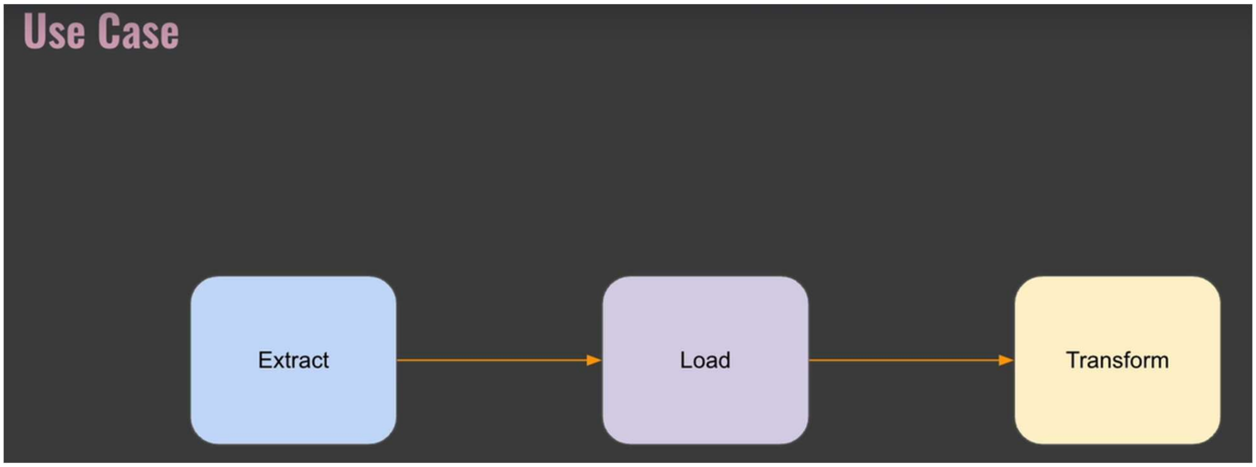
**5. Why airflow, why you need it?**

Let me show you this.

Let's imagine that you have the following data pipeline with three

tasks extract, load and transform, and it runs every day at 10:00 PM.

Very simple.

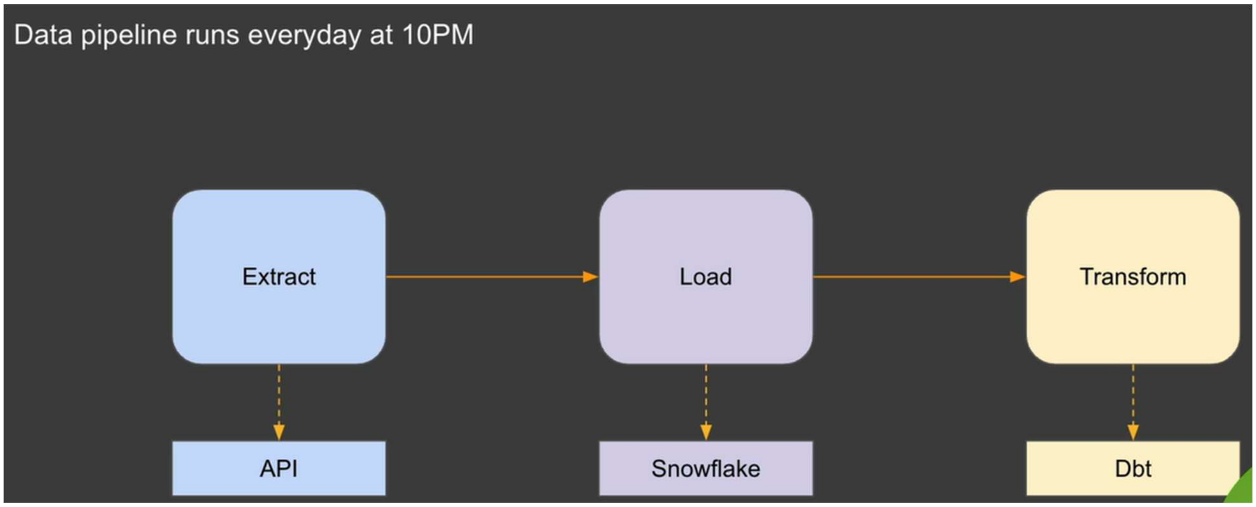


Use Case

Obviously at every step you are going to interact with an external

tool or an external system, for example, an API for extract.

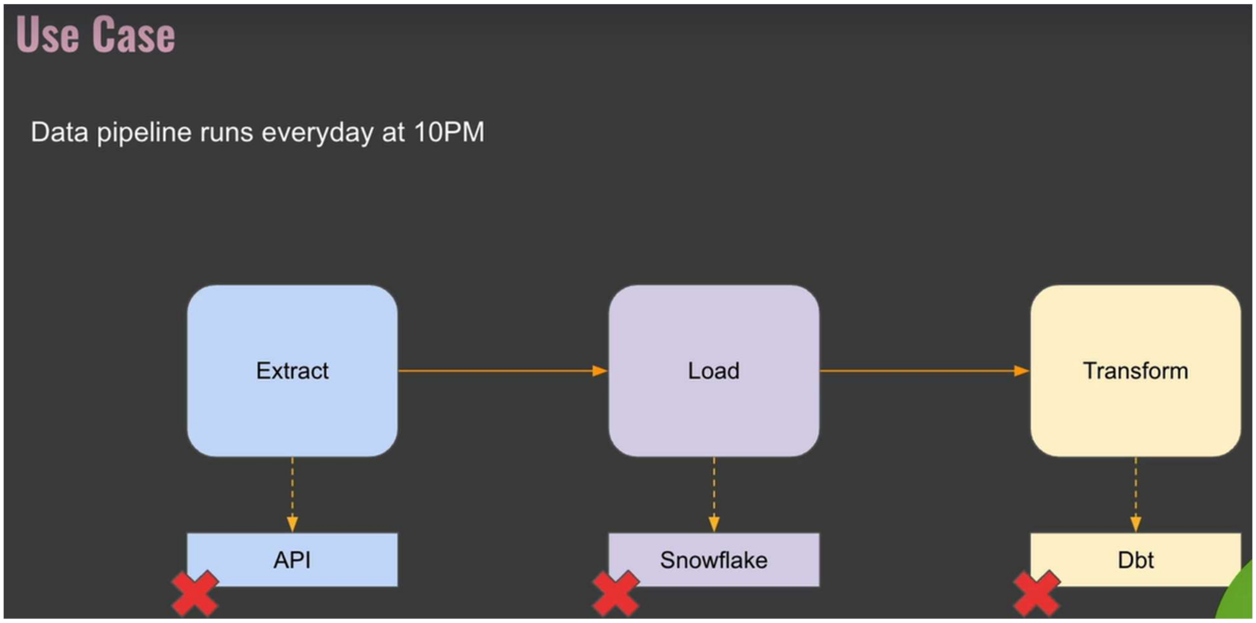
Snowflake for load, and DBT for transform.



Data pipeline runs everyday at 10PM

Now, what if the API is not available anymore?

Or what if you have a error in Snowflake or what if you made a mistake in your transformations with DBT.

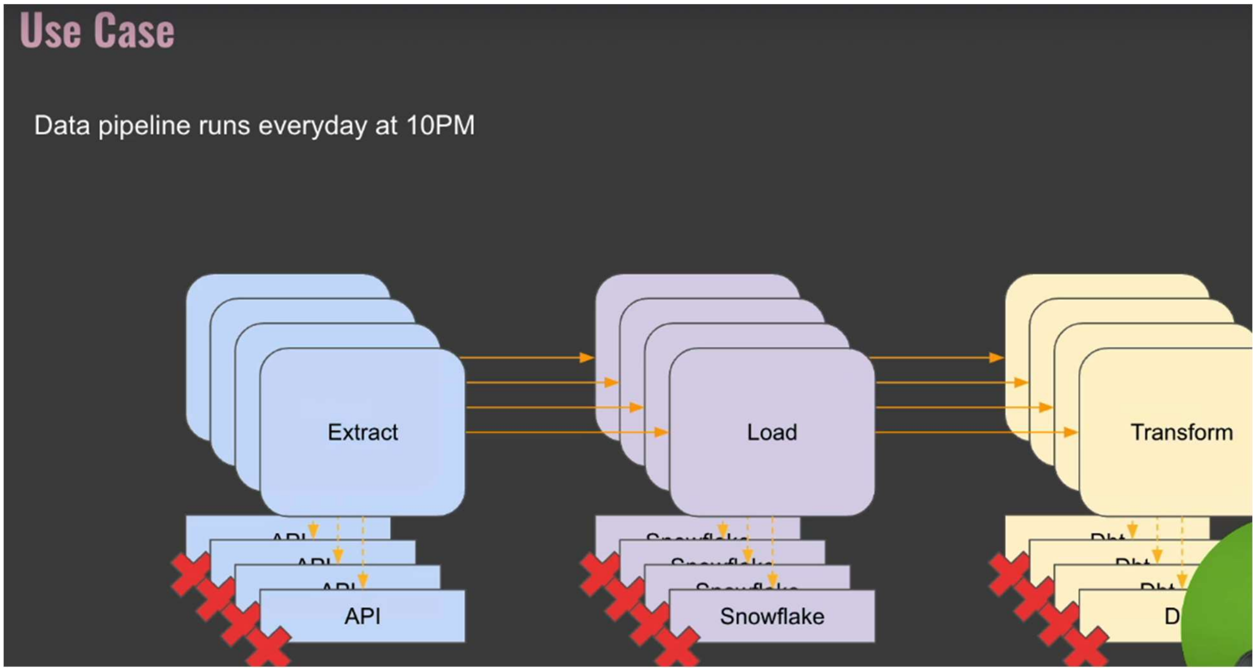


Use Case

Data pipeline runs everyday at 10PM

As you can see at every step, you can end up with a failure and you need to have a tool that manages this.

Also,what if instead of having one data pipeline, you have hundreds of data pipelines.



Use Case

Data pipeline runs everyday at 10PM

As you can imagine, it's gonna be a nightmare for you, and

this is why you need Airflow.

With Airflow you are able to manage failures automatically,

even if you have hundreds of data pipelines and millions of tasks.

So if you want to make your life easier, well, you are at the right place.

### 6. What is Airflow?

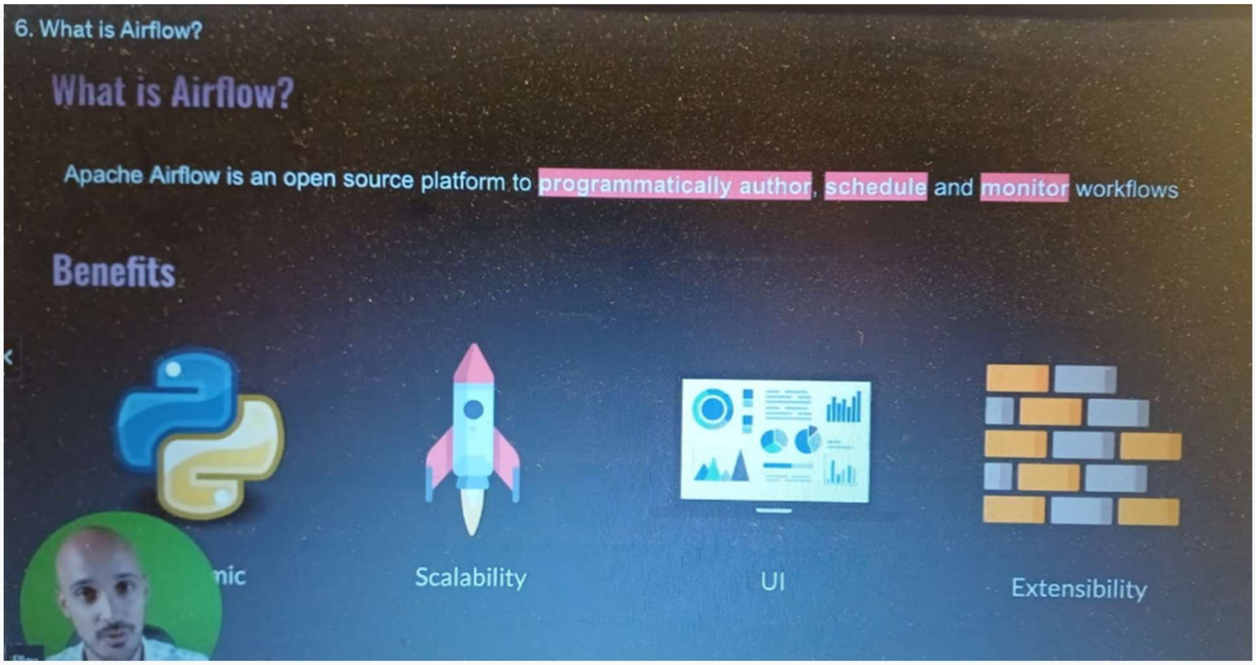
Think of Airflow as a cook, following a recipe where the
recipe is your data pipeline.
You have to follow this recipe by putting the right ingredients with
the right quantity in the right order.
And that's exactly what Airflow allows you.
Instead of having your recipe, you have your data pipeline and your ingredients
are your tasks, and you want to make sure that they are executed in the right order.
That being said, let me give you some technical terms to
explain exactly what is Airflow.

First thing first, Airflow is an open source platform, an open source
project to **programmatically, author, schedule** and **monitor workflows**.
The workflows are your data pipelines.
There are some benefits of using Airflow and the first one is, everything is
coded in Python or everything is **dynamic**.
Okay.
You don't have to use a static language like XML, if you used Oozie in the past.
With Airflow everything is coded in Python and so you can benefit from that language.
It is also very easy to use.

Next, it is very **scalable**.
Indeed, you can execute as many tasks as you want with Airflow.

Also, you have access to a beautiful **user interface**, which
is nice to have to monitor your tasks and your data pipelines.
Then last but not least, Airflow is truly **extensible**.

So you can add your own plugins, your own functionalities to Airflow.



6. What is Airflow?

What is Airflow?

Apache Airflow is an open source platform to programmatically author, schedule and monitor workflows

Benefits

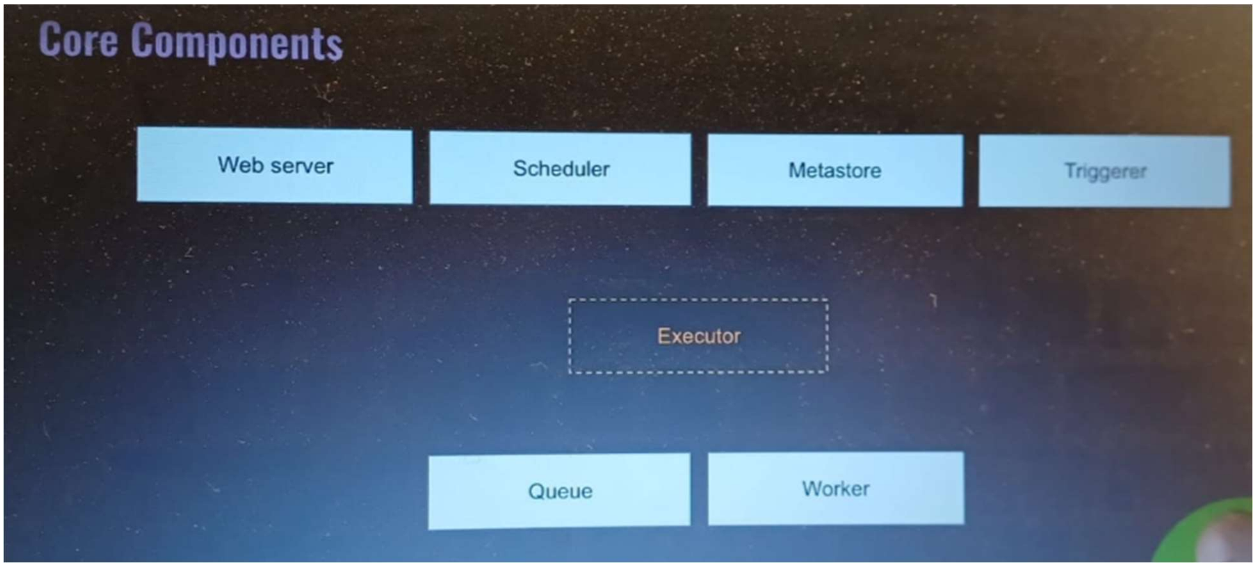
You don't have to wait for anyone to add anything on Airflow.
You can do it by yourself.

So keep in mind that Airflow is an orchestrator.
It allows you to execute your tasks in the right way, in the right order at the right time.

### 7. Core Components

Airflow brings core components, and it's always good to know what they are.
And the first core component is the **web server**.
The web server is a flask Python web server that allows you to access the user interface.
Also, you have the **scheduler**.
The scheduler is very critical because it is in charge of scheduling your tasks, your data pipelines.
So you want to make sure that your scheduler works otherwise, nothing work.
Then you have the metadatabase or the **metastore**.
The metadatabase is nothing more than a database that is compatible with SQL alchemy.
For example, Postgres, MySQL, Oracle DB, SQL server, and so on.
In this database, you will have metadata related to your data pipelines, your tasks, airflow users, and so on, that will be stored.
In addition to those components, you have another one, which is the triggerer.
I won't dive into the details here, but the triggerer allows you to run a specific kind of tasks that we are going to see later in the course.
That being said, those four core components are the components that you will see as soon as you run Airflow for the first time.

In addition, you have a concept called **executor** and an executor defines how and on which support your tasks are executed.
For example, if you have a Kubernetes cluster, you want to execute your tasks on this Kubernetes cluster, you will use the KubernetesExecutor.
If you want to execute your tasks in a Celery cluster, Celery is a Python framework to execute multiple tasks on multiple machines, you will use the CeleryExecutor.
Keep in mind that the executor doesn't execute any tasks.
Now, if you use the CeleryExecutor for example, you will have two additional core components, a **queue**, and a **worker**.
In a queue your tasks will be pushed in it in order to execute them in the right order, and the worker is where your tasks are effectively executed.
Now, keep in mind that with any executor, you always have a queue in order to execute your tasks in the correct order.
And if you don't have a worker, you have some sub processes where your tasks are executed or some PODs if you use Kubernetes.
So that's all the core components that you need to remember.



Core Components

Web server

Scheduler

Metastore

Triggerer

Executor

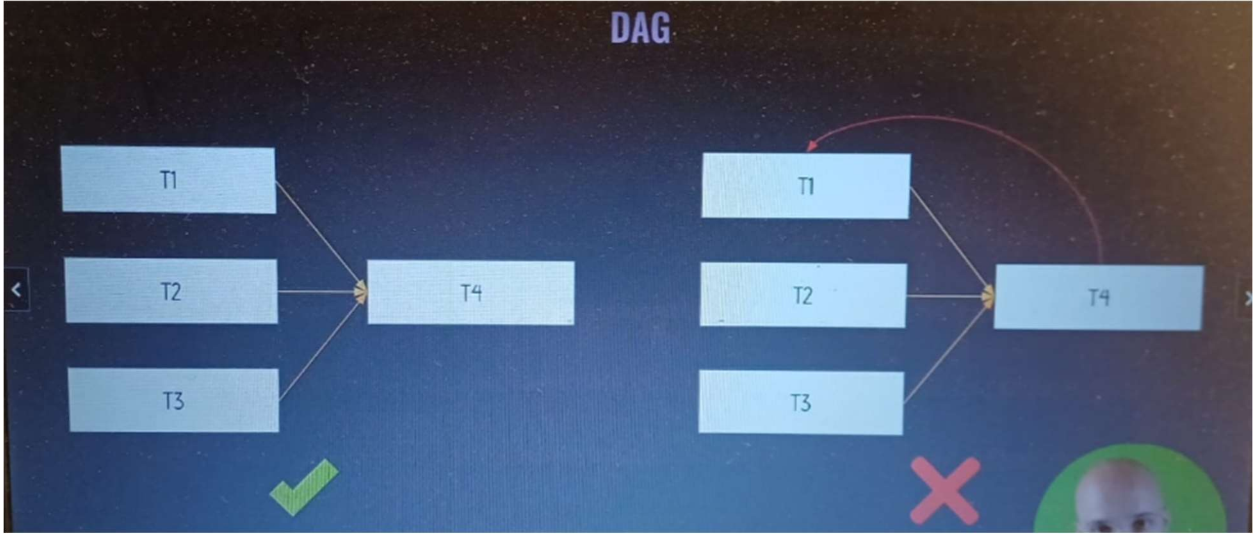
Queue

Worker

## 8. Core Concepts

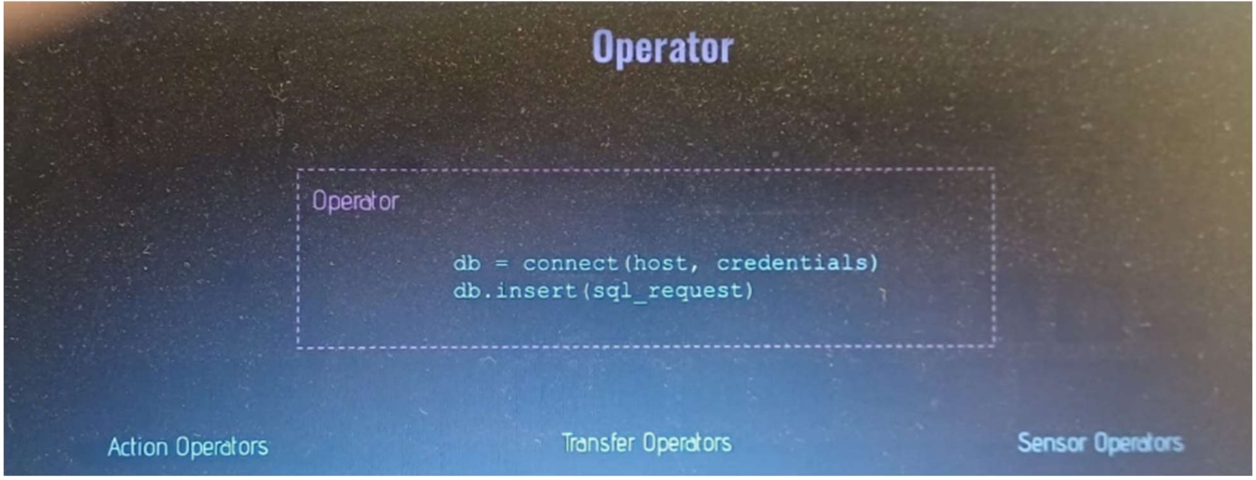
Let me give you the core concepts of Airflow that you need to know.
And the first one is the concept of **DAG.**

What is a DAG?
A DAG means directed acyclic graph, and it's nothing more than a graph with nodes, directed edges and no cycles.
For example, here you have a DAG.
T4, T1, T2, T3 are the nodes, the tasks.
You also have directed dependencies or edges.
T4 depends on T1, T2, and T3.
Then last but not least, you don't have any cycle.
That's what you can see here.
This, is not a DAG.
Why?
Because there is a cycle.
That's what you can see with T4 depends on T1 and T1 depends on T4.
This doesn't work in Airflow and you will get an error if you do that.



Now the second concept to know is the concept of **Operator**.
What is an Operator?
It's nothing more than a task.
So think of an operator as a way to encapsulate what you want to do.
For example, you want to connect to a database and execute a SQL request, you will use an operator.
There are three types of operators and the first one is **action operators**.
An action operator executes something.
For example, the **PythonOperator** executes a Python function,
the **BashOperator** executes a bash command.
Also, you have the **transfer operators** that may disappear at some point, but

basically they allow you to transfer data from a point A to point B.
For example, you want to transfer data from MySQL to Redshift.
Then last but not least, you have the **sensors** and the sensors are very useful because they allow you to wait for something to happen before moving to the next task.
For example, you are waiting for files you will use the FileSensor.
We will get back at those because they're extremely useful.



Operator

Operator

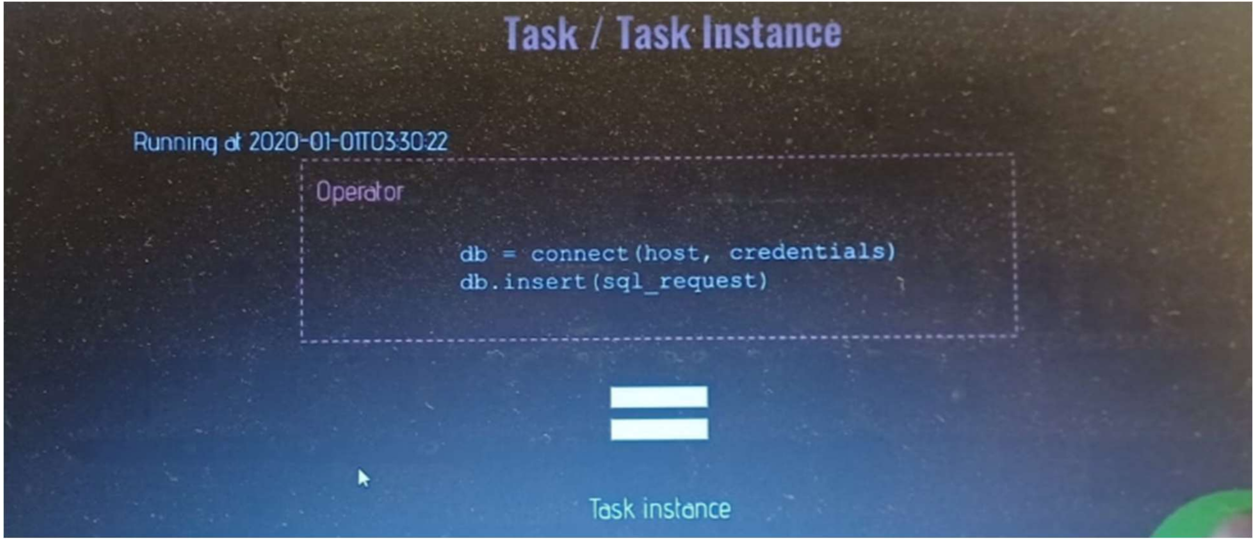
```
db = connect(host, credentials)
db.insert(sql_request)
```

Action Operators

Transfer Operators

Sensor Operators

Now, you know, the concept of DAG and Operator, there is another concept, which is the concept of **Task** and **Task Instance**.
Again, an Operator is a task and when you run an operator, so a task, you get a task instance.
It is as simple as that.



Task / Task Instance

Running at 2020-01-01T03:30:22

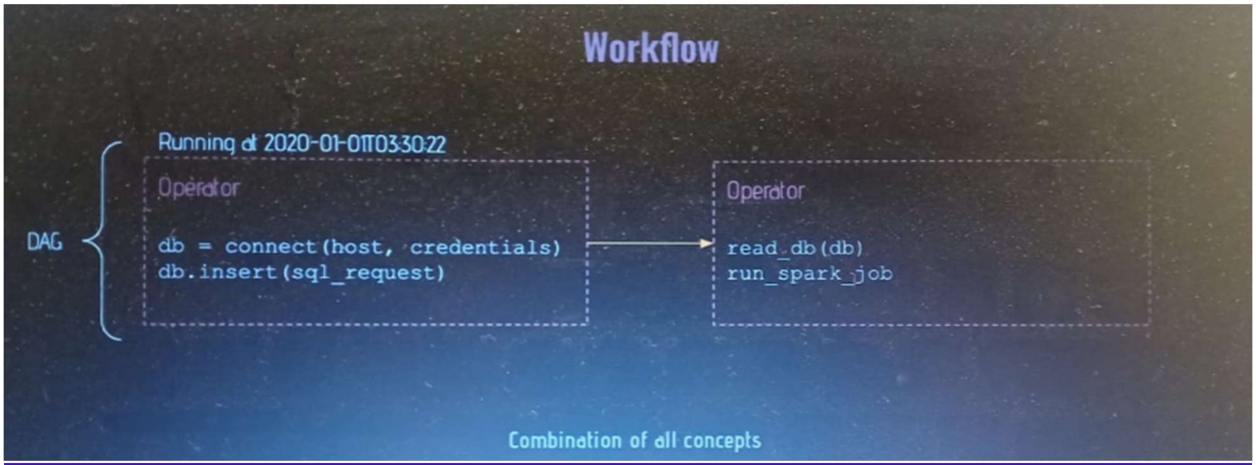
Operator

```
db = connect(host, credentials)
db.insert(sql_request)
```

=

Task instance

Finally, if you group all of those concepts together, you end up with the concept of **workflow** where you have your Operators, directed dependencies, and so a DAG, and that gives you the concept of workflow.
It's the combination of all concepts.



## Workflow

Running at 2020-01-01T03:30:22

DAG

Operator

```
db = connect(host, credentials)
db.insert(sql_request)
```

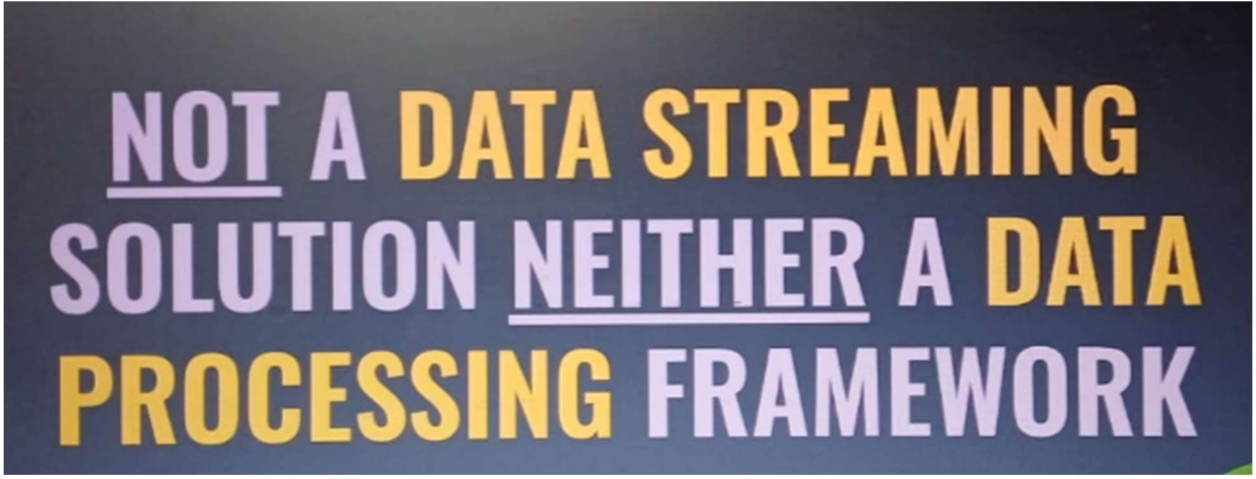
Operator

```
read_db(db)
run_spark_job
```

Combination of all concepts

## 9. Airflow is not…

You know, what is Airflow, but you don't know what it is not, and this is what you're going to discover right now.
So Airflow is not a data streaming solution, neither
a data processing framework.

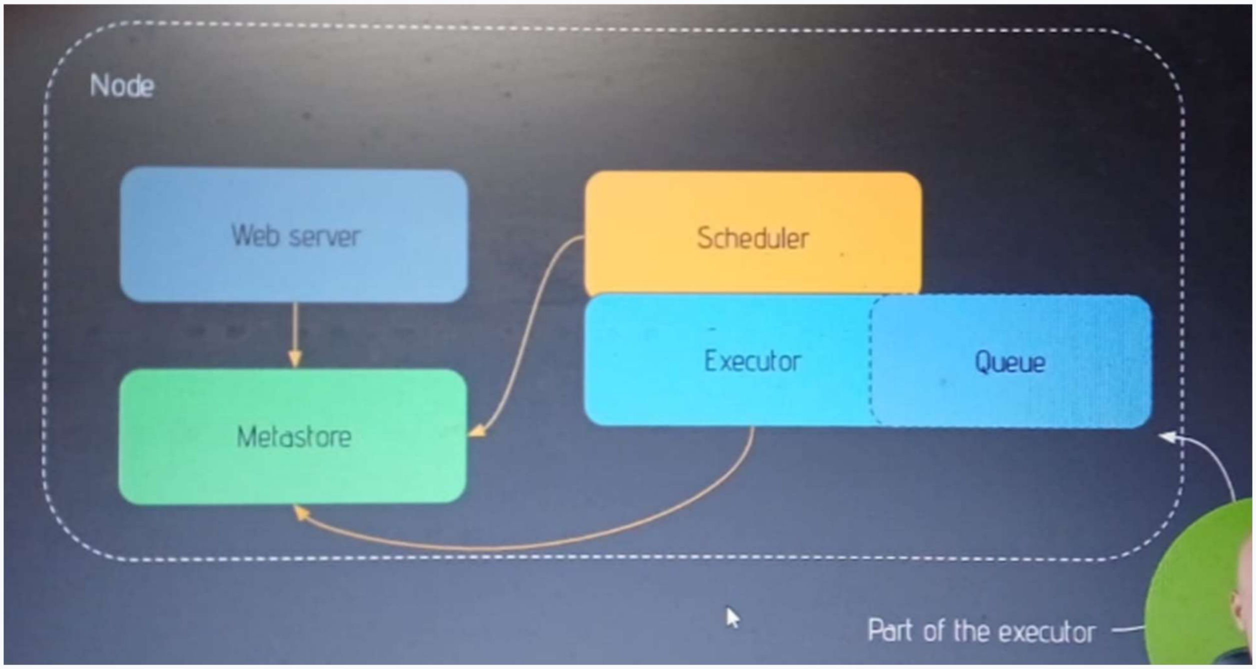


NOT A DATA STREAMING SOLUTION NEITHER A DATA PROCESSING FRAMEWORK

What does it mean?
It means that you are not going to schedule your data pipelines every second.
That doesn't work with Airflow.
Maybe it'll be possible in the future, but for now don't use
Airflow as a streaming solution.
Also, you shouldn't use it as a data processing framework like Spark
because Airflow doesn't expect that you process your data in Airflow,
in your tasks, in your operators.
Instead, you should use Airflow as a way to trigger the tool
that will process your data.
For example, you have the **SparkSubmitJobOperator**, you could use that
operator to process terabytes of data.
Again, that may change in the future.
You can still process data in Airflow, but be very careful because it is
not meant to do that, and so you can end up with memory overflow errors.
That being said, keep in mind, Airflow is an orchestrator and
it is the best orchestrator.

### 10. Single Node Architecture

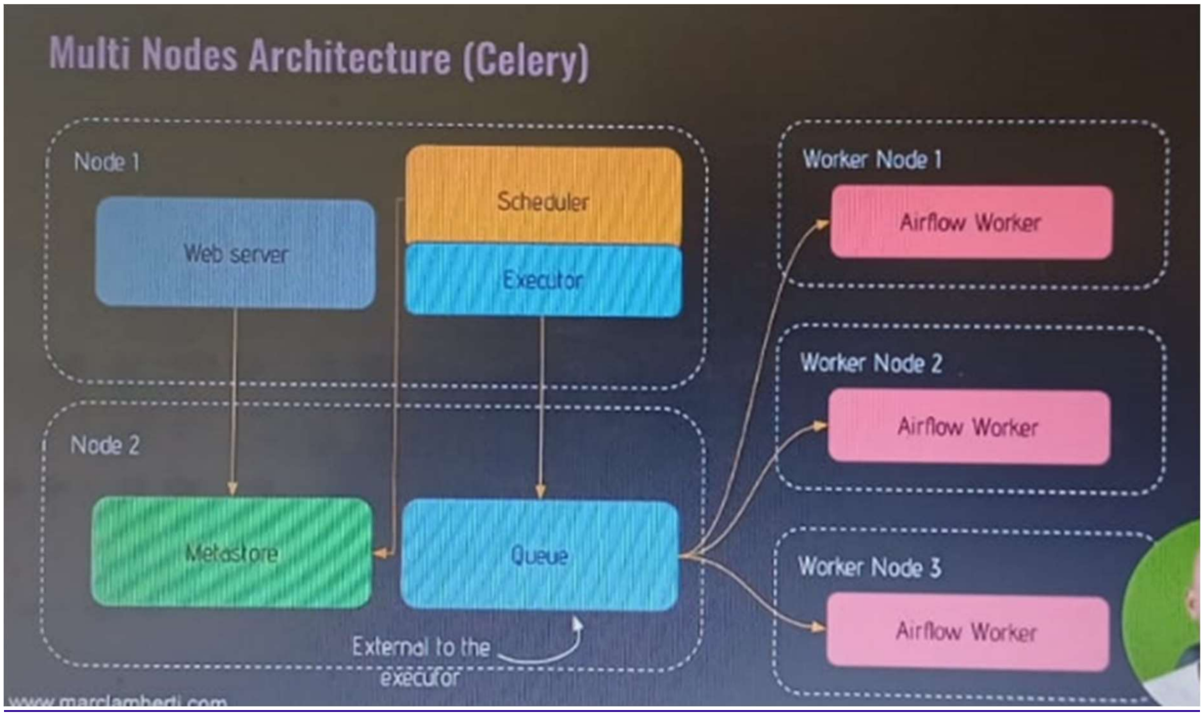
In Airflow, there are two architectures that are very common, and I would
like to show you the first one, which is the single node architecture.
And there is a good chance that you will start with that one because it
is the easiest one to deploy Airflow.
So what do you have?
Well, first thing first you have your machine or a node where the
**web server**, the metadatabase, or the **metastore**, as well as the **scheduler**
are running.
And as you can see, there is the **executor**, which is part of the scheduler.
Remember that the executor doesn't execute a task, but it defines how, and
on which system your tasks are executed.
In this case, your tasks are executed in processes on a single machine.
Next, you have the **web server** that communicates with the
database and obviously the **scheduler** as well as the **executor**
communicate with the metadatabase as well.
Remember that the metadatabase allows to exchange data between the different
components of Airflow, and there is also a **queue** which allows you to
execute the tasks in the correct order.
There is always a queue, regardless of the executor you use.
If you run Airflow for the first time, this is the architecture that you
will end up with and you will have the SequentialExecutor or the LocalExecutor as
we are going to see later in the course.



## 11. Multi Node Architecture

To run Airflow in production, you are not going to stay with
a single node architecture.
Indeed, you want to make sure that you don't have a single point of failure.
You want to make sure that your architecture is highly available and
you want to make sure that you're able to deal with the workload, with the
number of tasks that you want to execute,
and for that you need to use the multi nodes architecture.

In this example, we use Celery, but that works with Kubernetes as well.
So what do you have here?

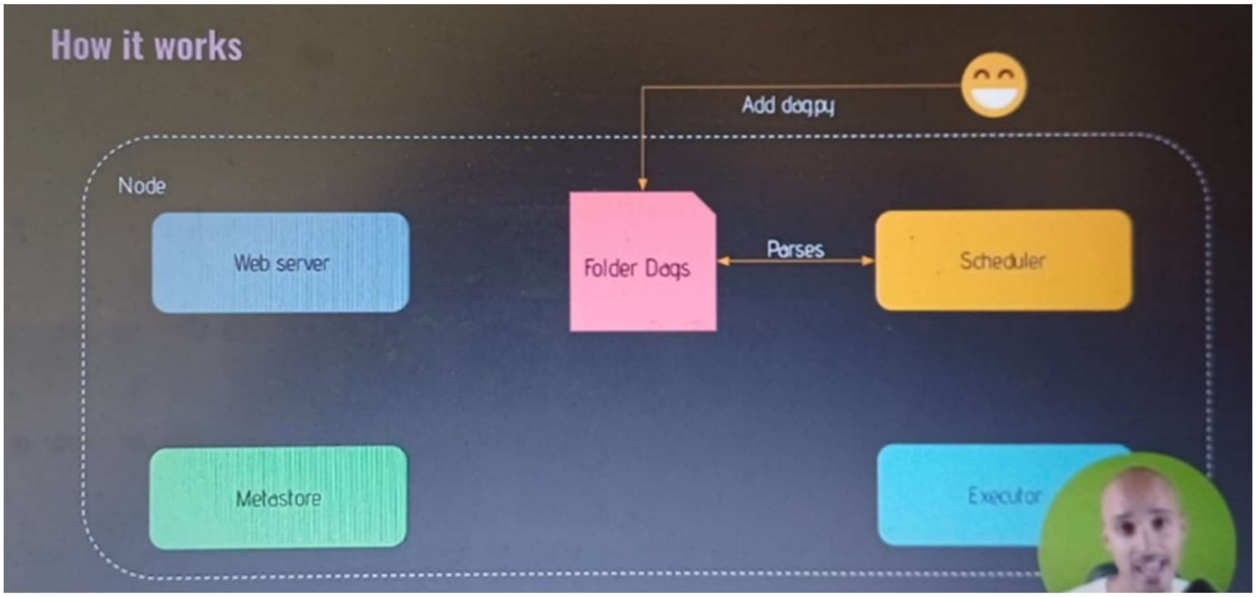


Well, first you have one node where the Web server and the Scheduler,
as well as the Executor, which is part of the Scheduler are.
In addition, you may have another node, which is where you will find the
meta database, as well as the queue.
In this case with the Celery executor, the queue is external to the executor.
Okay.
Keep in mind, that in this case for the Celery executor, you
will use Rabbit MQ or Reddis.
Now, in addition to those two machines, you have three additional machines where
you are going to execute your tasks, and we call those machines Worker nodes.
In every node, you have an Airflow Worker that you run with the
command *airflow celery worker*.
Next, as with the single node architecture, still the Scheduler

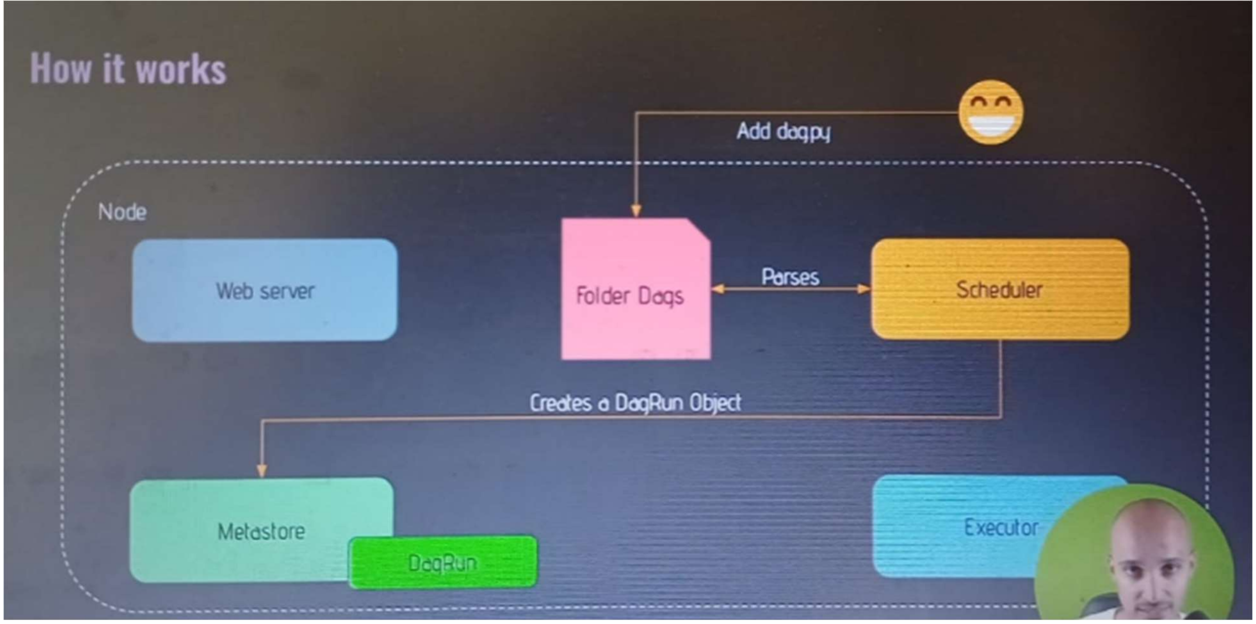
and the Web Server exchange data with the metadatabase and the Executor sends tasks to execute to the Queue in order to execute them in the right order, and finally, the Workers, the Airflow Workers pull the tasks from the queue and execute the tasks.
With this architecture,
if you need more resources to execute more tasks, you just need to add a new Airflow Worker on a new machine.
Also keep in mind that you should have at least two Schedulers as well as two Web servers, maybe a Load balancer in front of your web servers to deal with the number of requests on the Airflow UI, as well as PGBouncer to deal with the number of connections that will be made to your meta database.

## 12. How does it work?

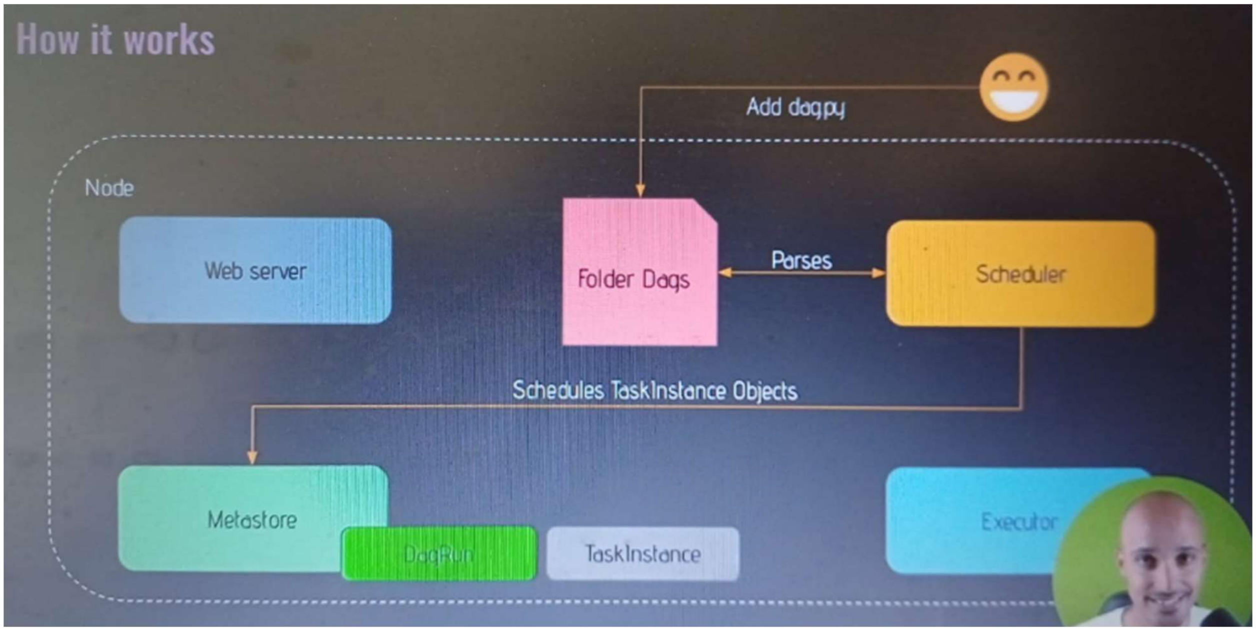
To debug your tasks and your data pipelines, you need to understand what's going on when the Scheduler triggers your tasks.
Let me show you this.
So you have one node with the components, the Web server, the Meta database, the Scheduler, the Executor, and the folder dags.
First you create a new DAG, dag.py and you put that file into the folder DAGs.
Next, the Scheduler parses this folder dags every five minutes by default to detect new DAGs.
So you may need to wait up to five minutes before getting your DAG on the Airflow UI.
Next, whenever you apply a modification to that DAG you may need to wait up to 30 seconds before getting your modification.
So keep in mind when you add a new DAG, the Scheduler parses every five minutes for new DAGs and for existing DAGs, the Scheduler parses for modifications every 30 seconds.



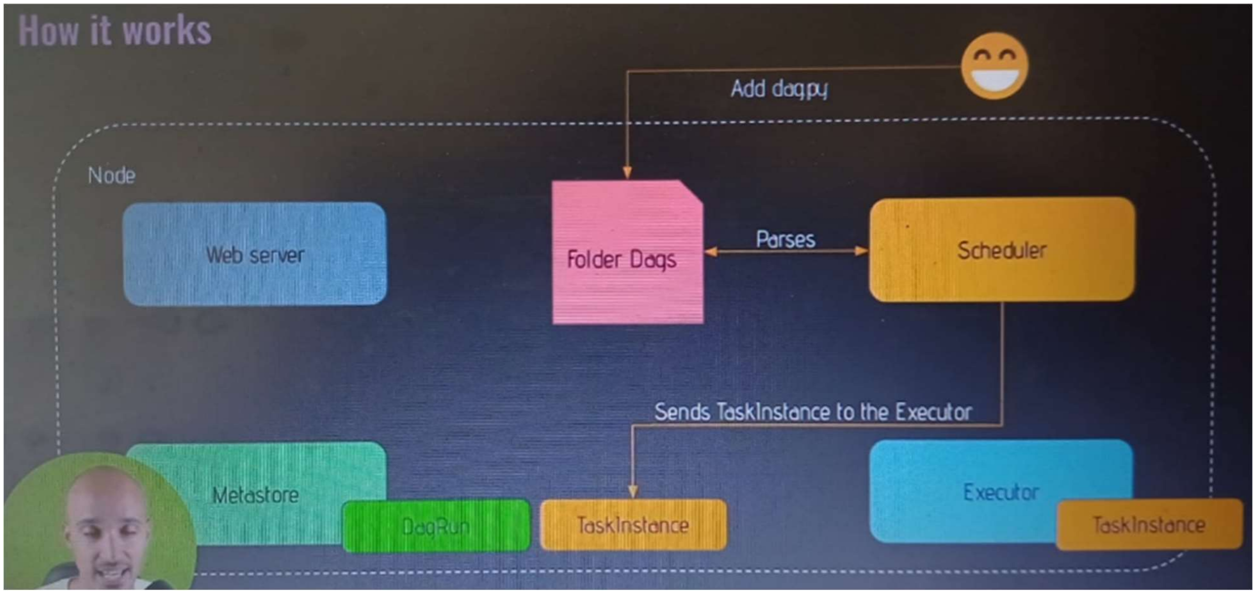
Next, the Scheduler runs the DAG, and for that, it creates a DAG Run object with the state Running.



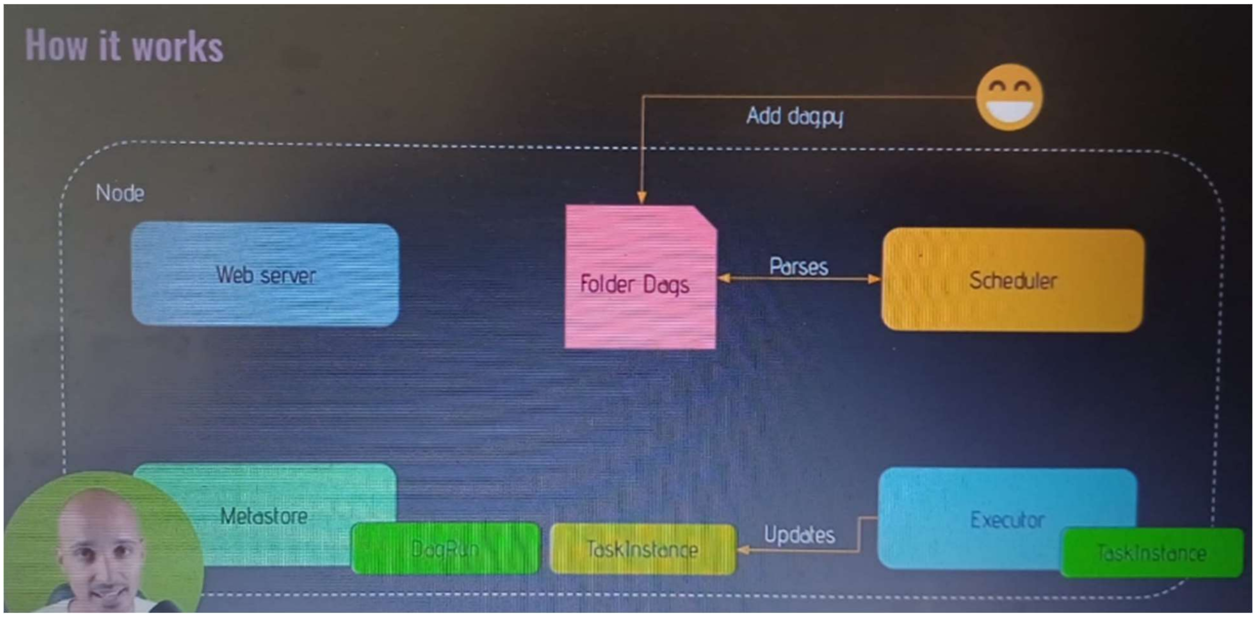
Then it takes the first task to execute and that task becomes a task instance object.

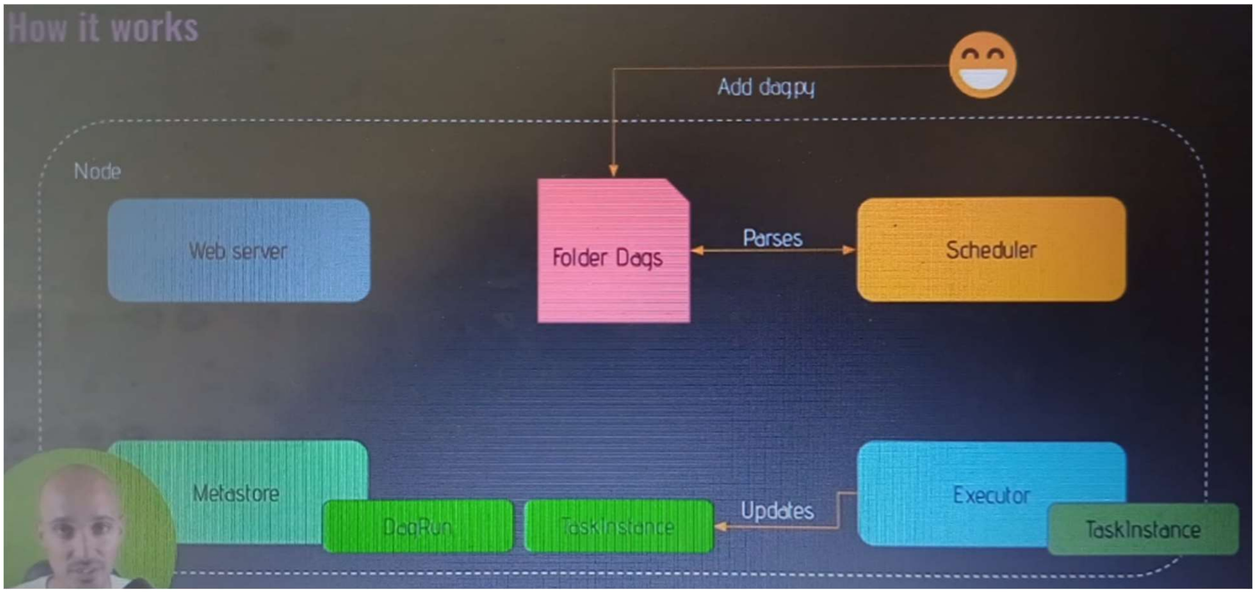


The task instance object has the state None and then Scheduled.
After that the Scheduler sends the task instance object into
the Queue of the Executor.

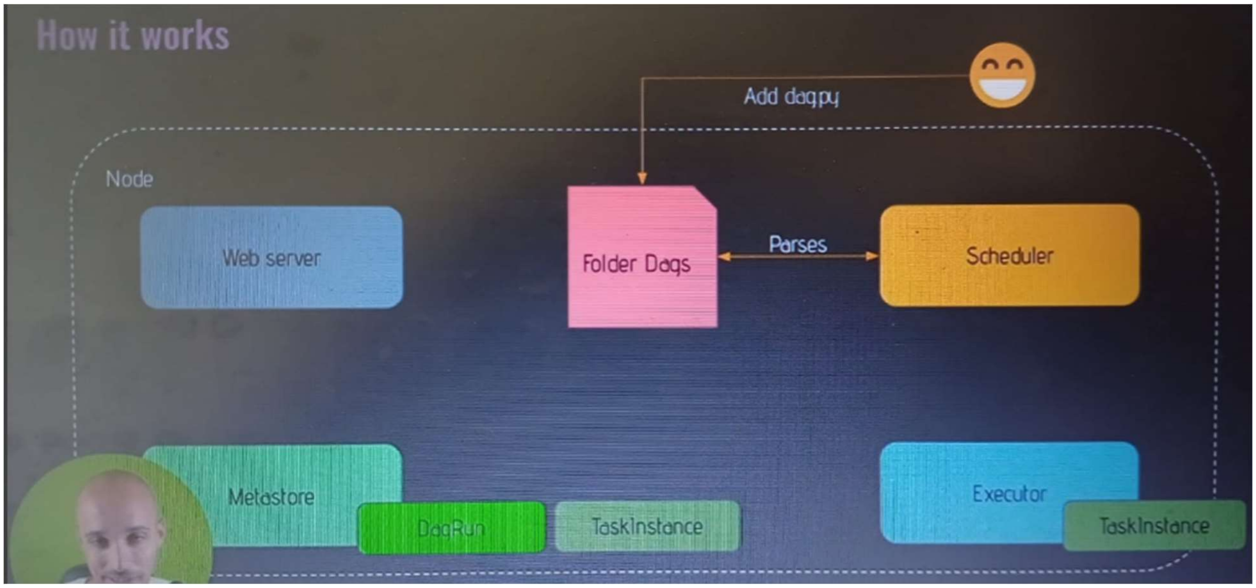


Now the state of the task is Queued and the Executor creates a sub process
to run the task, and now the task instance object has the state Running.

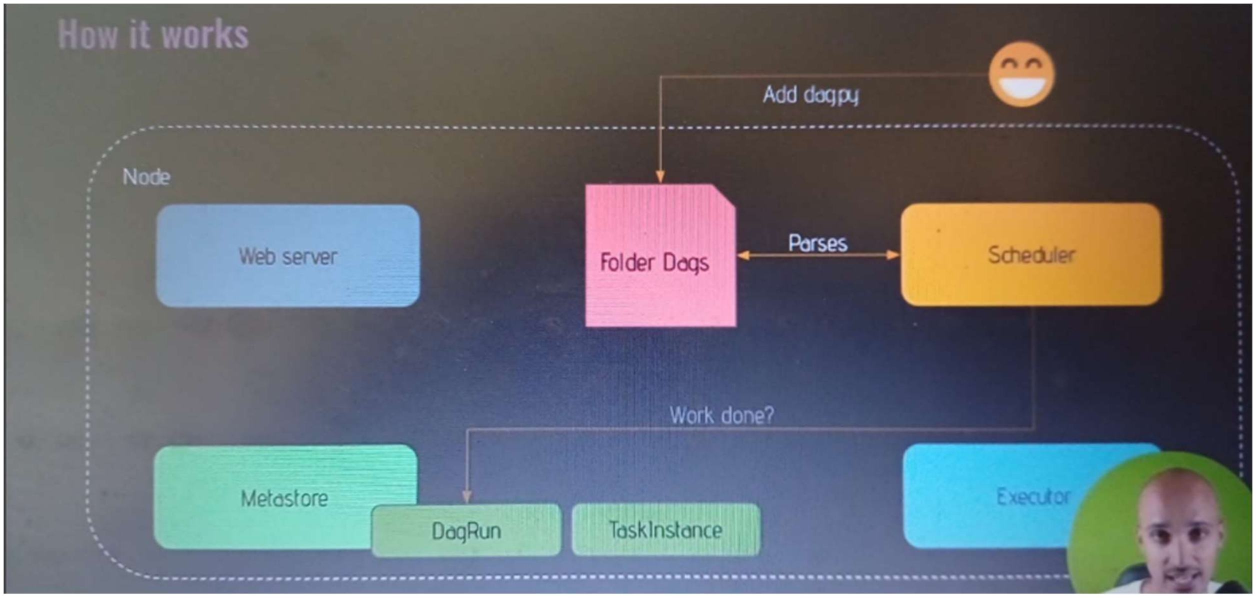




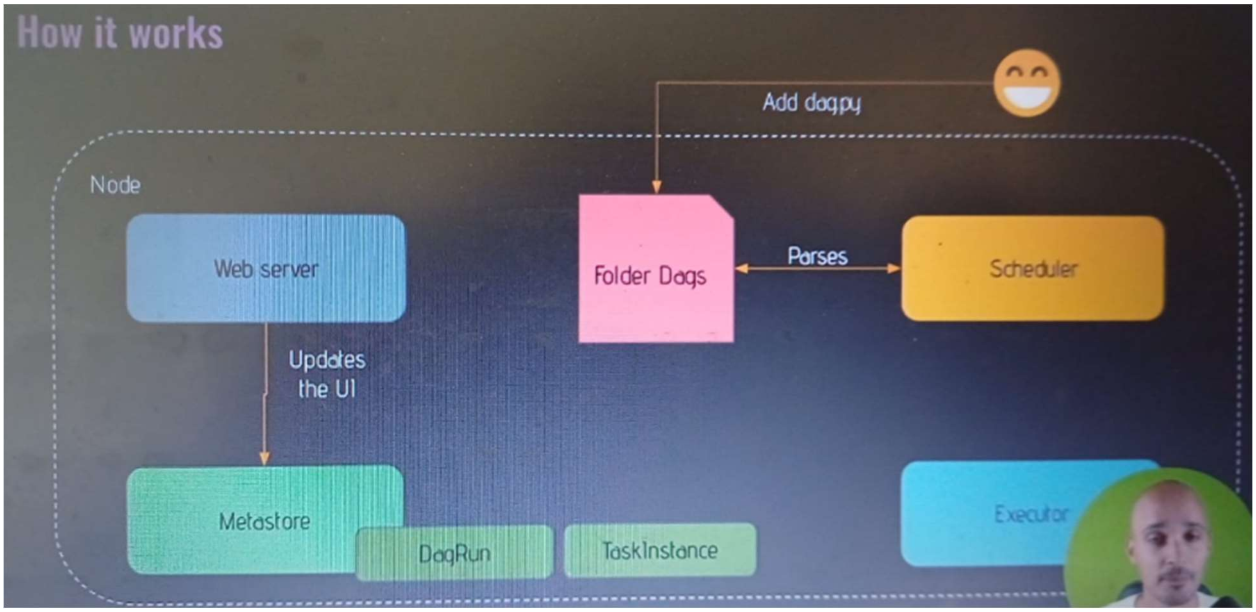
Once the task is done, the state of the task is Success or Failed.
It depends.



And the Scheduler checks, if there is no tasks to execute.



If the DAG is done in that case, the DAG Run has the state Success.



And basically you can update the Airflow UI to check the states of both the DAG Run and the task instances of that DAG Run.

So keep in mind when you run a DAG, that DAG has a DAG Run with the state **Queued**, then . **Success** or **Failed**.
Next, when a task runs that task becomes a task instance object with first, the state **None** then **Scheduled**, then **Queued** and then . **Success** are **Failed**.
There are other states, but that's the most important ones.