

## Relatório 18 - Pipelines de Dados II - Airflow

Guilherme Loan Schneider

### Descrição da atividade

#### 5. Distributing Apache Airflow

O SequentialExecutor é o executor padrão do Apache Airflow em instalações iniciais, especialmente quando se utiliza o banco de dados SQLite. Ele é o executor mais simples e limitado, executando apenas uma tarefa por vez, de forma sequencial. Isso o torna adequado apenas para ambientes de desenvolvimento muito básicos, prototipagem ou aprendizado.

Uma de suas principais limitações é justamente essa execução única e o fato de que o SQLite não lida bem com múltiplos acessos concorrentes, o que inviabiliza seu uso com outros executores mais robustos que oferecem paralelismo. Portanto, é recomendado utilizar apenas para testes e não para uso em produção.

Já o LocalExecutor é um executor mais avançado que permite a execução paralela de várias tarefas utilizando múltiplos processos na mesma máquina. Para funcionar corretamente, ele exige o uso de um banco de dados relacional, como o PostgreSQL, pois o SQLite não é compatível com operações concorrentes entre processos.

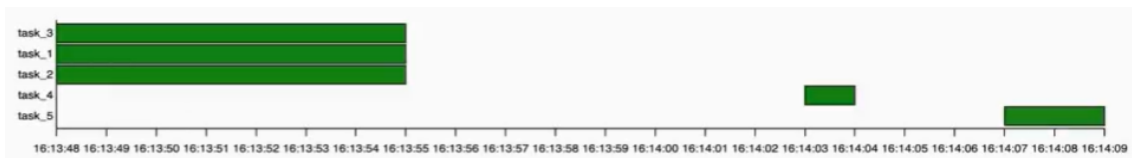
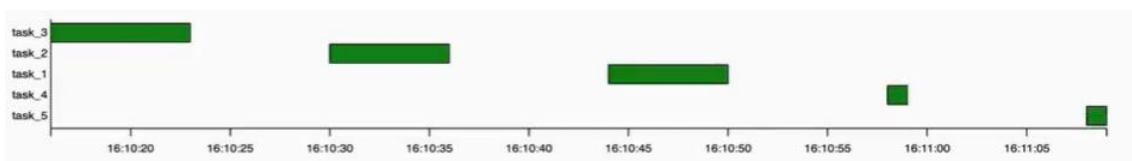
O LocalExecutor é ideal para quem precisa testar DAGs mais complexas ou até mesmo colocar o Airflow em produção em um ambiente de pequeno porte. Ele oferece boa performance sem a complexidade de configurar um cluster distribuído, como exigido pelo CeleryExecutor ou KubernetesExecutor. A Tabela abaixo demonstra algumas diferenças entre cada Executor comentado.

	LocalExecutor	CeleryExecutor	KubernetesExecutor
<b>Escalabilidade</b>	Limitada ao host local	Escalável horizontalmente (vários workers)	Altamente escalável via Kubernetes
<b>Isolamento de Tarefas</b>	Compartilham o mesmo host	Parcial (entre workers)	Total (cada tarefa em um pod isolado)
<b>Complexidade de Configuração</b>	Baixa	Média/Alta	Alta
<b>Requisitos de Banco de Dados</b>	PostgreSQL ou MySQL (não suporta SQLite)	PostgreSQL ou MySQL	PostgreSQL ou MySQL
<b>Tolerância a Falhas</b>	Baixa (depende do host local)	Média/Alta (workers independentes)	Alta (pods independentes e autocontidos)

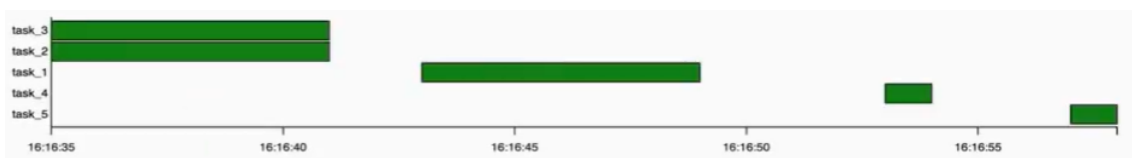
Gerenciamento de Recursos	Limitado	Limitado por máquina worker	Avançado (limites e requests por pod)
Ideal para	Testes locais e produções simples	Ambientes distribuídos com alto volume de tarefas	Ambientes em nuvem e infraestruturas modernas baseadas em K8s

Ainda nesse cenário, temos alguns dos vários parâmetros que podem ser alterados em sua utilização:

- **Paralellism** - Número máximo de tarefas que o scheduler pode executar simultaneamente, globalmente. Na primeira imagem abaixo, temos o caso de um valor de paralellism igual a 1. Em seguida, o valor de paralellism igual a 3.



- **dag\_concurrency** - Número máximo de tarefas que podem ser executadas em paralelo por DAG. Na imagem abaixo temos o valor dessa variável igual a 2, o que permite que duas tarefas sejam executadas em paralelo.



- **max\_active\_runs\_per\_dag** - Quantidade máxima de execuções ativas simultâneas por DAG.
- **worker\_concurrency** - Número máximo de tarefas que cada worker pode executar em paralelo (no caso do LocalExecutor, afeta quantos processos locais podem rodar tarefas).
- **sql\_alchemy\_conn** - String de conexão com o banco de dados (PostgreSQL ou MySQL).
- **fernet\_key** - Chave usada para criptografar dados sensíveis (como conexões).
- **max\_threads** - Número máximo de threads usados pelo scheduler para agendar tarefas.

Em seguida foi realizado a conexão com o PostgreSQL na interface do Airflow, permitindo que sejam feitas Querys para o banco de dados. Essas Querys funcionam de acordo com o banco de dados utilizado, nesse caso, o Postgres.

1 SELECT \* FROM dag\_run;

Show 100 entries

id	dag_id	execution_date	state	run_id	external_trigger	conf	end_date	start_date
1	parallel_dag	2019-01-01 00:00:00+00:00	success	scheduled_2019-01-01T00:00:00+00:00	False		2020-01-20 17:45:59.018549+00:00	2020-01-20 17:45:28.706823+00:00
2	parallel_dag	2019-01-02 00:00:00+00:00	success	scheduled_2019-01-02T00:00:00+00:00	False		2020-01-20 17:46:25.312958+00:00	2020-01-20 17:46:01.037941+00:00
3	parallel_dag	2019-01-03 00:00:00+00:00	success	scheduled_2019-01-03T00:00:00+00:00	False		2020-01-20 17:46:57.674726+00:00	2020-01-20 17:46:27.308843+00:00
4	parallel_dag	2019-01-04 00:00:00+00:00	running	scheduled_2019-01-04T00:00:00+00:00	False			2020-01-20 17:46:59.682429+00:00

Showing 1 to 4 of 4 entries

Previous 1 Next

## Celery Executor

O CeleryExecutor permite que as tarefas do Airflow sejam enviadas para uma fila distribuída e processadas por múltiplos workers, que podem estar em máquinas distintas. Essa arquitetura é baseada no framework Celery, amplamente usado para execução assíncrona de tarefas em aplicações Python.

Um ponto importante de se destacar é que, caso um worker trave ou tenha algum problema na execução, o Airflow passa as tarefas para outros Workers que estão em idle, tornando-o tolerante a falhas.

Na imagem abaixo temos a interface que nos permite visualizar os Workers Celery existentes. Nesse exemplo há somente um executor, mas é possível adicionar quantos forem necessários.

Flower Dashboard Tasks Broker Monitor Logout Docs Code

Active: 0 Processed: 0 Failed: 0 Succeeded: 0 Retried: 0

Search:

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@831974cc909c	Online	0	0	0	0	0	0.71, 0.43, 0.19

Showing 1 to 1 of 1 entries

Agora, quando executarmos uma DAG, ela será passada para o executor Celery.

	DAG	Schedule	Owner	Recent Tasks
Off	next_dag	00***	Airflow	
On	parallel_dag	00***	Airflow	
Off	pool_dag	00***	Airflow	
Off	queue_dag	00***	Airflow	

Na imagem abaixo, a execução da DAG foi feita pelo Celery Executor. É possível ainda alterar os mesmos parâmetros citados anteriormente para esse tipo de executor.

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@8da495f46f24f	Online	1	9	0	8	0	1.11, 0.62, 0.54

Abaixo existem alguns dos parâmetros do Celery na própria interface Flower.

Configuration options	
accept_content	['json', 'pickle']
broker_transport_options	{'visibility_timeout': 21600}
broker_url	redis://:*****@redis:6379/1
event_serializer	json
include	['celery.app.builtins', 'airflow.executors.celery_executor']
result_backend	db+postgresql://airflow:*****@postgres:5432/airflow
task_acks_late	True
task_default_exchange	default
task_default_queue	default
worker_concurrency	16
worker_prefetch_multiplier	1

## Kubernetes Executor

O `KubernetesExecutor` permite que o Airflow execute cada tarefa em um pod separado no Kubernetes. Em vez de utilizar workers fixos (como no `CeleryExecutor`), o `KubernetesExecutor` solicita ao Kubernetes que crie um pod para executar cada tarefa, o que garante isolamento total e melhor uso de recursos.

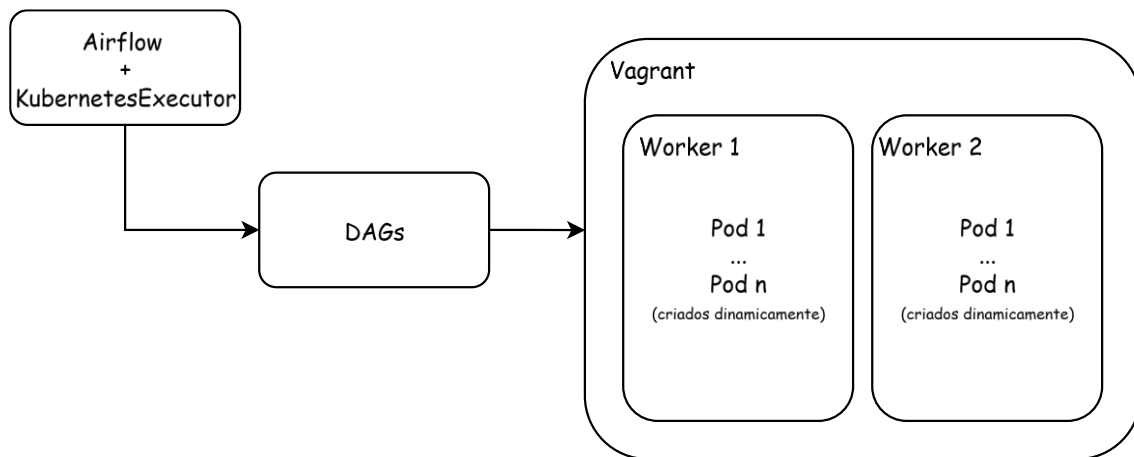
Como o Kubernetes cria um pod para executar cada tarefa, ele facilmente pode escalar o cluster conforme a demanda. Outro ponto importante é a integração com a nuvem, que se torna ideal para ambientes em EKS (AWS), GKE (Google), AKS (Azure).

Algumas nomenclaturas do Kubernetes que podem ser encontradas:

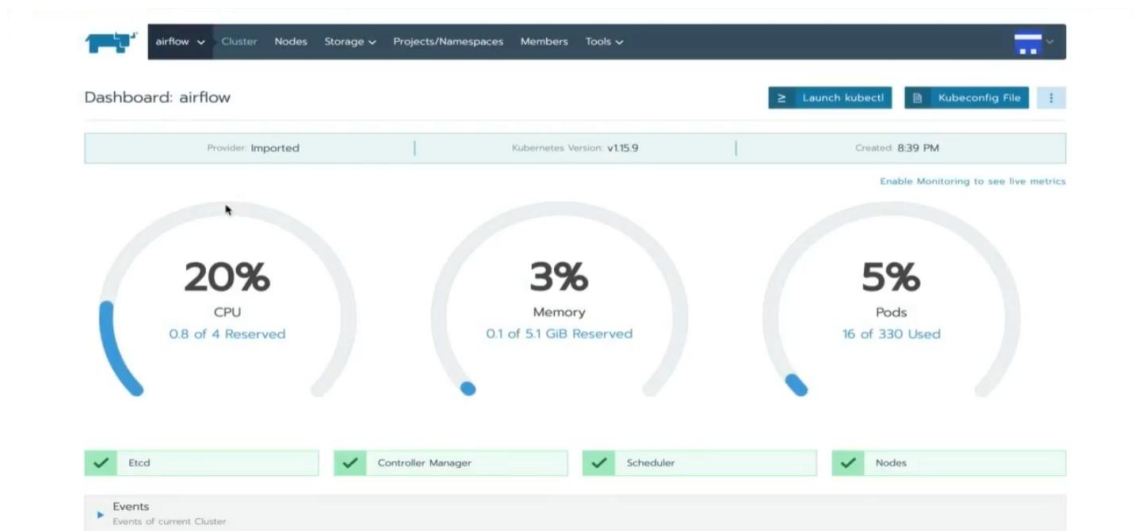
- Pod - Unidade básica de execução no Kubernetes. Pode conter um ou mais containers que compartilham rede e armazenamento.
- ReplicaSet - Garante que um número específico de réplicas de um pod esteja rodando. Normalmente gerenciado por um Deployment.
- Deployment - Controla a criação e o gerenciamento de múltiplas réplicas de um pod, com suporte a atualizações automáticas e rollback.
- Service - Abstração que define um endpoint de rede para expor um ou mais pods, permitindo comunicação interna ou externa.
- StorageClass - Define diferentes classes de armazenamento (como SSD ou HDD) com políticas específicas de provisionamento.
- PersistentVolumeClaim (PVC) - Requisição de armazenamento por parte de um pod. O PVC é vinculado a um PV disponível.

É importante salientar que, nesta seção, instala-se o Vagrant no host local para provisionar automaticamente múltiplas máquinas virtuais (VMs), que simularão um cluster Kubernetes. Utiliza-se o Rancher para gerenciar esse cluster de forma visual, incluindo a criação e monitoramento dos nós.

Cada VM atua como um nó do cluster e é responsável por executar os pods que rodam as tarefas das DAGs. Assim, a adição de novas VMs aumenta a capacidade de execução paralela do Airflow com o `KubernetesExecutor`. Entretanto, é importante observar que o custo computacional cresce proporcionalmente à adição de novos nós, exigindo mais recursos do host (ou custos em cloud, se for o caso).



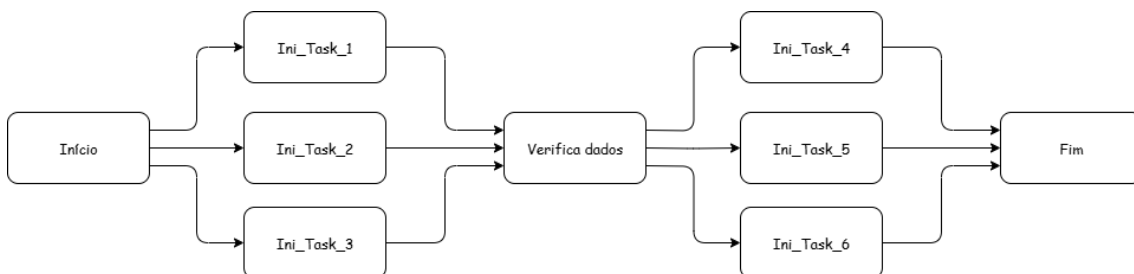
Na Figura abaixo, é demonstrado a interface do Rancher que está conectada com as VMs para executar tarefas em paralelo. É mostrado também o número de pods que podem ser criados, nesse caso, de 330.



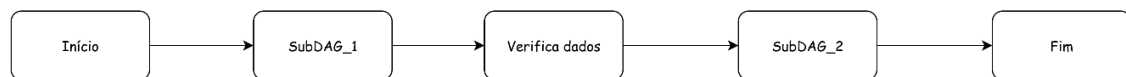
## 6. Improving your DAGs with advanced concepts

O primeiro tópico comentado pelo autor dos vídeos é minimizar repetições em DAGs, tornando a visualização mais simples, a partir de SubDAGs. As SubDAGs são gerenciadas a partir da DAG pai.

A Figura abaixo representa uma DAG qualquer sem a utilização das SubDAGs.



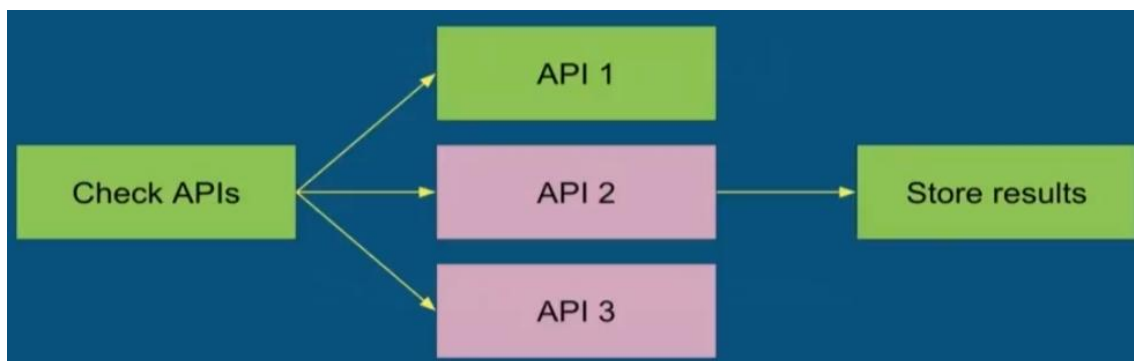
Já a Figura abaixo, demonstra a utilização das SubDAGs, a fim de simplificar o fluxo de execução das DAGs.



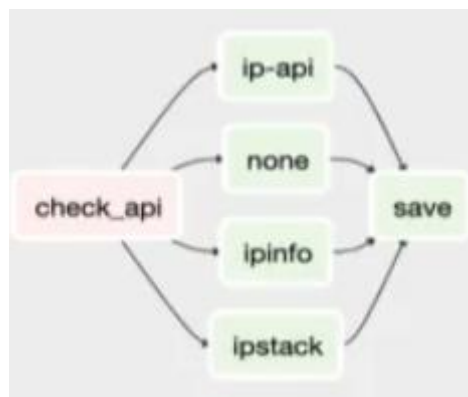
É importante salientar que para que as tarefas internas sejam executadas em paralelo, a SubDAG precisa:

- Ter o mesmo `schedule_interval` e `start_date` que a DAG principal.
- Ter `default_args` que respeita a DAG principal.
- Usar o mesmo executor (ex: Local, Celery).
- Ter um valor adequado de `max_active_runs`.

**Branching:** É uma forma que permite as DAGs escolherem diferentes caminhos a serem seguidos de acordo com o resultado de uma task específica. A imagem abaixo demonstra um caso onde uma DAG verifica 3 APIs diferentes, e dependendo se elas estiverem disponíveis ou não, ele tomará a decisão automaticamente de qual escolher.



Na representação abaixo, há o mesmo exemplo acima implementado no Airflow, onde há 3 APIs possíveis a serem utilizadas pela DAG e note também que há uma outra task chamada “none”. Essa task é utilizada quando nenhuma das APIs utilizáveis é retornada na função `BranchPythonOperator`, funcionando como um ponto de saída da DAG.



Pode-se também combinar tasks, ou seja, utilizar mais de uma durante a execução da DAG, como a `ipstack + ip-api` (Isso deverá ser alterado no código Python da DAG).

**Trigger Rules:** determinam quando uma tarefa pode ser executada, com base no estado das suas tarefas upstream. Temos as seguintes regras aplicáveis:

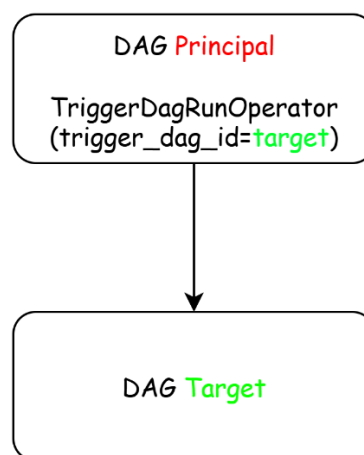
- `all_success` - A tarefa só executa se todas as tarefas upstream forem bem-sucedidas (Essa é a padrão do Airflow).

- `all_failed` - Executa somente se todas as tarefas upstream falharem.
- `one_success` - Executa se pelo menos uma tarefa upstream for bem-sucedida.
- `one_failed` - Executa se pelo menos uma tarefa upstream falhar.
- `all_done` - Executa independentemente de sucesso ou falha.
- `none_failed` - Executa se nenhuma tarefa upstream falhou.
- `none_skipped` - Executa se nenhuma tarefa upstream foi ignorada (skipped).

XCom: É o mecanismo nativo do Airflow para compartilhar dados entre tarefas. Ele é muito útil quando você precisa passar informações geradas por uma tarefa para outra dentro da mesma DAG, como uma consulta em um banco de dados e validar uma informação contida nele. Utiliza-se o `xcom_push()` para o envio de dados, e `xcom_pull()` para recuperar dados.

Os dados são armazenados em JSON no banco de metadados (xcom table). Deve-se evitar dados muito grandes nessa tabela, como imagens, arquivos, dentre outros.

TriggerDagRunOperator: é um operador do Airflow que permite disparar a execução de outra DAG a partir de uma DAG principal. É utilizado quando queremos encadear workflows independentes, modularizar processos ou executar pipelines complexos. A Figura abaixo demonstra a utilização desse operador, que dispara uma DAG a partir de outra.

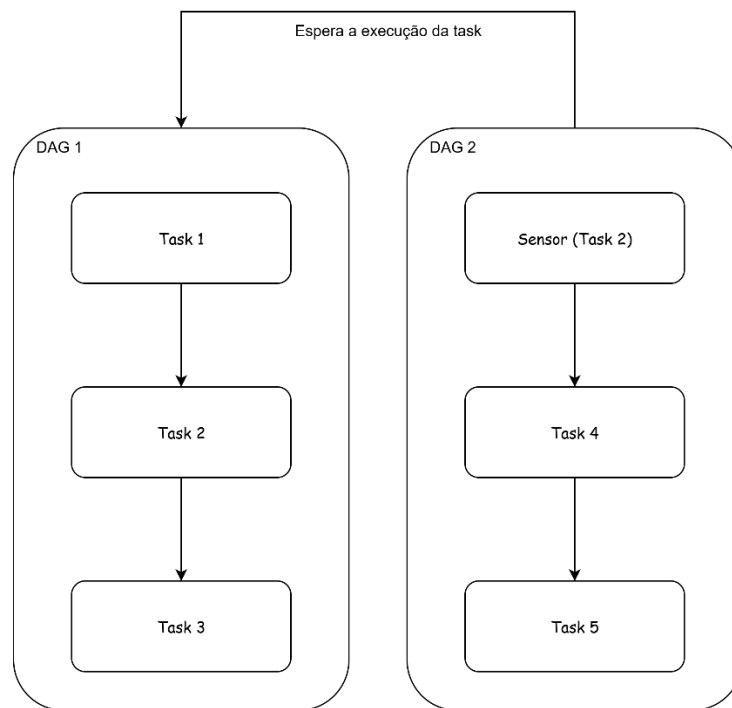


Importante notar que a Target não depende da outra terminar para iniciar, e vice-versa. Elas também são independentes uma da outra, então não é possível verificar informações da Target a partir da Principal, e da Principal para a Target.

Além disso, existem algumas questões que devem ser tomados com cuidado, como:

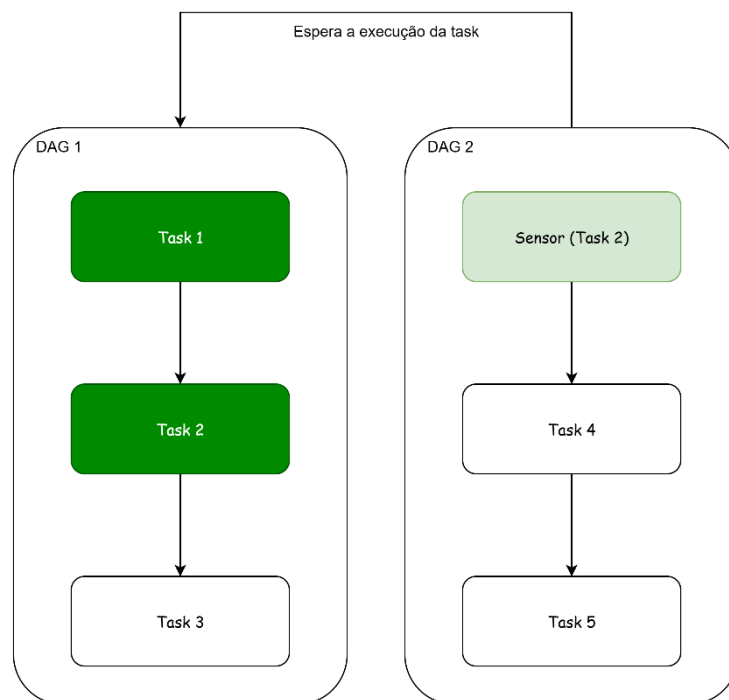
- Evite loops de DAGs: uma DAG chamando a outra, como a Principal chamando a Target e vice-versa;
- Controle de concorrência: A DAG acionada deve ter bem definido o `max_active_runs` para evitar múltiplas execuções simultâneas indesejadas.

ExternalTaskSensor: Ainda no aspecto de relação entre as DAGs, temos esse sensor que atua observando o estado de uma task de uma DAG pai, com o objetivo de executar alguma outra task relevante naquela execução.



A imagem abaixo demonstra como seria essa execução, onde quando a task 2 é executada e finalizada, a DAG com o sensor observador inicia, executando o seu fluxo da task 4 e 5.

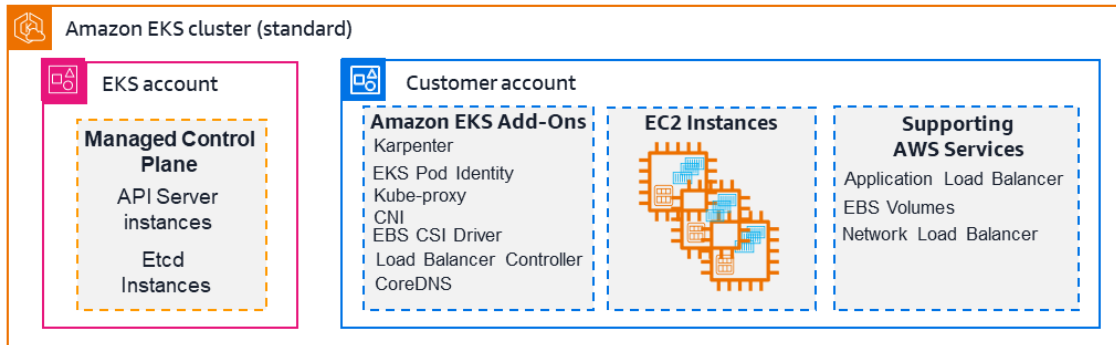
Isso pode ser útil em questão de logs, manipulação dos dados para um certo formato em outra DAG, dentre outros.





## 7. Deploying Airflow on AWS EKS with Kubernetes Executors and Rancher

Primeiramente, é importante saber que o AWS EKS (Elastic Kubernetes Service) é um serviço totalmente gerenciado que permite executar clusters do Kubernetes em uma infraestrutura sem servidor. Esse serviço possui estrutura definida na imagem abaixo.



Em seguida, é importante configurar uma conta desse serviço, para isso será necessário acessar o AWS Management Console. Para isso, siga o passo a passo no link abaixo para realizar a configuração e criação do EKS.

<[https://docs.aws.amazon.com/pt\\_br/eks/latest/userguide/create-cluster.html#step2-console](https://docs.aws.amazon.com/pt_br/eks/latest/userguide/create-cluster.html#step2-console)>

Após fazer essa configuração, você provavelmente chegará nessa tela do CLI do servidor EKS. Em seguida, será necessário instalar o Docker na máquina, para isso, rode o comando “sudo amazon-linux-extras install docker”.

```
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
3 package(s) needed for security, out of 24 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-8-71 ~]$
```

i-07298310ca7c0c960  
Public IPs: 35.180.33.242 Private IPs: 172.31.8.71

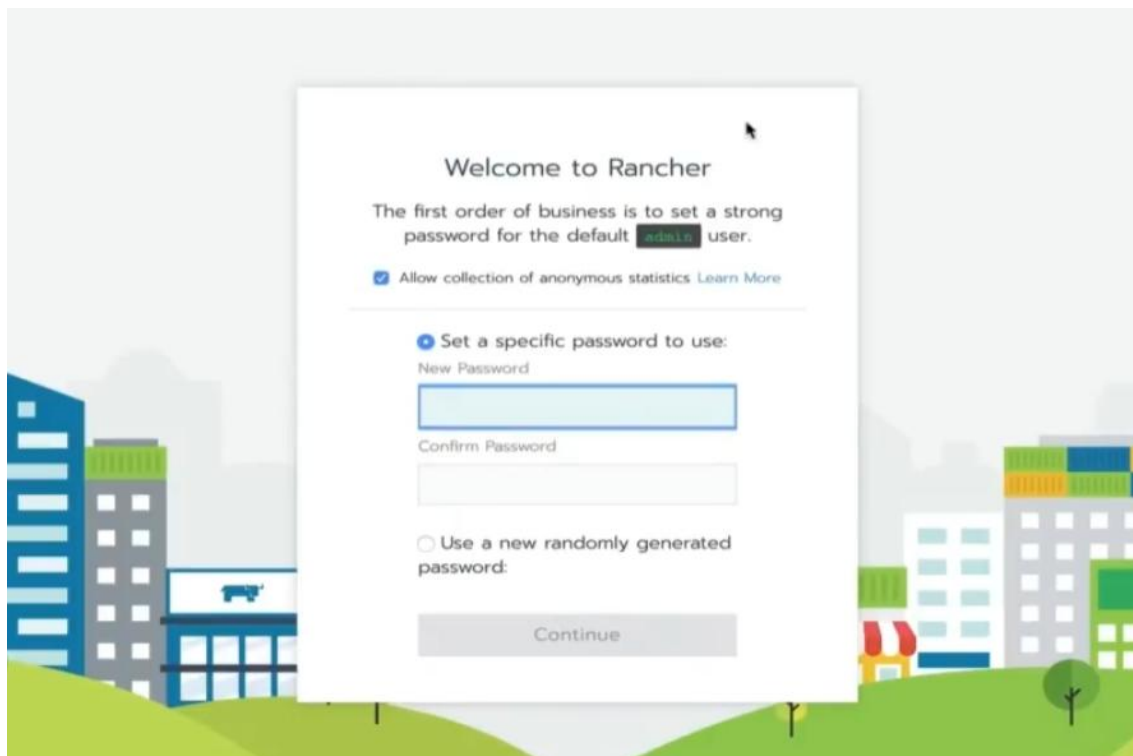
Em seguida inicie o serviço do Docker a partir do comando “sudo service docker start”.

```
Complete!
[ec2-user@ip-172-31-8-71 ~]$ sudo amazon-linux-extras install docker
```

Reinicie o servidor em seguida. Após isso, será necessário instalar o Rancher no nosso container Docker e em seguida rodar o container. Faça isso a partir do comando abaixo.

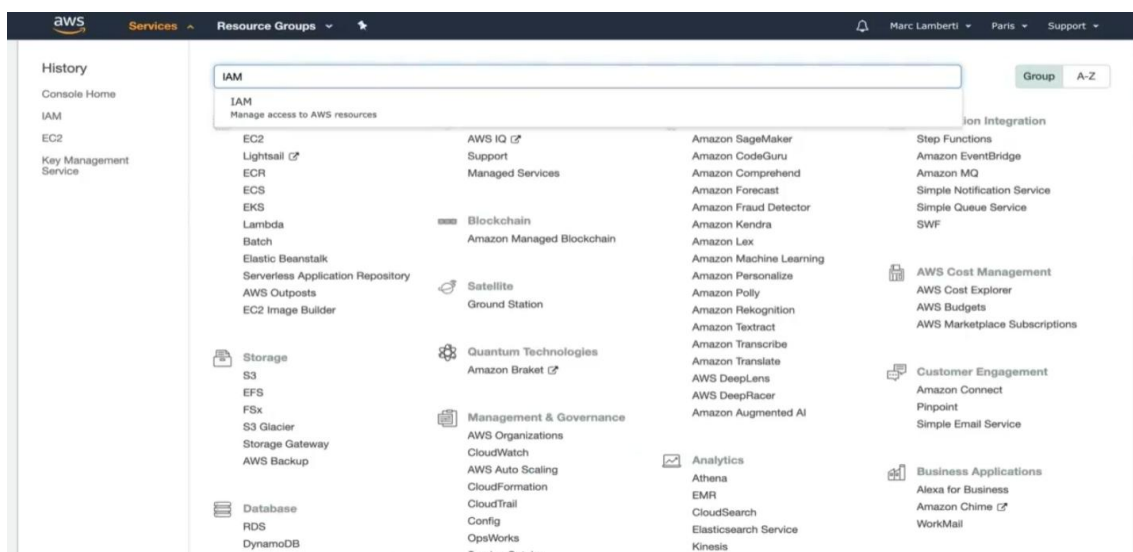
```
ec2-user@ip-172-31-8-71 ~]$ docker run -d --restart=unless-stopped --name rancher --hostname rancher -p 80:80 -p 443:443 rancher/rancher:v2.3.2
```

Após isso, copie o IP da máquina da AWS (estará logo abaixo do CLI, ao lado de Public IPs) e abra uma guia em seu navegador e cole o IP na barra de pesquisa. Pronto, o Rancher está instalado na máquina AWS EKS.

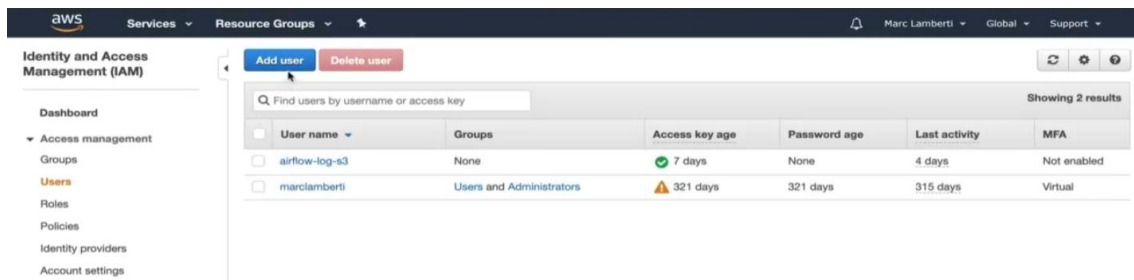


Em seguida precisaremos criar um usuário IAM, que fará a conexão com o nosso servidor AWS EKS. O usuário IAM é um Gerenciador de Identidade e Acesso e funciona para representar a pessoa ou a workload que usa o usuário do IAM para interagir com a AWS.

Para isso, abra o AWS Management Console e pesquise por IAM.



Acesse a seção “Usuários” ou “Users” na barra lateral esquerda e clique em “Adicionar usuário” ou “Add user”.



Durante a criação do usuário, insira as seguintes configurações:

## Add user

### Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name\*

[Add another user](#)

### Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type\* ☒ **Programmatic access**  
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**  
Enables a **password** that allows users to sign-in to the AWS Management Console.

## Add user

### Set permissions

- [Add user to group](#) [Copy permissions from existing user](#) [Attach existing policies directly](#)

[Create policy](#)

Filter policies  Showing 21 results

	Policy name	Type	Used as
<input checked="" type="checkbox"/>	<a href="#">AdministratorAccess</a>	Job function	Permissions policy (2)
<input type="checkbox"/>	<a href="#">AmazonAPIGatewayAdministrator</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AmazonWorkSpacesAdmin</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AmazonWorkSpacesApplicationManagerAdminAccess</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AWSAppSyncAdministrator</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AWSCloud9Administrator</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AWSCodeBuildAdminAccess</a>	AWS managed	None
<input type="checkbox"/>	<a href="#">AWSFMAAdminFullAccess</a>	AWS managed	None

Aqui não é necessário colocar nenhuma informação.

## Add user

### Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
<input type="text" value="Add new key"/>	<input type="text"/>	<a href="#">Remove</a>

You can add 50 more tags.

Por fim, é mostrado um resumo das configurações do usuário a ser criado.

## Add user

1 2 3 4 5

### Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

#### User details

User name	airflow-user
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

#### Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

#### Tags

No tags were added.

Pronto, o usuário está criado.

## Add user

1 2 3 4 5



### Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

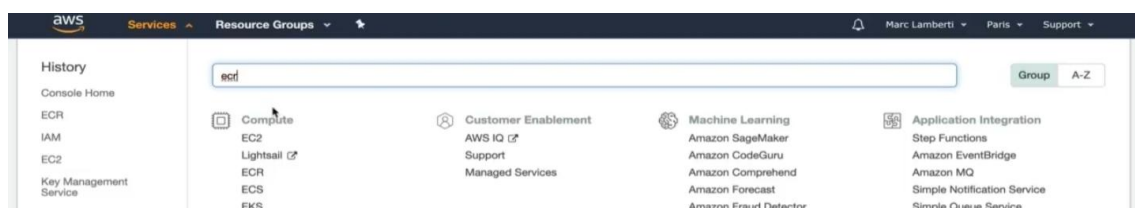
Users with AWS Management Console access can sign-in at: <https://953076219999.signin.aws.amazon.com/console>

Download .csv

User	Access key ID	Secret access key
airflow-user	AKIA53Z6FABPVZMGG3IS	***** Show

Após a criação do usuário, precisaremos criar um repositório ECR. Esse repositório armazenará as imagens do Docker que utilizaremos, permitindo criar, monitorar e excluir repositórios de imagens e definir permissões que controlam quem pode acessá-los usando operações de API do Amazon ECR.

Novamente, no AWS Management Console, pesquise por ECR.



Clique no botão laranja na seção de criar o repositório.



Em seguida, insira um nome para o repositório que utilizaremos e deixe desabilitado as duas configurações.

aws Services Resource Groups

ECR > Repositories > Create repository

## Create repository

**Repository configuration**

Repository name  
 953076219999.dkr.ecr.eu-west-3.amazonaws.com/

A namespace can be included with your repository name (e.g. namespace/repo-name).

**Tag immutability**  
 Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.  
☐ Disabled

**Scan on push**  
 Enable scan on push to have each image automatically scanned after being pushed to a repository. If disabled, each image scan must be manually started to get scan results.  
☐ Disabled

Cancel **Create repository**



Para fazer a instalação e configuração do AWS CLI que utilizaremos nesse repositório Docker, siga o passo a passo no link abaixo. Note que estamos utilizando uma máquina Linux, então vá para as configurações desse sistema operacional.

<<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>>


Note que durante o processo de instalação, precisaremos do usuário IAM criado, juntamente com a senha criada automaticamente para ele.

```
Marc@macbook-pro ~/airflow-materials/airflow-section-6 (master) $ sudo ./aws/install
Password:
You can now run: /usr/local/bin/aws2 --version
Marc@macbook-pro ~/airflow-materials/airflow-section-6 (master) $ aws2 --version
aws-cli/2.0.0dev3 Python/3.7.4 Darwin/19.2.0 botocore/2.0.0dev2
Marc@macbook-pro ~/airflow-materials/airflow-section-6 (master) $ aws2 configure
AWS Access Key ID [*****S6NH]: AKIA53Z6FABPVZMGG3IS
AWS Secret Access Key [*****/Yw3]: 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
Default region name [eu-west-3]:
Default output format [None]:
Marc@macbook-pro ~/airflow-materials/airflow-section-6 (master) $
```


Voltando para o Rancher que criamos no servidor AWS, faremos a criação de um cluster EKS com o Rancher. Primeiramente, iremos adicionar um novo servidor, clicando no Amazon EKS.


Global
Clusters
Apps
Users
Settings
Security
Tools


### Add Cluster - Select Cluster Type



**From existing nodes (Custom)**  
Create a new Kubernetes cluster using RKE, out of existing bare-metal servers or virtual machines.



**Import an existing cluster**  
Import an existing Kubernetes cluster. The provider that created it will continue to manage the provisioning and configuration of the cluster.

With RKE and new nodes in an infrastructure provider

Amazon EC2

Azure

DigitalOcean

Linode

vSphere

With a hosted Kubernetes provider



**Amazon EKS**

Azure AKS

Google GKE

[Cancel](#)

Nomeie o cluster que você está criando de uma forma fácil de ser lembrada. Note a região do servidor que está sendo criado o cluster, ele precisa ser o mesmo que você estiver conectado no AWS Management Console. Será necessário utilizar as mesmas credenciais do usuário IAM utilizado anteriormente.



Marc Lamberti


- US East (N. Virginia) us-east-1
- US East (Ohio) us-east-2
- US West (N. California) us-west-1
- US West (Oregon) us-west-2
- Asia Pacific (Hong Kong) ap-east-1
- Asia Pacific (Mumbai) ap-south-1
- Asia Pacific (Seoul) ap-northeast-2
- Asia Pacific (Singapore) ap-southeast-1
- Asia Pacific (Sydney) ap-southeast-2
- Asia Pacific (Tokyo) ap-northeast-1
- Canada (Central) ca-central-1
- Europe (Frankfurt) eu-central-1
- Europe (Ireland) eu-west-1
- Europe (London) eu-west-2
- Europe (Paris) eu-west-3**
- Europe (Stockholm) eu-north-1
- Middle East (Bahrain) me-south-1
- South America (São Paulo) sa-east-1

### Add Cluster - Amazon EKS

Cluster Name \*
Add a Description

airflowEKScuster



Note: Currently Amazon EKS will not create an ingress controller when launching a new cluster. If you need this functionality you will have to create an ingress controller manually after cluster creation.

**Member Roles**  
Control who has access to the cluster and what permission they have to change it.

**Labels & Annotations**  
Configure labels and annotations for the cluster. None

**Account Access**  
Choose the region and API Key that will be used to launch Amazon EKS

Region

eu-west-3

Access Key

AKIA53Z6FABPVZMGG3IS

Secret Key

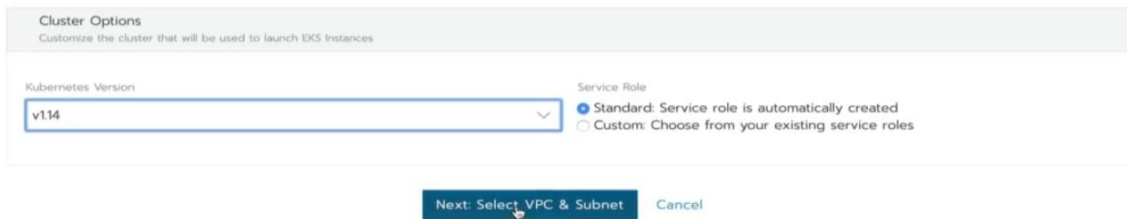
.....

Session Token *Optional*

Your AWS session token

Paste in your AWS key pair here. Use only IAM access keys, using keys generated from the root user will make the cluster unreachable.

Em seguida defina a versão do Kubernetes e prossiga para o próximo passo.



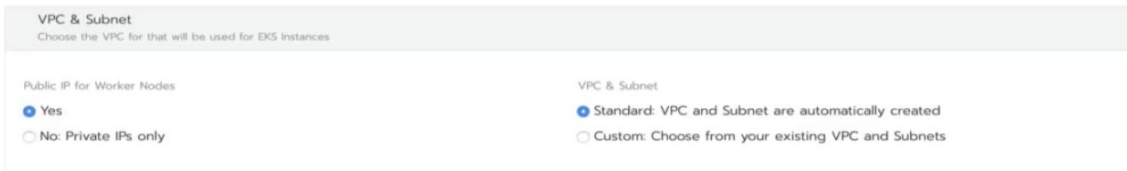
Cluster Options  
Customize the cluster that will be used to launch EKS Instances

Kubernetes Version  
v1.14

Service Role  
☒ Standard: Service role is automatically created  
☐ Custom: Choose from your existing service roles

Next: Select VPC & Subnet Cancel

Essa configuração é a padrão, deixe-a como está.

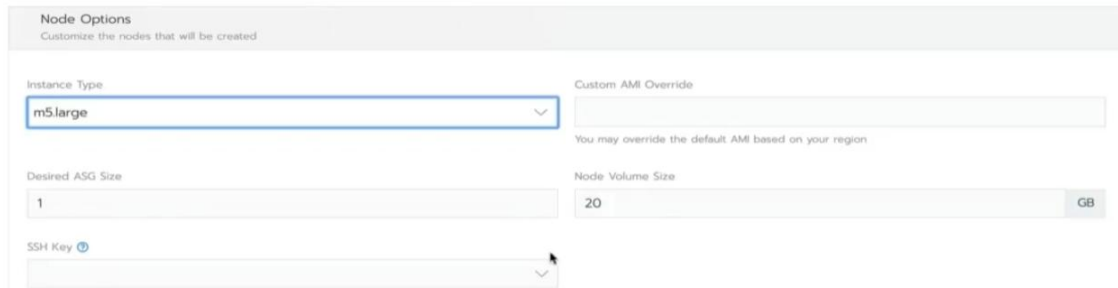


VPC & Subnet  
Choose the VPC for that will be used for EKS Instances

Public IP for Worker Nodes  
☒ Yes  
☐ No: Private IPs only

VPC & Subnet  
☒ Standard: VPC and Subnet are automatically created  
☐ Custom: Choose from your existing VPC and Subnets

Por fim, defina o tipo da instância, isso significa o tamanho e poder computacional que você utilizará (note que tipos maiores e mais fortes custam mais caro). Os demais valores permanecem os mesmos. Por fim, crie o cluster.



Node Options  
Customize the nodes that will be created

Instance Type  
m5.large

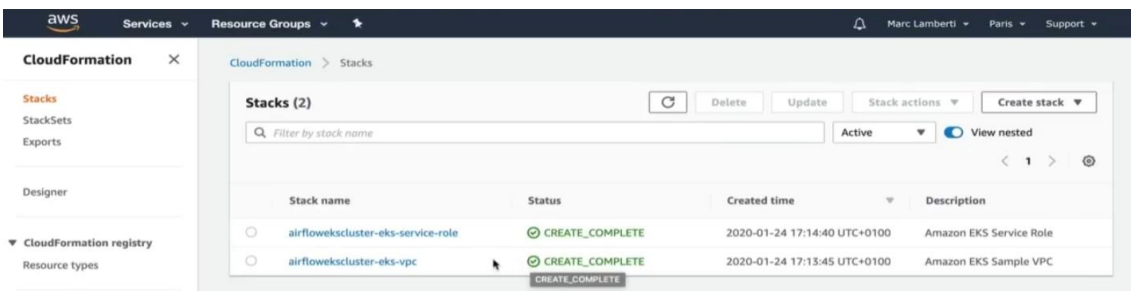
Custom AMI Override  
You may override the default AMI based on your region

Desired ASG Size  
1

Node Volume Size  
20 GB

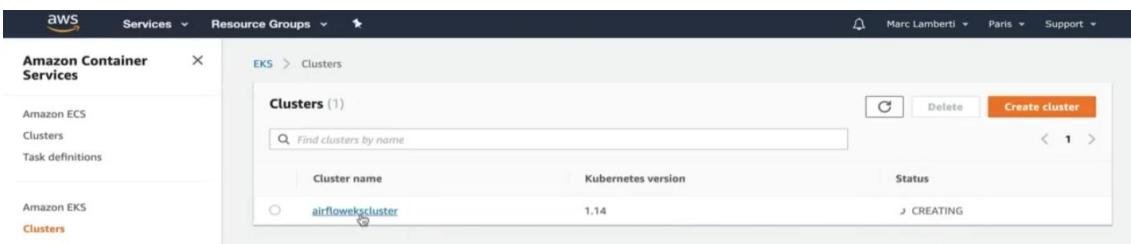
SSH Key  
[Default Key]

Acessando o serviço CloudFormation da AWS, você conseguirá ver que a AWS já conseguiu detectar o cluster criado.



Stack name	Status	Created time	Description
airflowekscluster-eks-service-role	CREATE_COMPLETE	2020-01-24 17:14:40 UTC+0100	Amazon EKS Service Role
airflowekscluster-eks-vpc	CREATE_COMPLETE	2020-01-24 17:13:45 UTC+0100	Amazon EKS Sample VPC

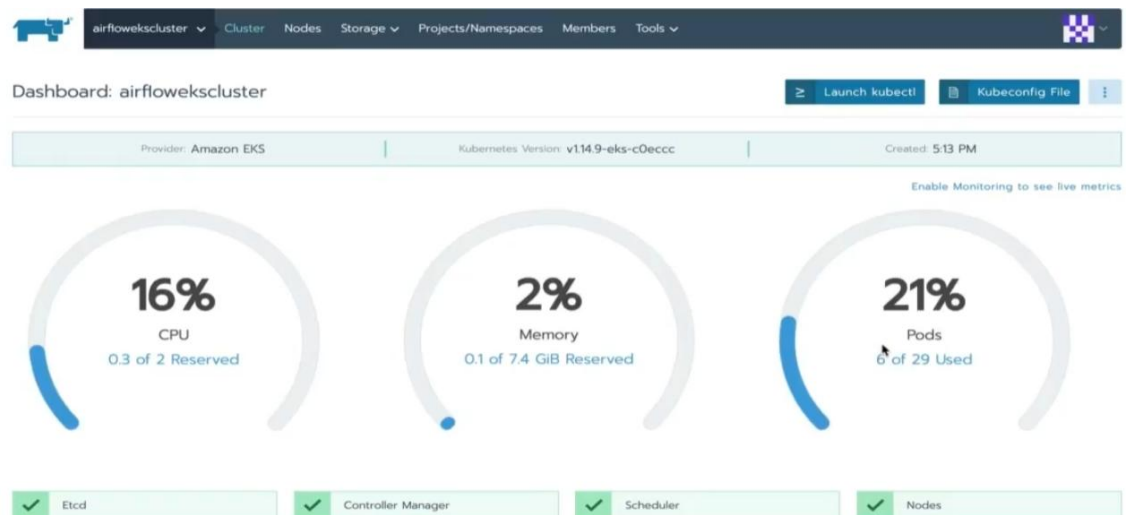
O mesmo acontece para o ECR.



Cluster name	Kubernetes version	Status
airflowekscluster	1.14	CREATING

No Rancher nós conseguimos ver que já é possível acessar o cluster.



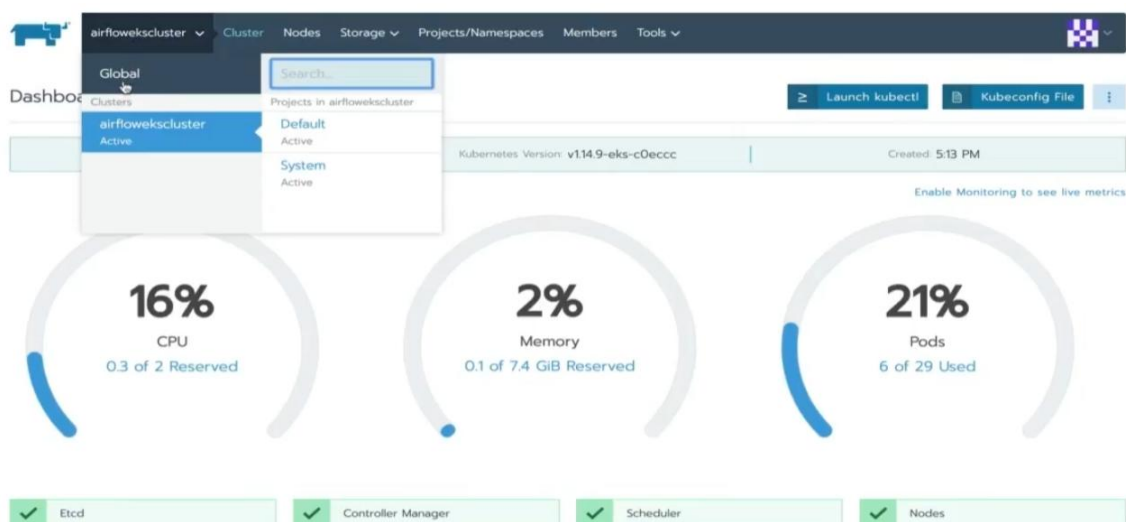


Por fim, precisaremos configurar o Nginx Ingress, que é um componente essencial quando usamos Kubernetes para expor aplicações web para o mundo externo.

Um Ingress é um recurso que define como o tráfego externo é roteado para dentro do cluster. O Ingress Controller é o componente que executa isso de fato.

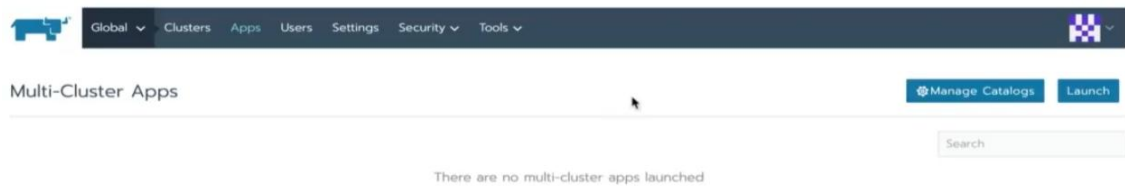
Para isso, precisaremos configurar um Catalog, que é um conjunto de aplicações pré-configuradas empacotadas como Helm Charts. Ele permite que você:

- Instale apps no cluster via interface gráfica.
- Personalize parâmetros como porta, domínio, volumes, usuários etc.
- Crie facilmente estruturas Kubernetes prontas para uso.

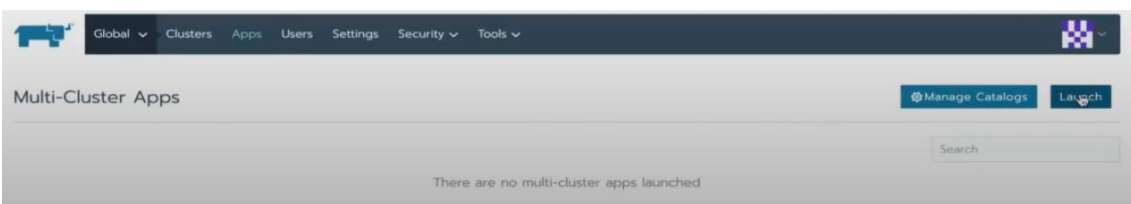
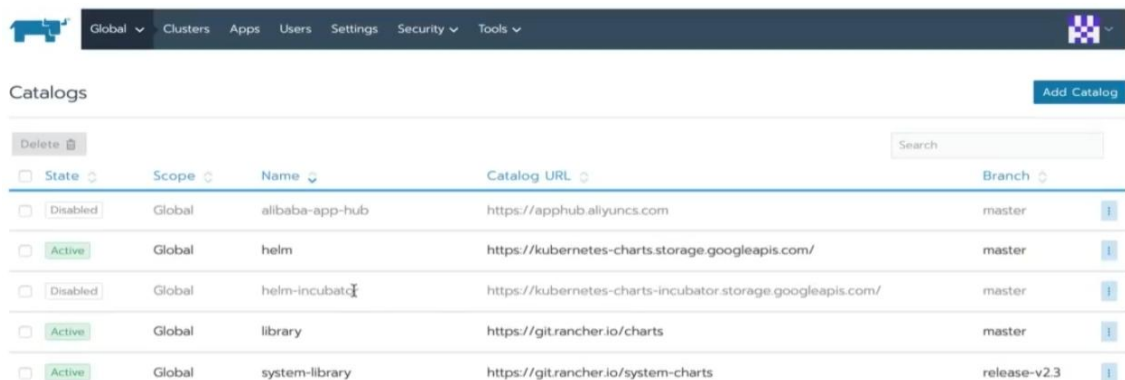


Em seguida vá para a aba “Apps”, e clique em Manage Catalogs.

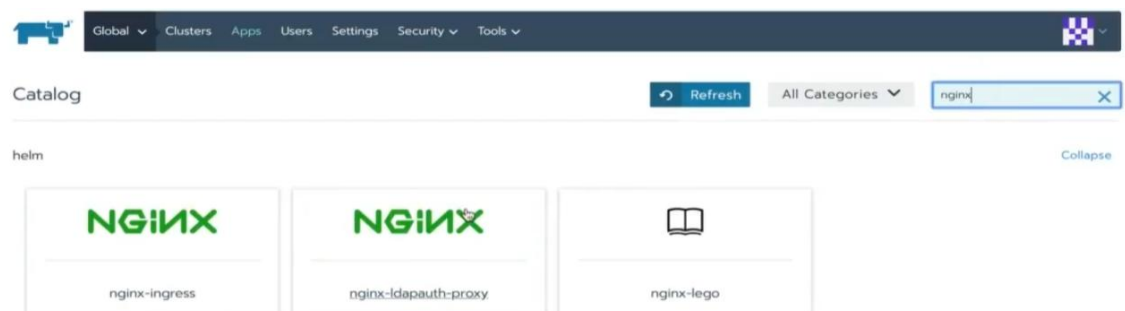




Aqui, adicionaremos o Helm Chart chamado “helm”. Em seguida, volte para a guia de aplicativos novamente e clique em Launch.



Em seguida, procure pelo aplicativo “nginx-ingress”, ele fará a conexão com acessos externos.



Nas opções de configuração do Nginx, insira um nome e defina o projeto a ser instalado, no nosso caso será o “Default”.

**Configuration Options**  
 Helm templates accept a comma separated list of strings

Name \*  Add a Description Template Version

TARGETS

Target Projects \*

Add a target project

airflowekscluster

Default

System

no targets

UPGRADES

Upgrade Strategy

☐ Rolling update (batched)

☒ Upgrade all apps simultaneously

ROLES

Available Roles

☐ Project - This app will be able to access and manage resources in the projects in which it is deployed

☒ Cluster - This app will be able access and manage all resources in the clusters in which it is deployed

Answers

Variable \*

e.g. FOO

Value

e.g. bar

ProTip: Paste lines of key-value pairs into any key field for easy bulk entry

+ Add Answer

Por fim, clique em “Launch”. Pronto, seu Nginx Ingress já estará funcionando.

**Answer Overrides**  
 Override individual answers on a per cluster/project basis

+ Add Override

Search

Scope Question Answer

There are no overrides

PREVIEW

Launch Cancel

## 8. Monitoring Apache Airflow

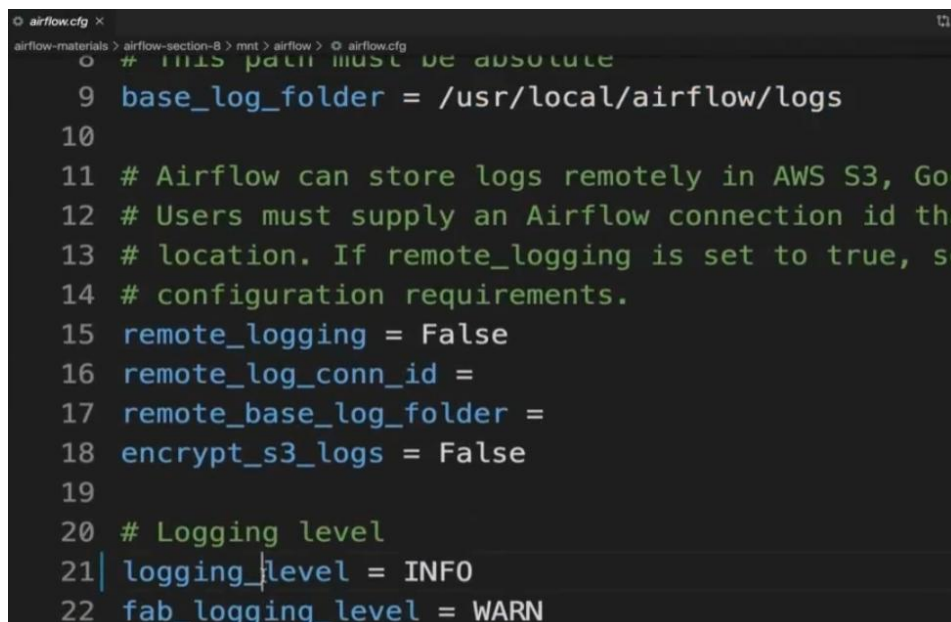
Essa seção tem como foco mostrar o monitoramento de instâncias do Airflow, bem como as DAGs em execução, ou que são utilizadas de forma agendada, Workers, o próprio backend, dentre outros. É importante acompanhar a execução das execuções pois é imprescindível saber se a DAG é executada corretamente, se há algum erro em uma API, ou na transformação dos dados, dentre inúmeros fatores que possam ocorrer nas estruturas citadas anteriormente.

Existem cinco níveis de log, INFO, ERROR, DEBUG, WARNING e CRITICAL, esses dependem do tipo de erro que se deu na execução. Abaixo há um exemplo da implementação de um logger.

```
def setup_logging(filename):
    """Creates log file handler for daemon process"""
    root = logging.getLogger()
    handler = logging.FileHandler(filename)
    formatter = logging.Formatter(settings.SIMPLE_LOG_FORMAT)
    handler.setFormatter(formatter)
    root.addHandler(handler)
    root.setLevel(settings.LOGGING_LEVEL)

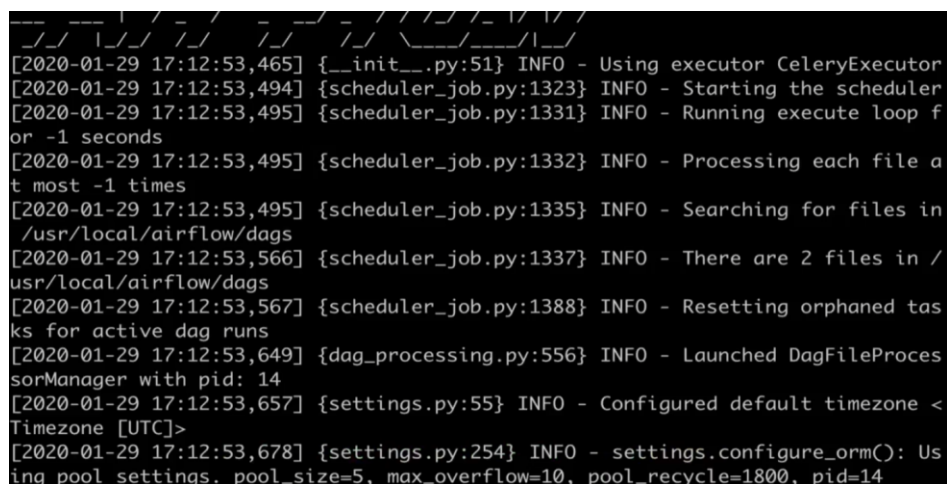
    return handler.stream
```

Para que seja definido o nível do log, é necessário alterar o parâmetro chamado “logging\_level” no arquivo de configuração do Airflow. Esse por sua vez pode receber os cinco níveis citados anteriormente. Por padrão, o valor é “INFO”.



```
airflow-section-8 > mnt > airflow > airflow.cfg
# This path must be absolute
9 base_log_folder = /usr/local/airflow/logs
10
11 # Airflow can store logs remotely in AWS S3, Google Cloud Storage or Azure Blob Storage
12 # Users must supply an Airflow connection id that points to the bucket
13 # location. If remote_logging is set to true, se
14 # configuration requirements.
15 remote_logging = False
16 remote_log_conn_id =
17 remote_base_log_folder =
18 encrypt_s3_logs = False
19
20 # Logging level
21 logging_level = INFO
22 fab_logging_level = WARN
```

Com esse valor, ao acessar os registros de log, a partir do comando “docker logs f <id\_instancia>” da instância do Airflow, é mostrado uma lista com mensagens de acompanhamento do fluxo normal da aplicação.



```

[2020-01-29 17:12:53,465] {__init__.py:51} INFO - Using executor CeleryExecutor
[2020-01-29 17:12:53,494] {scheduler_job.py:1323} INFO - Starting the scheduler
[2020-01-29 17:12:53,495] {scheduler_job.py:1331} INFO - Running execute loop f
or -1 seconds
[2020-01-29 17:12:53,495] {scheduler_job.py:1332} INFO - Processing each file a
t most -1 times
[2020-01-29 17:12:53,495] {scheduler_job.py:1335} INFO - Searching for files in
/usr/local/airflow/dags
[2020-01-29 17:12:53,566] {scheduler_job.py:1337} INFO - There are 2 files in /
usr/local/airflow/dags
[2020-01-29 17:12:53,567] {scheduler_job.py:1388} INFO - Resetting orphaned tas
ks for active dag runs
[2020-01-29 17:12:53,649] {dag_processing.py:556} INFO - Launched DagFileProces
sorManager with pid: 14
[2020-01-29 17:12:53,657] {settings.py:55} INFO - Configured default timezone <
Timezone [UTC]>
[2020-01-29 17:12:53,678] {settings.py:254} INFO - settings.configure_orm(): Us
ing pool settings. pool_size=5, max_overflow=10, pool_recycle=1800, pid=14

```

Passando o valor da variável “logging\_level” para “WARNING”, os logs são ocultados, por conta de receber apenas avisos de execução que não impedem de executar.

```
airflow.cfg
airflow-section-8 > mnt > airflow > airflow.cfg
9 base_log_folder = /usr/local/airflow/logs
10
11 # Airflow can store logs remotely in AWS S3, Go
12 # Users must supply an Airflow connection id th
13 # location. If remote_logging is set to true, s
14 # configuration requirements.
15 remote_logging = False
16 remote_log_conn_id =
17 remote_base_log_folder =
18 encrypt_s3_logs = False
19
20 # Logging level
21 logging_level = WARNING
22 fab_logging_level = WARN
```

Como é possível verificar na imagem abaixo, nenhum log é mostrado, indicando que não há Warnings na execução.

```
Marc@macbook-pro ~/airflow-materials/airflow-section-8 (master) $ docker logs 29c7230dc687
```

```
-----
  ____  |__ ( )-----  __/1  /-----
  ____ /| |_ /__ /___ /___ /___ /___ \ | / / /
  ____ _| / /___ /___ /___ /___ /___ /___ /___ /
  __/___ | / /___ /___ /___ /___ /___ /___ /___ /
Marc@macbook-pro ~/airflow-materials/airflow-section-8 (master) $
```

Isso se repetirá para os outros casos também:

- **DEBUG** - Mensagens detalhadas, úteis apenas durante o desenvolvimento e depuração.
- **ERROR** - Erros que afetam a execução da DAG ou task, mas que não encerram o sistema inteiro.
- **CRITICAL** - Indica um erro grave, que geralmente exige atenção imediata ou interrompe todo o sistema.

Elasticsearch: É uma plataforma distribuída de busca e análise de dados em tempo real, usada em sistemas como o Apache Airflow para armazenar, indexar e consultar logs e métricas.

Essa plataforma também permite a seguinte estrutura de armazenagem, como o ID, nome do log, mensagem de saída, nível do log, processo e tipo.

```
{
  "_id": 2,
  "Name": "my_log",
  "Message": "it works",
  "LogLevel": "INFO",
  "Process": "ueryd_8990"
  "_type": "log_sys"
}
```

Além disso, ele permite buscar logs a partir do DAG ID, task ID, status (failed, success, etc.), host ou worker específico, data de execução.

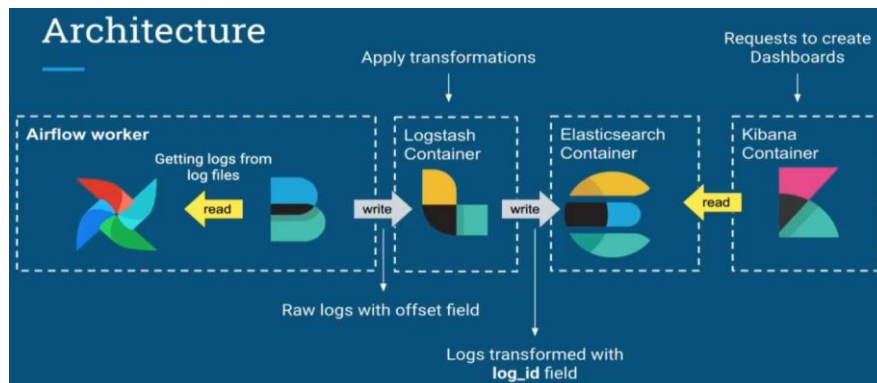
A partir dele e outras ferramentas open source, é possível combiná-los para coleta, armazenamento, análise e visualização de dados log e métricas, chamado de ELK (Filebeat + Elasticsearch + Logstash + Kibana, o primeiro não tem relação direta com a sigla).

Ao utilizar o “ELK Stack” com o Airflow, a arquitetura de organização fica da seguinte forma:



Como esse processo será feito?

1. Coleta de logs (Airflow Worker → Filebeat)
  - a. O Filebeat é encarregado de ler continuamente os arquivos de log do Airflow e extrai eventos linha a linha.
2. Processamento dos logs (Filebeat → Logstash)
  - a. Os dados brutos são enviados do Filebeat para o Logstash por meio de uma conexão (geralmente via protocolo beats).
3. Indexação (Logstash → Elasticsearch)
  - a. O Logstash envia os logs já processados para o Elasticsearch Container.
4. Visualização (Elasticsearch → Kibana)
  - a. O Kibana Container se conecta ao Elasticsearch para ler os dados indexados.



Além dos logs, o Airflow também pode emitir métricas de desempenho e uso (número de DAGs executadas, tempo de execução de tarefas, número de falhas etc.). Isso é feito por meio do StatsD, que é integrado nativamente ao Airflow através do parâmetro `statsd_on`.

As métricas coletadas podem ser enviadas a um servidor StatsD ou Telegraf, que as encaminha ao InfluxDB (base de dados de séries temporais). O Grafana então é utilizado como ferramenta de visualização.



Essa arquitetura é conhecida como TIG Stack:

- Telegraf – Coletor de métricas.
- InfluxDB – Banco de dados de métricas.
- Grafana – Painel de visualização e monitoramento.

No Airflow, exemplos de métricas expostas:

- `airflow.scheduler_heartbeat`;
- `airflow.dag_processing.time`;
- `airflow.task.duration`;
- `airflow.operator_failures`.

Triggers e Alertas com Grafana: O Grafana, quando conectado ao InfluxDB ou ao Elasticsearch, permite configurar alertas e triggers com base em condições específicas, como:

- Número de falhas em tarefas de uma DAG em um intervalo de tempo;
- Duração acima do esperado de uma task;
- Execuções que não ocorreram no período previsto.

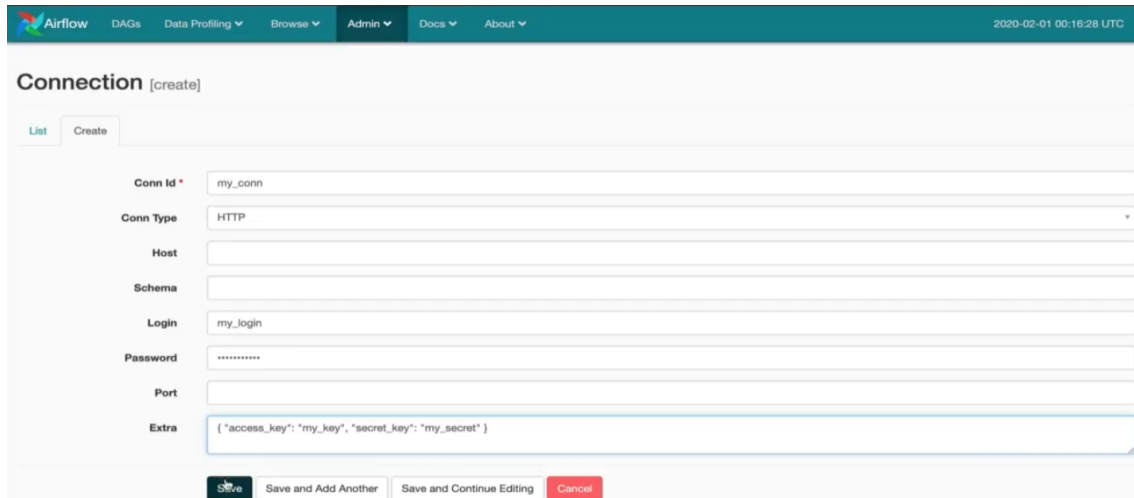
Além disso, esses alertas podem ser configurados para acionar notificações via e-mail, Slack, Microsoft Teams ou outras ferramentas de mensageria corporativa.

## 9. Security in Apache Airflow

O primeiro tópico de segurança se dá na questão de dados sensíveis sendo acessados via consulta no banco de dados, onde é possível recuperar senhas, logins,



informações extras, dentre outros. No exemplo abaixo, cria-se uma conexão com nome “my\_conn”, login sendo “my\_login”, senha “my\_password” e informações extras.

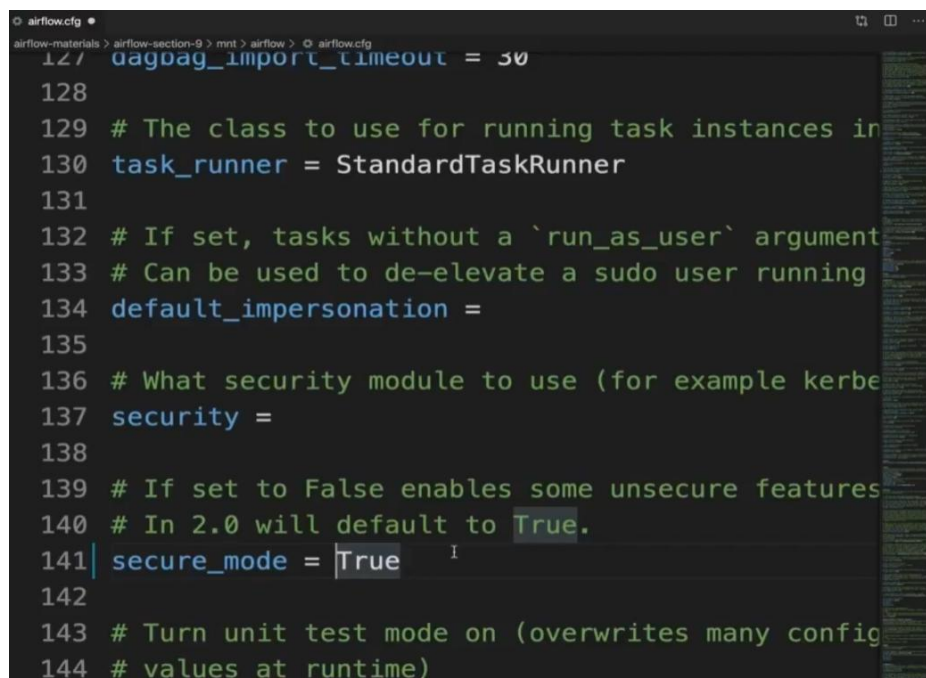


Conectando-se ao banco de dados rodando no nosso Airflow, é possível recuperar esses dados, como mostra a imagem abaixo, de forma completamente explícita e sem filtros.

```
airflow=# SELECT login,password,extra FROM connection WHERE conn_id='my_conn';
 login | password |
-----+-----+-----
 my_login | my_password | { \"access_key\": \"my_key\", \"secret_key\": \"my_secret\" }
(1 row)

airflow=#
```

O primeiro passo para tornar esses arquivos confidenciais, é alterar uma variável chamada de “secure\_mode” na configuração do Airflow para “True”. Em seguida reiniciar o Airflow.



Em seguida, caso não exista o pacote “crypto” no arquivo de montagem do container do Airflow, é imprescindível adicioná-lo. Em seguida reinicie o container para as alterações serem aplicadas.

```
airflow-section-9 > docker > airflow > Dockerfile > ...  
    locales \  
&& sed -i 's/^# en_US.UTF-8 UTF-8$/en_US.UTF-8 UTF-8/' /etc/locale.gen \  
&& locale-gen \  
&& update-locale LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8 \  
&& useradd -ms /bin/bash -d ${AIRFLOW_USER_HOME} ${AIRFLOW_USER} \  
&& pip install -U pip setuptools wheel \  
&& pip install pytz \  
&& pip install pyOpenSSL \  
&& pip install ndg-httpsclient \  
&& pip install pyasn1 \  
&& pip install apache-airflow[crypto,celery,redis] \  
&& pip install 'redis==3.2' \  
&& if [ -n "${PYTHON_DEPS}" ]; then pip install ${PYTHON_DEPS} \  
&& apt-get purge --auto-remove -yqq $buildDependencies \  
&& apt-get autoremove -yqq --purge \  
&& apt-get clean \  

```

Agora vem a parte mais importante que é a **Fernet\_key**, que é uma chave que impede que sejam feitas alterações, manipulações, dentre outros, em informações sensíveis contidas no banco de dados. É permitida apenas a alteração dessas informações caso o usuário tenha essa chave.

Para gerar essa chave, precisaremos nos conectar ao Webserver do nosso container. Faremos isso copiando o ID do webserver juntamente com o comando explicitado abaixo.

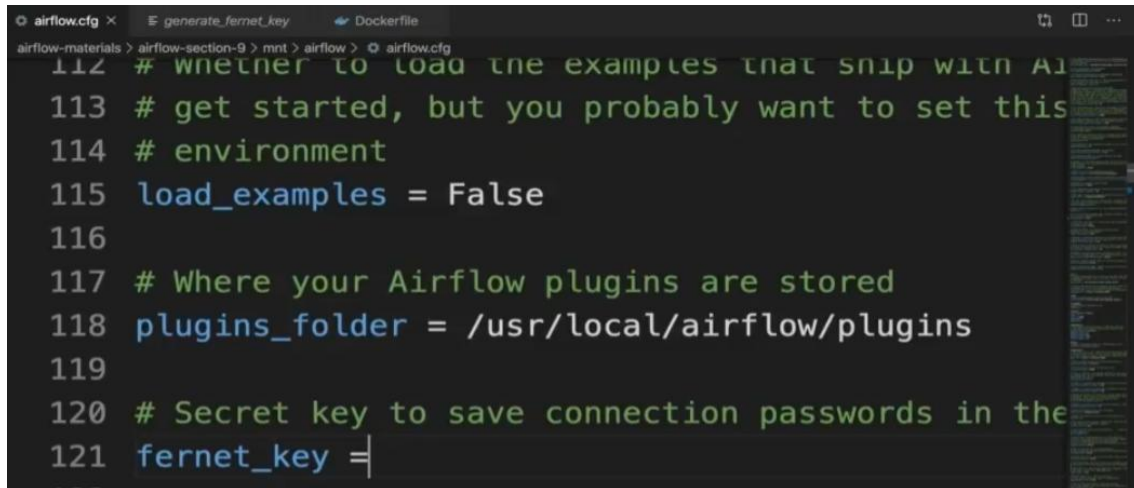
```
marclamberti/docker-airflow-security:1.10.7 "/entrypoint.  
sh webs..." 8 minutes ago Up 8 minutes (healthy) 5555/tcp, 8793/tcp, 0.  
0.0.0:8080->8080/tcp airflow-section-9_webserver_1  
327b5ba28795 postgres:9.6 "docker-entry  
point.s..." 8 minutes ago Up 8 minutes 5432/tcp  
airflow-section-9_postgres_1  
4899d02af264 redis:5.0.5 "docker-entry  
point.s..." 8 minutes ago Up 8 minutes 6379/tcp  
airflow-section-9_redis_1  
Marc@macbook-pro ~/airflow-materials/airflow-section-9 (master) $ docker exec -  
it a0e73db4be89 /bin/bash
```

Ao conectado, utilizaremos o comando: `<python -c "from cryptography.fernet import Fernet; print(Fernet.generate_key().decode())">` para gerar essa chave.



```
Marc@macbook-pro ~/airflow-materials/airflow-section-9 (master) $ docker exec -
it a0e73db4be89 /bin/bash
airflow@a0e73db4be89:~$ python -c "from cryptography.fernet import Fernet; prin
t(Fernet.generate_key().decode())"
d1APp_8RfZDvFon4RSU6HvDYG0hiFU1tS3wqNZEMM6Q=
airflow@a0e73db4be89:~$
```

Copie essa chave e a insira no arquivo de configuração do Airflow, na variável `fernet_key`



```
airflow-materials > airflow-section-9 > mint > airflow > airflow.cfg
112 # whether to load the examples that ship with Airflow
113 # get started, but you probably want to set this
114 # environment
115 load_examples = False
116
117 # Where your Airflow plugins are stored
118 plugins_folder = /usr/local/airflow/plugins
119
120 # Secret key to save connection passwords in the db
121 fernet_key =
```

Reinicie o container e a configuração estará feita. Realizando a consulta novamente a conexão criada anteriormente, o seguinte resultado é mostrado na tela:

```
airflow=# SELECT login,password,extra FROM connection WHERE conn_id='my_conn';
 login | password
-----+-----
 my_login | gAAAAABeNMnaNsatPisjx97nZhyz8KdqG360BLu6zi_KtivZxU10n0YhKqX80Lg1atY
2q30etBCPzP4UHvI4hct9lsr6fb3Ezw== | gAAAAABeNMnaF30wdttiC1ADY3XeY1G8Dltmotv5f2d
60PVd3-XsxbuNmKVVrVMKkalodPIRuLURNKvxioI3y7IhVodK4h63l1DsbqrQ3s2a8F6TFR_F3t-A5_
ubLZ4m02ywCTvRLvbKLyos7nkwkLoNELHEGAHdNQ==
(1 row)
```

Note que pode ser que você precise editar as conexões que já existiam anteriormente à adição da `fernet_key`.

Você pode também fazer a rotação de `Fernet_Key`, que consiste em basicamente trocar a `fernet_key` contida no arquivo de configuração do Airflow por uma outra, muitas vezes por questões de segurança, periodicidade (trocar a cada 7 dias), dentre outros. A variável ficaria assim:

```
Fernet_key = fernet_key_nova, fernet_key_old
```

Para efetuar a rotação dessas Keys, basta estar conectado ao Webserver e utilizar o comando `airflow rotate_fernet_key`.

```
airflow@a0e73db4be89:~$ python -c "from cryptography.fernet import Fernet; print(Fernet.generate_key().decode())"
7GDmTHBpnaKseRIeJZ747PUBYBg1T2MM0x30XX-w8=
airflow@a0e73db4be89:~$ airflow rotate_fernet_key
[2020-02-01 00:54:31,440] {settings.py:254} INFO - settings.configure_orm(): Using pool settings. pool_size=5, max_overflow=10, pool_recycle=1800, pid=311
airflow@a0e73db4be89:~$ exit
```

Em seguida você poderá excluir a `fernet_key_old` da variável.

Escondendo variáveis do Airflow UI: Para esconder variáveis na UI do Airflow, temos algumas palavras que são caracterizadas como sensíveis no Backend do Airflow. Ao inserir elas no nome de uma variável, imediatamente o seu valor será escondido.

```
DEFAULT_SENSITIVE_VARIABLE_FIELDS = (
    'password',
    'secret',
    'passwd',
    'authorization',
    'api_key',
    'apikey',
    'access_token',
)
```

Por exemplo, no caso abaixo nenhuma palavra sensível é inserida, o que permite a sua visualização na UI sem qualquer filtro.

**Variable** [create]

List Create

Key \* my\_var



Val my\_value

Save Save and Add Another Save and Continue Editing Cancel

**Variables**

Choose file No file chosen Import Variables

List (1) Create Add Filter With selected Search: key, val, is\_encrypted

	Key	Val	Is Encrypted
<input type="checkbox"/>  	my_var	my_value	<input checked="" type="checkbox"/>

Agora, quando inserimos alguma das palavras classificadas como sensível, o campo “val” ficará escondido, similarmente a visualização de uma senha, com “\*”.

Variable [create]

List Create

Key \* my\_password

Val \*\*\*\*\*

Save Save and Add Another Save and Continue Editing Cancel

Variables

Choose file No file chosen Import Variables

List (2) Create Add Filter With selected Search: key, val, is\_encrypted

	Key	Val	Is Encrypted
<input type="checkbox"/>	my_password	*****	<input checked="" type="checkbox"/>
<input type="checkbox"/>	my_var	my_value	<input checked="" type="checkbox"/>

### Autenticação no Airflow e Filtro de Usuário

Podemos também adicionar autenticação na UI do Airflow para permitir apenas usuários específicos de acessarem. É importante adicionar esse tipo de configuração dado em um ambiente de trabalho em que várias pessoas utilizam a UI, para ter maior controle sobre as ações dos usuários, bem como limitar o acesso as DAGs de cada um.

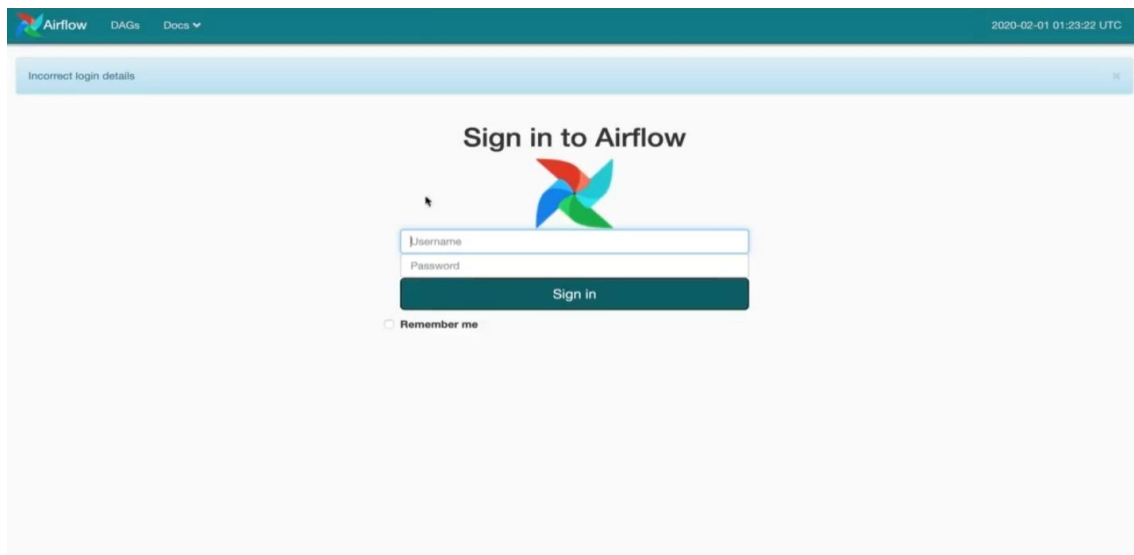
Para isso, faremos a alteração da variável “authenticate” para “True” e adicionar a variável “auth\_backend” com valor igual a “airflow.contrib.auth.backends.password\_auth”.

```
251 access_logfile = -
252 error_logfile = -
253
254 # Expose the configuration file in the web server
255 # This is only applicable for the flask-admin based web UI (non F
256 # In the FAB-based web UI with RBAC feature,
257 # access to configuration is controlled by role permissions.
258 expose_config = False
259
260 # Set to true to turn on authentication:
261 # https://airflow.apache.org/security.html#web-authentication
262 authenticate = True
263 auth_backend = airflow.contrib.auth.backends.password_auth
```

Adicionaremos também, no arquivo de montagem do container Dockerfile, a instalação do Flask, que fará o gerenciamento de usuários para nós.

```
Dockerfile x
airflow-section-9 > docker > airflow > Dockerfile > ...
V 1111 \
rsync \
netcat \
locales \
&& sed -i 's/^# en_US.UTF-8 UTF-8$/en_US.UTF-
&& locale-gen \
&& update-locale LANG=en_US.UTF-8 LC_ALL=en_U
&& useradd -ms /bin/bash -d ${AIRFLOW_USER_HO
&& pip install -U pip setuptools wheel \
&& pip install pytz \
&& pip install flask-bcrypt \
&& pip install pyOpenSSL \
&& pip install ndg-httpsclient \
&& pip install pyasn1 \
&& pip install apache-airflow[crypto,celery,p
&& pip install 'redis==3.2' \
```

Reiniciando o container e acessando a UI do Airflow, nos depararemos com uma tela de login, que antes não existia. Note que não existe nenhum usuário por enquanto, faremos a criação de um na sequência.



Para criar um usuário no Airflow, precisaremos nos conectar ao Webserver novamente e rodar o seguinte comando:

```
<
import airflow
from airflow import models, settings
from airflow.contrib.auth.backends.password_auth import PasswordUser
user = PasswordUser(models.User())
user.username = 'admin'
```

```

user.email = 'admin@airflow.com'
user.password = 'admin'
session = settings.Session()
session.add(user)
session.commit()
session.close()
exit()
>

```

```

74d0a76d908a      marclamberti/docker-airflow-security:1.10.7   "/entrypoint.
sh webs..."    37 seconds ago      Up 35 seconds (healthy)    5555/tcp, 8793/tcp, 0
.0.0.0:8080->8080/tcp    airflow-section-9_webserver_1
4d7a550fb59e      postgres:9.6                                          "docker-entry
point.s..."    38 seconds ago      Up 36 seconds              5432/tcp
                        airflow-section-9_postgres_1
728860a46b92      redis:5.0.5                                          "docker-entry
point.s..."    38 seconds ago      Up 36 seconds              6379/tcp
                        airflow-section-9_redis_1
Marc@macbook-pro ~/airflow-materials/airflow-section-9 (master) $ docker exec -
it 74d0a76d908a /bin/bash
airflow@74d0a76d908a:~$

```

Na imagem abaixo, criamos então o usuário admin, com senha admin e outras demais configurações.

```

>>> import airflow
[2020-02-01 01:24:46,638] {settings.py:254} INFO - settings.configure_orm(): Us
ing pool settings. pool_size=5, max_overflow=10, pool_recycle=1800, pid=87
>>> from airflow import models, settings
>>> from airflow.contrib.auth.backends.password_auth import PasswordUser
>>> user = PasswordUser(models.User())
>>> user.username = 'admin'
>>> user.email = 'admin@airflow.com'
>>> user.password = 'admin'
>>> session = settings.Session()
>>> session.add(user)
>>> session.commit()
>>> session.close()
>>> exit()
airflow@74d0a76d908a:~$

```

Pronto, a configuração está feita do usuário. Mas como poderemos filtrar apenas para aparecer as DAGs do respectivo usuário que fez login? Para isso, precisaremos alterar a variável “filter\_by\_owner” para “True” no arquivo de configuração do Airflow. Reinicie o servidor na sequência.

Antes da configuração do filtro por usuário, as DAGs apareciam assim para o usuário “admin” (Note a coluna “Owner”):

**DAGs**

Search:

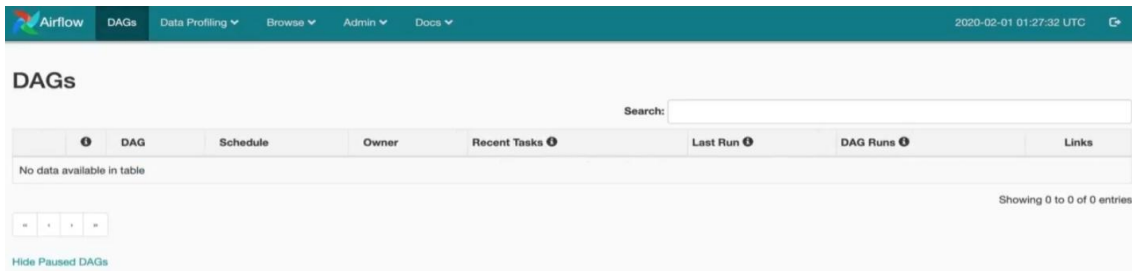
	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	finance_dag	daily	airflow				
	marketing_dag	daily	airflow				

Showing 1 to 2 of 2 entries

Hide Paused DAGs

Após a configuração, as DAGs já não estão mais ali, justamente pois ele não é dono de nenhuma DAG por enquanto.

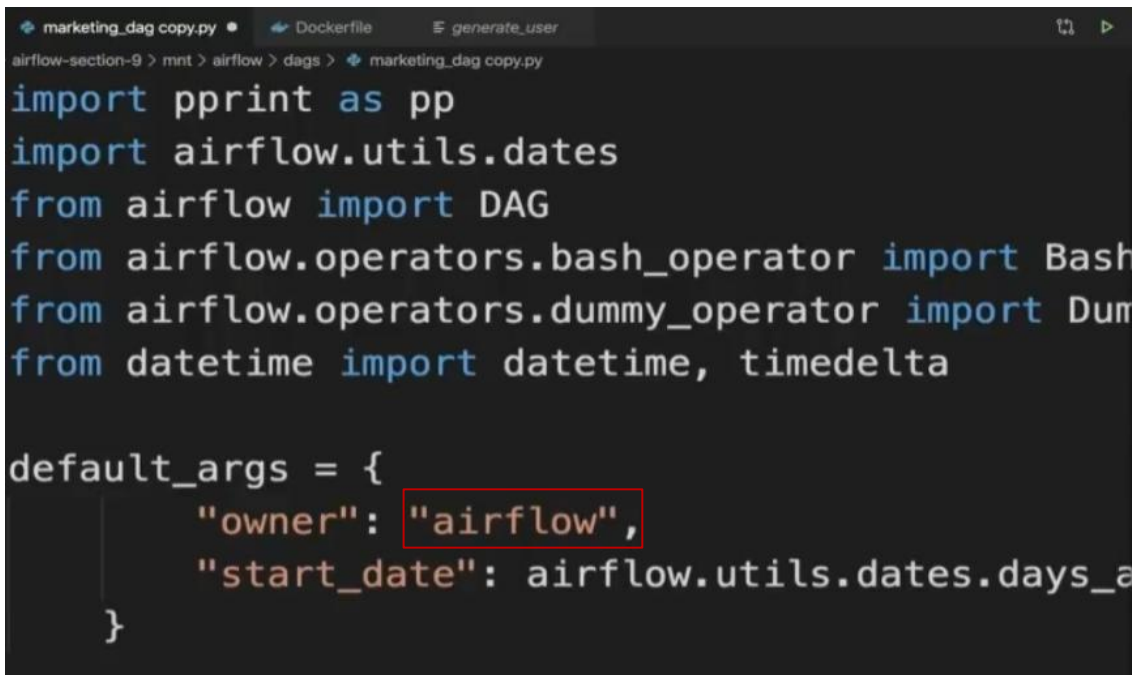




The image shows the Airflow web interface. At the top, there's a navigation bar with 'Airflow' logo and tabs for 'DAGs', 'Data Profiling', 'Browse', 'Admin', and 'Docs'. The current date and time are '2020-02-01 01:27:32 UTC'. Below the navigation bar, the 'DAGs' section is active. It features a search bar and a table with columns: DAG, Schedule, Owner, Recent Tasks, Last Run, DAG Runs, and Links. The table is currently empty, with a message 'No data available in table' and 'Showing 0 to 0 of 0 entries'. There are also pagination controls and a 'Hide Paused DAGs' link.

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
No data available in table						

Para que apareça alguma DAG para o usuário “admin”, é necessário que ele crie, ou uma do zero, ou altere o proprietário das DAGs que já existem, como no caso abaixo:

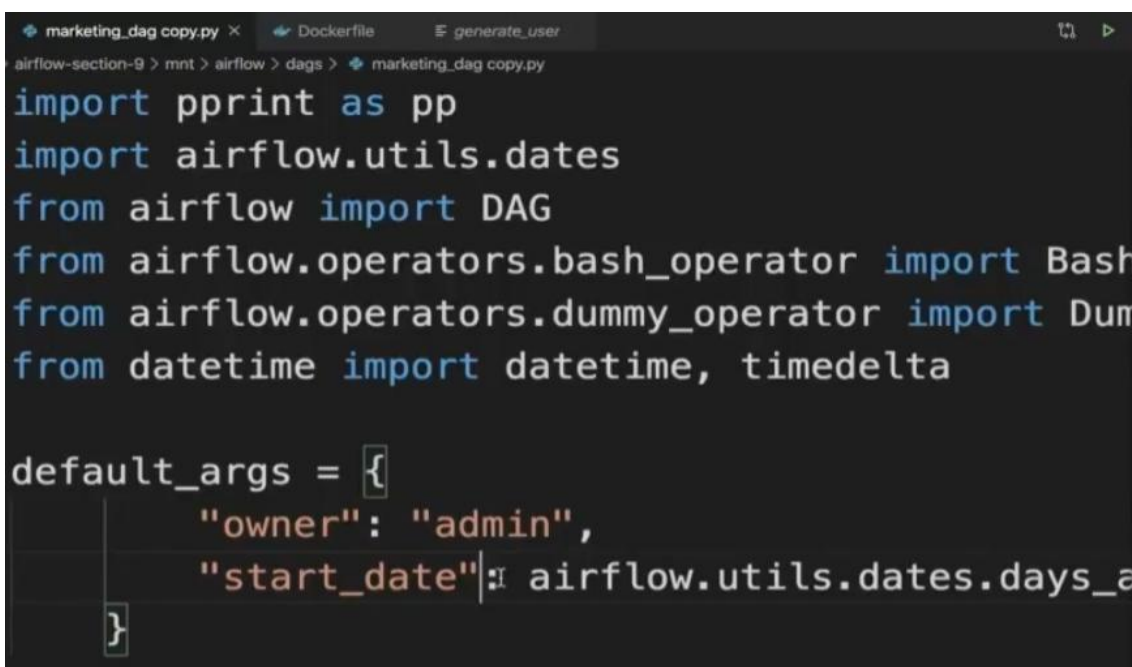


The image shows a code editor with the file 'marketing\_dag copy.py'. The code defines a DAG with the following imports and default arguments:

```
import pprint as pp
import airflow.utils.dates
from airflow import DAG
from airflow.operators.bash_operator import Bash
from airflow.operators.dummy_operator import DummyOperator
from datetime import datetime, timedelta

default_args = {
    "owner": "airflow",
    "start_date": airflow.utils.dates.days_ago(1),
}
```

Após alterar o proprietário para “admin”, a DAG irá aparecer para o usuário.

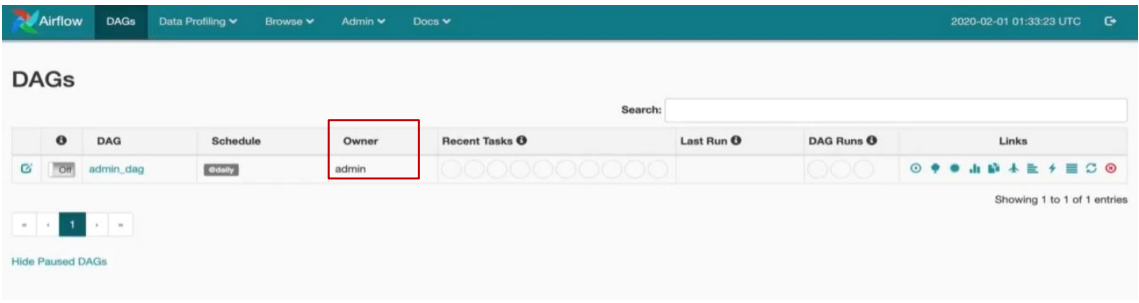


The image shows the same code editor with the file 'marketing\_dag copy.py'. The code is identical to the previous one, but the owner has been changed to 'admin' in the default arguments:

```
import pprint as pp
import airflow.utils.dates
from airflow import DAG
from airflow.operators.bash_operator import Bash
from airflow.operators.dummy_operator import DummyOperator
from datetime import datetime, timedelta

default_args = {
    "owner": "admin",
    "start_date": airflow.utils.dates.days_ago(1),
}
```

Note que o Owner foi alterado para “admin”.



The screenshot shows the Apache Airflow web interface. At the top, there's a navigation bar with the Airflow logo and tabs for DAGs, Data Profiling, Browse, Admin, and Docs. The current date and time are 2020-02-01 01:33:23 UTC. Below the navigation bar, the 'DAGs' section is displayed. A search bar is present. The main table lists DAGs with columns: DAG, Schedule, Owner, Recent Tasks, Last Run, DAG Runs, and Links. The 'admin\_dag' is listed with a 'daily' schedule and 'admin' as the owner. The 'Owner' column for 'admin\_dag' is highlighted with a red box. Below the table, there's a pagination control showing '1' of 1 entries and a 'Hide Paused DAGs' link.

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
admin_dag	daily	admin				

## Referencias

[Apache Airflow: The Hands-On Guide \(Seção 5 à 9\)](#)