centra en la instalación y configuración de PostgreSQL, una base de datos relacional que se utilizará para almacenar la información sobre las prácticas de los estudiantes. Se incluyen instrucciones detalladas para la instalación en sistemas basados en Linux, así como la configuración de usuarios y contraseñas.

Finalmente, se guía al usuario en la configuración del proyecto de prácticas en Airflow y Next.js, detallando cómo migrar y gestionar los flujos de trabajo, establecer las variables necesarias y realizar las conexiones a la base de datos. Con todas estas configuraciones en su lugar, el sistema está listo para ejecutarse, proporcionando una solución integral para la gestión de prácticas y el seguimiento de actividades de los estudiantes.

Introducción

Este manual te guiará en la instalación y configuración de las herramientas necesarias para crear un entorno de desarrollo web moderno. Utilizando **Apache Airflow**, **Next.js** y **PostgreSQL**, aprenderás cómo implementar un sistema que permita gestionar flujos de trabajo automatizados y desarrollar aplicaciones web dinámicas.

Durante el proceso, instalarás y configurarás Apache Airflow para la orquestación de tareas, Next.js para el desarrollo de la interfaz web, y PostgreSQL como base de datos para almacenar la información de las prácticas. Todo esto se integrará para formar un sistema funcional que gestione de manera eficiente las prácticas de los estudiantes.

Requisitos del Sistema

Para seguir este manual, asegúrate de contar con los siguientes requisitos:

- **Sistema operativo:** Linux o **WSL** (Windows Subsystem for Linux).
- **Node.js** (versión 16 o superior) para trabajar con Next.js.
- **PostgreSQL** para gestionar las bases de datos.
- **Python 3.x** y **pip** para instalar Apache Airflow.
- Un editor de código como **Visual Studio Code**.

Con estos requisitos, estarás listo para empezar con la instalación y configuración de las tecnologías necesarias.

Tecnologías

Para este manual, se va a tener en consideración tres principales tecnologías que serán las directrices fundamentales para el desarrollo y despliegue de las prácticas destinadas a los practicantes. Estas tecnologías son:

1. **Apache Airflow**

Es una plataforma de orquestación de flujos de trabajo que permite programar, automatizar y monitorear tareas complejas. Se utilizará para gestionar procesos como la automatización de tareas relacionadas con la carga, transformación y análisis de datos, necesarias para la lectura y evaluación de prácticas realizadas por los practicantes.

2. **Next.js**

Es un framework de React que facilita la creación de aplicaciones web modernas con renderizado del lado del servidor (SSR) y generación de sitios estáticos (SSG). Será utilizado para desarrollar una interfaz interactiva que permita a los practicantes visualizar sus prácticas, recibir retroalimentación y gestionar su progreso de manera dinámica y eficiente.

3. **PostgreSQL**

Es un sistema de gestión de bases de datos relacional de código abierto, conocido por su capacidad para manejar datos estructurados y no estructurados. Servirá como el almacén central para gestionar y organizar los datos de las prácticas de los estudiantes, garantizando integridad, seguridad y fácil acceso para consultas y análisis posteriores.

Estas bases podrán realizar la lectura, gestión y despliegue de prácticas, ofreciendo una solución integral y eficiente que facilite tanto la interacción de los practicantes como el seguimiento de sus actividades.



Instalación y Configuración de Airflow (Máquina Local)

La instalación de Airflow que haremos será desarrollada en un entorno de una máquina local. Para ello deberán seguirse los siguientes pasos.

Prerrequisitos

Antes de comenzar, asegúrate de cumplir con los siguientes requisitos:

- Sistema operativo: Linux Windows con WSL2.

Pasos para la Instalación

1. Instalar airflow:

El paso pionero a la instalación y configuración de airflow es instalar el paquete que contiene el airflow con el siguiente comando en nuestro terminal de Linux

```
>>pip install apache-airflow
```

2.Agregar dependencias adicionales (Base de Datos):

este paso es necesario para dar soporte a bases de datos específicas, añade las dependencias correspondientes separadas por comas en caso de desear incluir otras tecnologías a futuro para el proyecto.

```
>>pip install apache-airflow[Postgres]
```

3.Inicialización de Airflow:

Antes de usar Airflow, es necesario configurar su entorno base y crear un usuario administrador. por lo tanto deberemos ejecutar dos comandos, uno en consecuencia de otro.

```
>>airflow db init
```

```
>>airflow users create \
  --username admin \
  --firstname Admin \
  --lastname User \
  --role Admin \
  --email admin@example.com \
  --password adminpassword
```

Nota. Cada dato es arbitrario de cada persona.

Al completar estos pasos, se generará automáticamente en el directorio `/home` una carpeta llamada **airflow**. Esta carpeta contendrá todas las configuraciones necesarias para ejecutar el programa y gestionar sus componentes.

Dentro de esta estructura, encontrarás una subcarpeta clave llamada **dags**. Este directorio es donde se crearán y almacenarán los flujos de trabajo (DAGs) que luego podrás desplegar y gestionar desde la interfaz web de Apache Airflow.

Esta organización estructurada facilita el desarrollo, mantenimiento y despliegue de flujos de trabajo, garantizando un entorno eficiente y ordenado para tus proyectos.



Como lanzar Apache Airflow

1. Iniciar el servidor web

Para acceder a la interfaz gráfica de Airflow, inicia el servidor web ejecutando en bash:

```
>>airflow webserver --port 8080
```

Propósito: Este comando lanza la interfaz web de Apache Airflow, que permite visualizar, gestionar y monitorear los DAGs.

- La opción `--port 8080` especifica el puerto en el que se ejecutará el servidor. Puedes cambiar este valor si el puerto 8080 ya está en uso.
- Por defecto, la interfaz estará disponible en <http://localhost:8080>.
- Cabe recalcar que para que apache airflow funcione correctamente no solo hay que lanzar la interfaz web, también hay que lanzar el siguiente comando.

2. Ejecutar el programador de tareas (scheduler)

Además del servidor web, es necesario ejecutar el programador de tareas con:

```
>>airflow scheduler
```

Propósito: El scheduler es el componente que asigna las tareas a los trabajadores y asegura que los flujos de trabajo (DAGs) se ejecuten según lo programado. Este servicio revisa periódicamente las tareas pendientes y las encola para su ejecución (Sin él no podríamos ver los DAGs que tenemos creados).

Instalación y Configuración de Airflow (Contenedor Docker)

Prerrequisitos

Antes de comenzar, asegúrate de cumplir con los siguientes requisitos:

- **Docker** instalado en tu sistema.
- **Docker Compose** instalado (versión 1.29 o superior).
- Sistema operativo: Linux, macOS o Windows con WSL2.

Pasos para la Instalación

1. Crear un archivo Docker Compose

Crea un archivo llamado **docker-compose.yaml** con la siguiente configuración básica para Apache Airflow: Este archivo se puede encontrar en la carpeta Docker del repositorio

2. Preparar el Entorno

Crea los directorios necesarios para los DAGs, logs y plugins:

```
>>mkdir -p dags logs plugins
```

Inicializa la base de datos de Airflow:

```
>>docker-compose up airflow-db -d
```

```
>>docker-compose run airflow-webserver airflow db init
```

Crea un usuario administrador para Airflow:

```
>>docker-compose run airflow-webserver airflow users create \
--username admin \
--firstname Admin \
--lastname User \
--role Admin \
--email admin@example.com \
--password adminpassword
```


3. Lanzar los Servicios

Para iniciar todos los servicios de Airflow, ejecuta:

```
>>docker-compose up -d
```

4. Acceso a la Interfaz de Airflow

La interfaz de usuario de Apache Airflow estará disponible en <http://localhost:8080>. Inicia sesión con las credenciales del usuario creado en el paso anterior.

5. Añadir DAGs

1. Coloca tus archivos de DAGs en el directorio **dags** que creaste anteriormente.
2. Los DAGs se cargarán automáticamente en la interfaz web de Airflow.

6. Detener los Servicios

Para detener todos los servicios:

```
>>docker-compose down
```

Instalación y Configuración de NextJS

Next.js es un framework de React que permite crear aplicaciones modernas con renderizado del lado del servidor (SSR), generación estática y otras funcionalidades avanzadas. Esta guía te llevará paso a paso para configurar Next.js en tu entorno de desarrollo.

Requisitos Previos

Tener Node.js instalado (versión 16 o superior). Puedes verificar la versión instalada con:

```
>>node -v
```

```
>>npm -v
```

1. Tener un gestor de paquetes como npm (incluido con Node.js) o yarn.
 2. Un editor de código, preferiblemente Visual Studio Code (VS Code).
-

Pasos para la Instalación

1. Configura tu Entorno

```
>>cd ruta/proyecto
```

2. Clonar Proyecto de Git

Usa el siguiente comando para inicializar un nuevo proyecto con Next.js:

```
>>Git Clone https://github.com/robleaustral/Lectura-Forms-Apach.Airflow.git
```

3. Navega al Directorio del Proyecto

Una vez creado el proyecto, accede al directorio del mismo:

```
>>cd /web
```

4. Inicia el Servidor de Desarrollo (Guardar para más tarde)

```
>>npm run dev
```

Instalación y Configuración de Postgres

Pasos para la Instalación

1. Actualizar el Sistema

Antes de comenzar, asegúrate de que tu sistema esté actualizado ejecutando el siguiente comando:

```
>>sudo apt update && sudo apt upgrade -y
```

Este comando actualiza los paquetes de tu sistema.

2. Instalar PostgreSQL

Para instalar PostgreSQL en un sistema basado en Debian/Ubuntu, ejecuta el siguiente comando:

```
>>sudo apt install postgresql postgresql-contrib -y
```

Este comando instalará PostgreSQL y algunas herramientas adicionales que pueden ser útiles.

3. Configurar la Contraseña del Usuario

Una vez instalado PostgreSQL, configura una contraseña para el usuario `postgres` con el siguiente comando:

```
>>sudo -u postgres psql -c "ALTER USER postgres WITH PASSWORD 'password';"
```

Asegúrate de reemplazar `password` con la contraseña que quieras asignar.

4. Verificar la Instalación

Puedes verificar que PostgreSQL esté correctamente instalado y que la clave haya sido configurada ejecutando:

```
>>psql -version
```

Esto te mostrará la versión de PostgreSQL instalada.

5. Iniciar el Servicio de PostgreSQL

El servicio de PostgreSQL debe iniciarse automáticamente después de la instalación, pero puedes asegurarte de que esté en ejecución con el siguiente comando:

```
>>sudo systemctl start postgresql
```

6. Habilitar PostgreSQL al Inicio

Para asegurarte de que PostgreSQL se inicie automáticamente al arrancar el sistema, ejecuta:

```
>>sudo systemctl enable postgresql
```

Notas extra:

- Ahora PostgreSQL está instalado y el usuario **postgres** tiene una contraseña configurada.
- Para detener el servicio de PostgreSQL, ejecuta:

```
>>sudo systemctl stop postgresql
```

Sumado a esto, podemos instalar un gestor de bases de datos como DBeaver para hacer más fácil el uso de estas. (OPCIONAL)

1. Descargar DBeaver

Primero, descarga el paquete de instalación de DBeaver ejecutando el siguiente comando en tu terminal:

```
>>wget https://dbeaver.io/files/dbeaver-ce_latest_amd64.deb
```

Este comando descargará el archivo de instalación más reciente de DBeaver para S.O 64 bits.

2. Instalar DBeaver

Luego instalar el paquete con el comando siguiente:

```
>>sudo apt install ./dbeaver-ce_latest_amd64.deb
```

3. Lanzar DBeaver

Finalmente, para lanzar DBeaver, ejecuta el siguiente comando en la terminal:

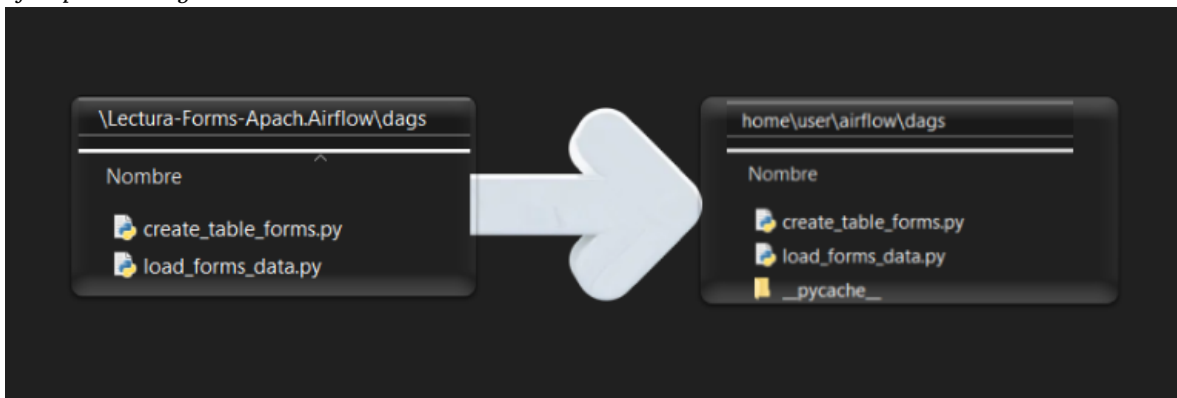
```
>>dbeaver
```

Configuración del Proyecto de Prácticas

Airflow

Ahora que hemos configurado todas las tecnologías necesarias, estamos listos para ejecutar correctamente el gestor de prácticas de la UCh. El primer paso será organizar los DAGS correspondientes a nuestro proyecto. Estos se encuentran almacenados en el repositorio, dentro de la carpeta **/dags**. Lo único que necesitamos hacer es copiar todo el contenido de esta carpeta y pegarlo en el directorio **/home/user/airflow/dags**: en caso de ser el entorno virtual, se deberá llevar a la carpeta dags del mismo.

Figura 1.
Ejemplo De Migración DAGS



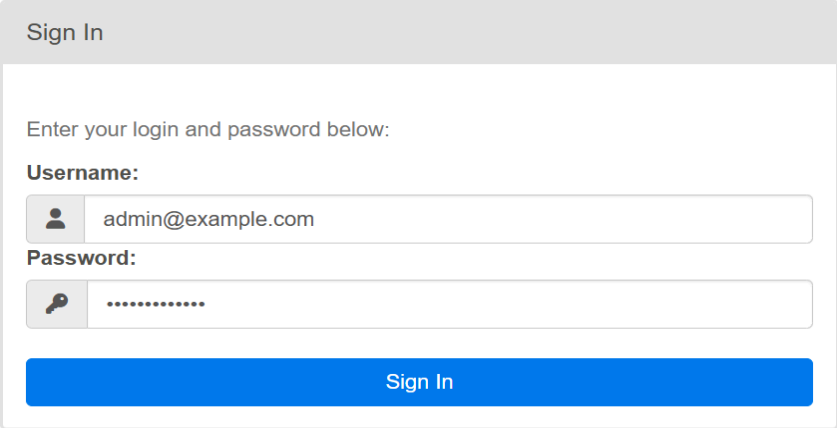
Nota. La Figura muestra cómo debería ser la migración de las carpetas.

Antes de lanzar, recordar abrir el archivo txt alojado en la carpeta raíz del repositorio llamado **"requirements_python.txt"** para descargar las librerías que usarán los DAGS

Luego de ello, ya podremos lanzar Airflow, tal como se detalla en la sección **"Resumen de la ejecución conjunta"** en la página 7.

Una vez lanzado apache airflow debería desplegarse un login el cual nos pedirá nuestras credenciales que definimos en **"Instalación y Configuración de Airflow"** en la página 6.

Imagen 1
Página de Login

A login form titled "Sign In". It contains a text input for "Username:" with the value "admin@example.com" and a password input for "Password:" with masked characters. A blue "Sign In" button is at the bottom.

Sign In

Enter your login and password below:

Username:

admin@example.com

Password:

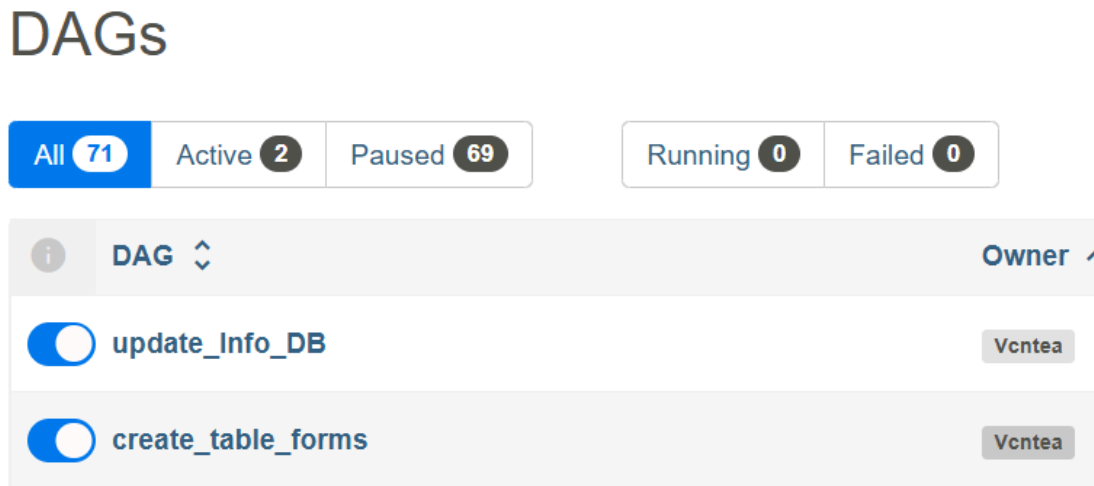
.....

Sign In

Nota. Vista principal del login y credenciales.

Una vez dentro de airflow podremos visualizar todos los DAGS existentes, Nosotros buscaremos los DAGS llamados: update_info_DB y create_table_forms. Y los marcaremos como activos.

Imagen 2
Menú de gestión de DAGS Apache

A screenshot of the Apache Airflow DAGs management interface. It shows a header with "DAGs" and a filter bar with buttons for "All 71", "Active 2", "Paused 69", "Running 0", and "Failed 0". Below is a table with columns "DAG" and "Owner". Two DAGs are listed: "update_Info_DB" and "create_table_forms", both owned by "Vcntea".

DAGs

All 71 Active 2 Paused 69 Running 0 Failed 0

DAG	Owner
<input checked="" type="checkbox"/> update_Info_DB	Vcntea
<input checked="" type="checkbox"/> create_table_forms	Vcntea

Nota. Vista principal del menú de inicio.

En caso de querer añadir algún DAG, estos se tendrán que crear dentro de la carpeta **airflow/dags**. Para ello, basta con crear un archivo Python con el flujo de trabajo (DAG) correspondiente, utilizando la estructura y las tareas que desees automatizar.

Antes de Ejecutar los DAGS. primero deberán Establecerse unas variables de Airflow, las cuales se pueden añadir desde el menú de admin/variables. O puedes hacerlo pegando estos comando en el bash.

```
>>airflow variables set FRIENDLY_NAMES '['marca_temporal', 'tipo_practica',  
"nombre_contacto", "cargo_contacto", "correo_contacto", "telefono_contacto",  
"nombre_empresa", "sitio_web_empresa", "unidad_empresa", "fechas_practica", "modalidad",  
"sede_practica", "regimen_trabajo", "labores", "beneficios", "requisitos_especiales"]'
```

Siguiente comando:

```
>>airflow variables set CSV_URL  
'https://docs.google.com/spreadsheets/d/1eVkJZ-zrQc7oRAXOIhizLi8F4hcX9bG03loDFqfv7Q  
TU/export?format=csv'
```

Estas variables se pueden modificar en menu/variables en el navBar con la finalidad de agregar en caso de que se añadan columnas en el forms o cambiar el URL de ser necesario.

Y por último establecer la base de datos, se puede realizar mediante el comando bash que está a continuación o manualmente desde admin/connections en la navBar

```
>>airflow connections add 'postgres_default' \  
--conn-type 'postgres' \  
--conn-host 'localhost' \  
--conn-schema 'Nombre_base_datos' \  
--conn-login 'postgres' \  
--conn-password 'password' \  
--conn-port '5432'
```

Ahora podemos ejecutar los DAGS, primero ejecutaremos el DAG llamado:

“create_table_forms”: Este nos dará Inicializará la base de datos con una tabla llamada forms, la cual tendrá la estructura del formulario de Google.

Luego ejecutaremos:

“update_info_DB”: El cual se irá ejecutando diariamente para que pueda alimentar la base de datos con la información correspondiente.

Con todo esto, ya tenemos finalizado todo de momento con AirFlow y nuestra base de datos funcionando, ahora nos resta configurar nuestro entorno en NEXT JS.

NextJs

Primero navegaremos a la ruta de donde se encuentra la página web.

```
>>cd ruta/web
```

Ahora instalamos dependencias

```
>>npm i
```

Ahora inicializamos nuestro ORM llamado prisma

```
>>npx prisma init
```

De esto se nos generará un archivo dotenv. en donde deberemos poner nuestra base de datos con el siguiente formato.

```
DATABASE_URL="postgresql://<postgres_usuario>:<contraseña>@host:puerto/nombre_base_datos?schema=public"
Ejemplo
DATABASE_URL="postgresql://postgres:password@localhost:8059/postgres?schema=public"
```

Seguido de esto ingresamos el comando

```
>>npx prisma db pull
```

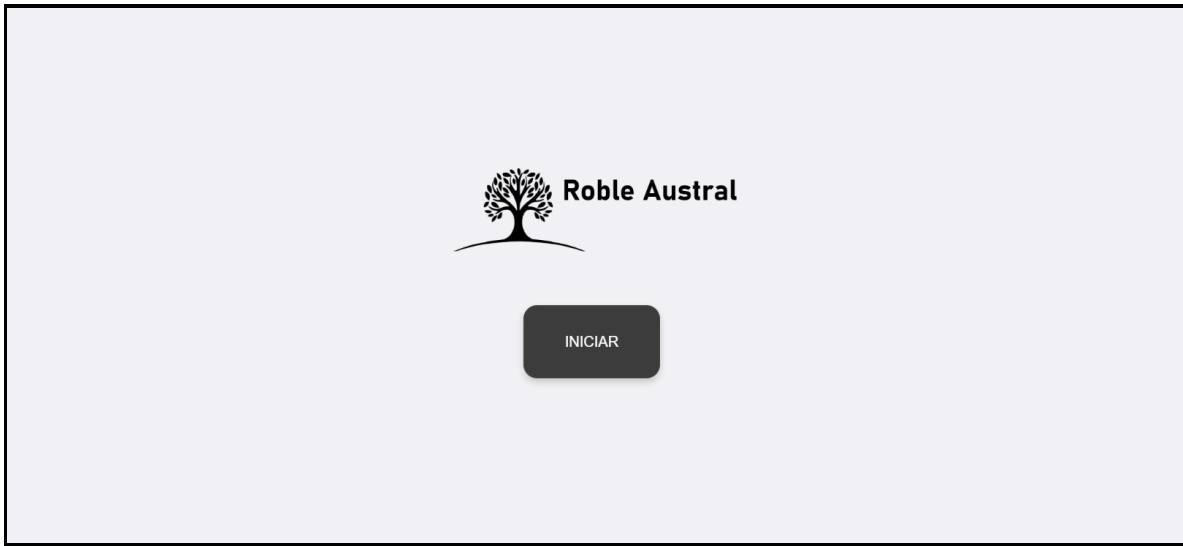
Para traer la información de la estructura de la base de datos a nuestro ORM Prisma. Con ello podremos ahora lanzar el programa y se realizarán correctamente las consultas de backend.

Luego, podremos lanzar el servicio web, tal como se detalla en la sección "**Instalación y Configuración de Next JS**" en la página 8.

Proyecto de Prácticas En Funcionamiento

Imagen 3

Menú de Inicio para prácticas.



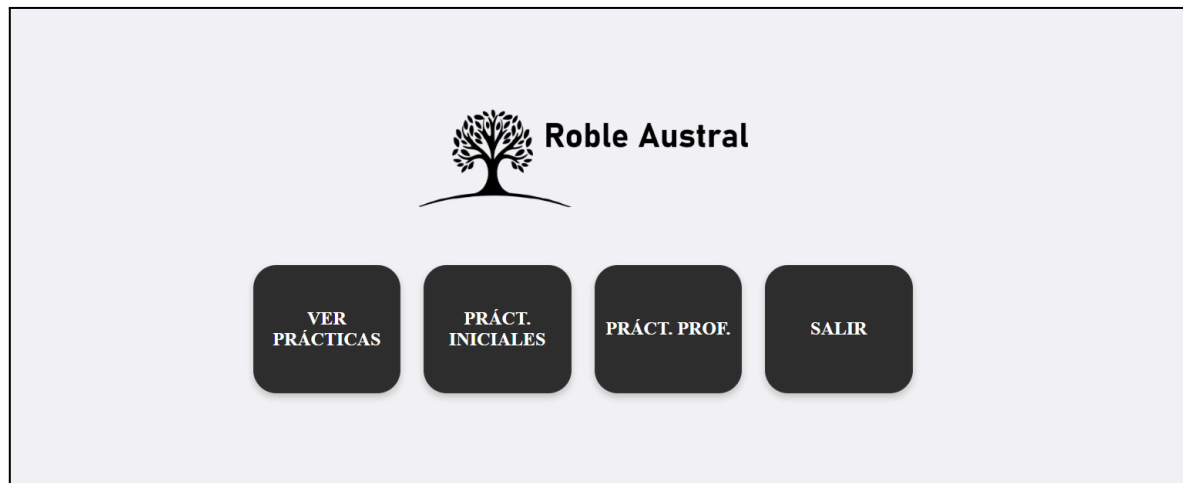
Nota. Vista principal del menú de inicio.

En la página principal con dirección: <http://localhost:3000/>

se podrá ver el botón de inicio para dar pie al menú que muestra cada opción relacionada con las prácticas.

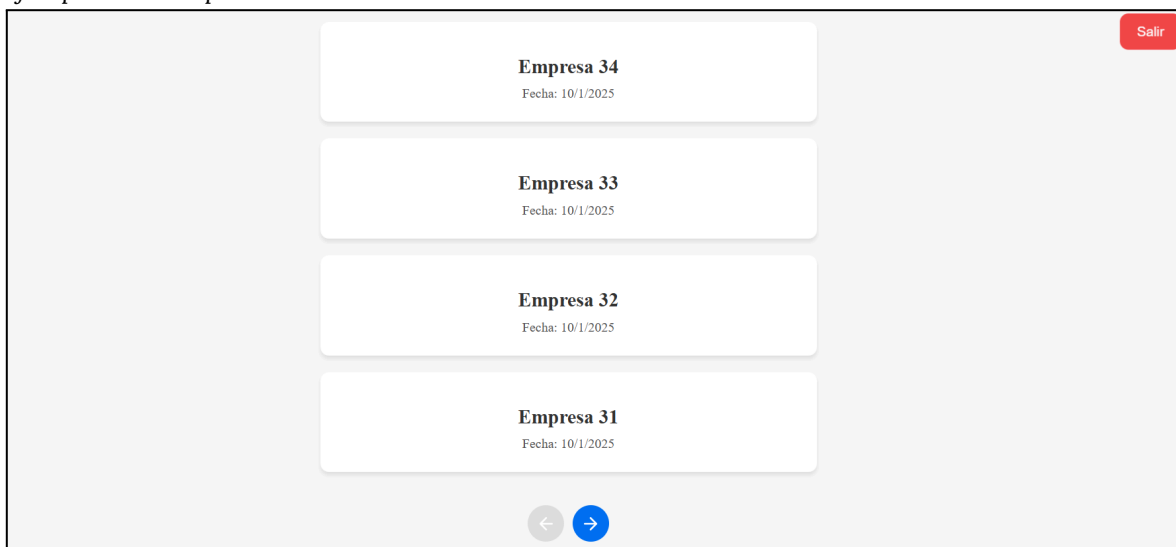
Imagen 4

Menú de Prácticas



Nota. Vista del menú con las diferentes prácticas.

Imagen 5
Ejemplo vista de prácticas



Nota. Vista del menú con ejemplos de práctica

Imagen 6
Ejemplo vista de práctica individual

The screenshot displays a form titled "EMPRESA 33" in bold black text at the top center. Below the title, there are ten horizontal input fields, each with a label on the left and a value on the right. The labels and values are: "Tipo de práctica:" with "Inicial"; "Contacto:" with "Contacto 33 (Cargo 33)"; "Correo:" with "contacto33@empresa.com"; "Teléfono:" with "9632097005"; "Fechas:" with "2024-12-08 - 2024-12-09"; "Modalidad:" with "presencial"; "Régimen de trabajo:" with "full time (6 horas diarias o más)"; "Labores:" with "Labores 33"; "Beneficios:" with "Beneficios 33"; and "Requisitos especiales:" with "Requisitos 33". At the bottom center, there is a blue button labeled "Volver".

Nota. se puede apreciar el detalle de cada variable que es pedida en el forms pero para los practicantes.

Creación de Dags

Apache Airflow es una plataforma de orquestación de flujos de trabajo que permite programar, monitorear y gestionar tareas de manera eficiente y flexible. Un *DAG* (Directed Acyclic Graph) es el concepto central en Airflow, y se utiliza para definir un conjunto de tareas que deben ejecutarse en una secuencia específica. A continuación, se presentan los conceptos clave que se deben tener en cuenta al crear un DAG en Airflow.

1. ¿Qué es un DAG?

Un DAG (Directed Acyclic Graph) es un grafo dirigido acíclico que representa un conjunto de tareas y sus dependencias. Cada nodo del grafo es una tarea (o *task*), y las aristas representan las dependencias entre estas tareas. En Airflow, el DAG se utiliza para especificar cómo y cuándo deben ejecutarse las tareas.

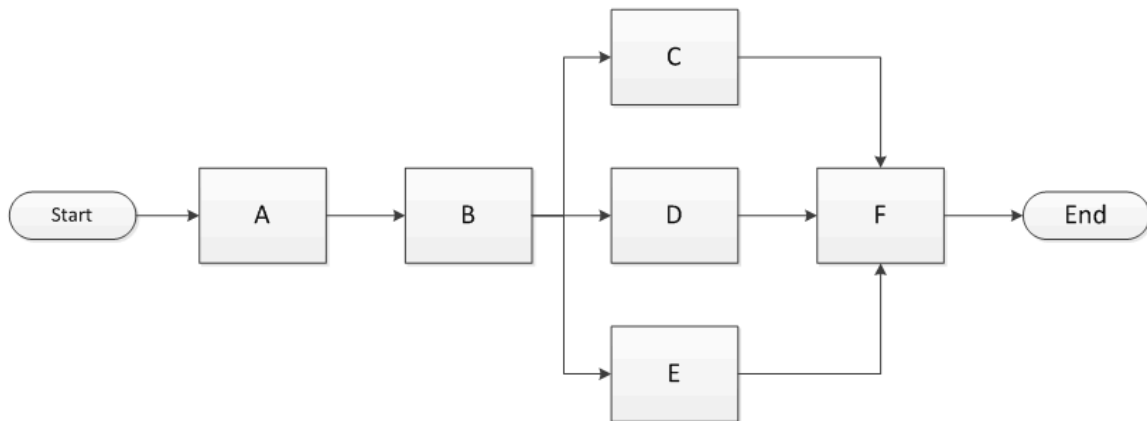
Características de un DAG:

- **Dirigido:** Las tareas tienen un orden de ejecución definido, lo que implica que el flujo de trabajo tiene un comienzo y un final.
- **Acíclico:** No se permiten ciclos dentro del grafo. Es decir, no puede haber una tarea que dependa indirectamente de sí misma.
- **Orientado:** El flujo de tareas se ejecuta en una secuencia definida, según las dependencias entre ellas.

El DAG define la estructura general de un flujo de trabajo, incluyendo la programación de ejecución y las dependencias entre las tareas.

Figura 2

Ejemplo De Flujo de tareas



Nota. Podemos ver como hay una dependencia entre tareas, que es lo que busca representar al explicar que es un DAG.

2. Estructura de un DAG en Airflow

En Airflow, un DAG se define mediante un archivo Python, normalmente ubicado en el directorio `dags/` del proyecto. A continuación, se muestra un ejemplo básico de un DAG:

Imagen 7

Ejemplo DAG Inicial

```
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator
from datetime import datetime

# Definir la función que ejecutará una tarea
def my_task():
    print("Ejecutando tarea")

# Crear un DAG
dag = DAG(
    'my_first_dag',
    description='Mi primer DAG de ejemplo',
    schedule_interval='@daily', # Ejecutar una vez al día
    start_date=datetime(2023, 1, 1),
    catchup=False, # No ejecutar tareas pasadas
)

# Definir tareas
task_1 = DummyOperator(task_id='start', dag=dag)
task_2 = PythonOperator(task_id='execute_task', python_callable=my_task, dag=dag)
task_3 = DummyOperator(task_id='end', dag=dag)

# Definir las dependencias entre las tareas
task_1 >> task_2 >> task_3
```

Nota. Se muestra como se crea un orden de tareas y se les da dependencias

En este ejemplo:

- **DummyOperator** se usa como una tarea de marcador de posición.
- **PythonOperator** ejecuta una función de Python como tarea.

El flujo de trabajo está definido por las dependencias entre las tareas: **task_1** debe completarse antes de **task_2**, y **task_2** debe completarse antes de **task_3**.

3. Componentes Principales de un DAG

- **Tareas (Tasks):** Son las unidades de trabajo dentro de un DAG. Cada tarea puede ejecutar una función específica, ejecutar un script, o incluso interactuar con otros sistemas.
- **Operadores (Operators):** Son las clases que definen lo que hace cada tarea. Algunos ejemplos comunes de operadores son:
 - **DummyOperator:** Un operador que no hace nada, útil para crear tareas de marcador de posición o dividir flujos.
 - **PythonOperator:** Ejecuta funciones de Python.
 - **BashOperator:** Ejecuta comandos Bash.
 - **BranchOperator:** Permite ramificar el flujo de trabajo según una condición.
- **Dependencias entre tareas:** Las tareas se conectan entre sí mediante dependencias. Estas dependencias definen el orden en que se ejecutan las tareas.

4. Secuencia de las Tareas

La secuencia de las tareas en un DAG es fundamental para garantizar que las dependencias entre ellas se respeten. Existen varias formas de definir estas dependencias en Airflow:

- **>> (Shift Right):** Este operador establece que una tarea debe ejecutarse después de otra.
- **<< (Shift Left):** Este operador establece que una tarea debe ejecutarse antes de otra.
- **Métodos `set_upstream()` y `set_downstream()`:** Estos métodos también permiten definir dependencias, aunque no son tan comunes como los operadores de desplazamiento (>> y <<).
- `task_1 >> task_2` # task_2 depende de task_1

5. Variables de Airflow

Las variables en Airflow permiten almacenar valores que pueden ser utilizados por los DAGs y las tareas. Las variables pueden contener configuraciones, credenciales u otros datos que se necesiten en tiempo de ejecución. Se pueden definir desde la interfaz de usuario de Airflow o mediante código.

Imagen 8

Importación de variables.

```
from airflow.models import Variable

# Obtener una variable
my_variable = Variable.get("my_variable_name")

# Establecer una variable
Variable.set("my_variable_name", "valor")
```

6. Parámetros del DAG

Cuando se crea un DAG, existen varios parámetros que permiten controlar su comportamiento:

- **schedule_interval:** Define la frecuencia con la que el DAG debe ejecutarse. Puede ser una expresión cron, un string de intervalo (como `@daily`), o `None` si el DAG no se ejecuta de forma periódica.
- **start_date:** Especifica la fecha y hora en que el DAG debe comenzar a ejecutarse.
- **end_date:** Opcionalmente, se puede especificar cuándo debe finalizar el DAG.
- **catchup:** Si está configurado en **True**, Airflow ejecutará las instancias de DAG que faltaron entre el **start_date** y el momento actual.
- **retries:** Número de intentos de reejecución que Airflow debe intentar si una tarea falla.
- **retry_delay:** Especifica cuánto tiempo debe esperar Airflow entre los intentos de reejecución.

7. Manejo de Errores y Retries

Airflow permite configurar el comportamiento de las tareas en caso de error mediante las siguientes opciones:

- **retries:** Define el número de intentos que se deben hacer antes de que la tarea se marque como fallida.
- **retry_delay:** Establece el tiempo de espera entre cada intento de reejecución.
- **on_failure_callback y on_success_callback:** Son funciones que se ejecutan en caso de que una tarea falle o tenga éxito, respectivamente. Estas funciones pueden ser útiles para notificaciones o registros personalizados.

8. SubDAGs

Un SubDAG es un DAG dentro de otro DAG. Es útil cuando necesitas organizar tareas en un grupo lógico, permitiendo reutilizar parte del flujo de trabajo.

Imagen 9

Ejemplo SubDAG o DAG Anidado.

```
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.subdag_operator import SubDagOperator

def subdag(parent_dag_name, child_dag_name, args):
    dag_subdag = DAG(
        dag_id=child_dag_name,
        default_args=args,
        schedule_interval="@daily",
    )

    with dag_subdag:
        task_1 = DummyOperator(task_id="task_1", dag=dag_subdag)
        task_2 = DummyOperator(task_id="task_2", dag=dag_subdag)

        task_1 >> task_2

    return dag_subdag
```

9. DAGs Dinámicos y Parametrización

Un DAG dinámico es aquel que genera tareas de manera programática, en lugar de definir todas las tareas estáticamente en el código. Esto puede ser útil cuando el flujo de trabajo debe adaptarse a entradas externas, como la creación de tareas según una lista de archivos, fechas o parámetros.

Imagen 10

Ejemplo DAG Dinámico.

```
for i in range(5):
    task = PythonOperator(
        task_id=f'task_{i}',
        python_callable=my_task,
        op_args=[i],
        dag=dag
    )
```

10. Consideraciones de Rendimiento y Escalabilidad

- **Parallelismo:** El parámetro `dag_concurrency` se utiliza para limitar el número máximo de tareas que se pueden ejecutar en paralelo dentro de un DAG.
- **Executor:** Airflow puede usar diferentes ejecutores (como `SequentialExecutor`, `LocalExecutor`, `CeleryExecutor`, etc.) que controlan cómo se distribuyen y ejecutan las tareas en los recursos disponibles.
- **Pool de recursos:** Airflow permite crear pools para limitar la cantidad de recursos (como CPU, memoria) que se asignan a un conjunto de tareas.