*Diana Margarian*

# Rapport de Stage

# Gestion des Fichiers et Synchronisation avec S3

## Introduction

Durant ce stage, j'ai eu l'opportunité de travailler sur un projet de gestion et de synchronisation de fichiers avec le stockage cloud d'Amazon (S3). Le projet inclut plusieurs services interconnectés, comme un agent qui gère les fichiers locaux, un système de base de données pour stocker des métadonnées, ainsi que la génération d'URLs présignées pour l'accès direct aux fichiers sur S3. Tous ces services sont conteneurisés à l'aide de Docker et orchestrés par Docker Compose.

En plus de cette mission principale, j'ai également travaillé sur la documentation du projet à l'aide d'Obsidian.

Par la suite, j'ai réalisé une étude comparative sur Docusaurus afin de tester ses fonctionnalités et de le comparer à MediaWiki, la solution actuellement utilisée par l'entreprise pour la documentation technqiue. Ce rapport représente les différentes technologies utilisées, ainsi que les choix architecturaux réalisés pour la gestion et la synchronisation des données.

## Objectifs du Projet

L'objectif principal de ce projet était de mettre en place une solution pour synchroniser des fichiers locaux avec un **bucket S3**, en utilisant une base de données PostgreSQL pour enregistrer les informations relatives aux fichiers et faciliter leur gestion. Le système devait également permettre de créer des **URLs présignées** pour accéder directement à des fichiers stockés sur S3.

En résumé, les principales fonctionnalités à implémenter étaient les suivantes :

- Synchronisation des fichiers locaux avec un stockage S3.
- Stockage des métadonnées relatives aux fichiers dans une base de données PostgreSQL.
- Création d'URLs présignées pour l'accès aux fichiers sur S3.

# Technologies et Outils Utilisés

### Docker & Docker Compose

Le projet repose sur Docker, une technologie permettant de conteneuriser les différents services afin de simplifier leur déploiement et gestion. Docker Compose est utilisé pour orchestrer ces services.

- **Dockerfile** : Le projet contient un fichier Dockerfile qui définit l'image de base (Python 3.9-slim), installe les dépendances et configure l'exécution du script Python principal.
- **Docker Compose** : Le fichier docker-compose.yml définit l'architecture multi-containers avec les services suivants :
  - **agent-portail** : Le service qui gère les fichiers locaux et leur synchronisation avec S3.
  - **postgres** : Base de données PostgreSQL qui stocke les métadonnées des fichiers.
  - **urls-generator** : Service qui génère les URLs présignées pour l'accès aux fichiers S3.
  - **grafana** : Service de monitoring des métriques système.

### AWS S3 et PostgreSQL

- **S3** : Amazon S3 est utilisé pour stocker les fichiers, permettant une gestion cloud scalable et sécurisée. Le service agent-portail se charge de la synchronisation des fichiers vers le bucket S3.
- **PostgreSQL** : Une base de données PostgreSQL est utilisée pour stocker les métadonnées associées aux fichiers, telles que le nom du fichier, son emplacement, sa date de création, etc.

### Logs et Journalisation

La gestion des logs est réalisée via un système de journalisation Python (logging). Les logs peuvent être redirigés soit vers la console (stdout), soit vers un fichier (logfile), selon les variables d'environnement.

# Développement du Projet

### Gestion des Variables d'Environnement

Les variables d'environnement jouent un rôle clé dans la configuration des différents services du projet. Elles permettent de rendre le code flexible et adaptable à différents environnements (local, développement, production). Par exemple, les clés d'accès à AWS, le nom du bucket S3, ainsi que les informations de connexion à la base de données PostgreSQL sont stockées dans des variables d'environnement. Cela permet de centraliser et de sécuriser les informations sensibles.

Les variables utilisées dans le projet incluent :

- **AWS_REGION**, **AWS_BUCKET_NAME**, **AWS_ACCESS_KEY_ID**, **AWS_SECRET_ACCESS_KEY** pour interagir avec AWS S3.

- **DB_HOST**, **DB_NAME**, **DB_USER**, **DB_PASSWORD** pour la connexion à PostgreSQL.

- **LOG_METHOD**, **LOG_PATH** pour la gestion des logs.

## Synchronisation des Fichiers avec S3

La synchronisation des fichiers locaux avec S3 est gérée par le service agent-portail. Voici les principales étapes :

- **Vérification des fichiers locaux** : Le programme examine les fichiers locaux dans un répertoire spécifié et les compare à ceux stockés sur S3.

- **Calcul de la somme de contrôle MD5** : Afin de s'assurer que les fichiers sont correctement synchronisés, une somme de contrôle MD5 est calculée pour chaque fichier, tant localement que sur S3.

-,**Upload vers S3** : Les fichiers qui sont nouveaux ou modifiés (en fonction de la somme MD5) sont envoyés vers le bucket S3.

- **Suppression des fichiers obsolètes** : Les fichiers qui ne sont plus présents localement sont supprimés de S3 pour maintenir la synchronisation.

## Base de Données PostgreSQL

La base de données PostgreSQL est utilisée pour stocker les métadonnées des fichiers (nom, chemin, date de dernière modification, etc.). La connexion à la base de données se fait grâce à la fonction connect_db(), qui récupère les informations nécessaires depuis les variables d'environnement.

## Génération d'URLs Présignées

Le service urls-generator génère des **URLs présignées** qui permettent à des utilisateurs d'accéder directement à des fichiers stockés dans S3, sans avoir besoin de credentials AWS. Ces URLs sont temporaires et peuvent être configurées pour expirer après un certain délai.

## Gestion des Logs

Un système de logs a été mis en place pour tracer l'exécution du programme. Selon la variable d'environnement LOG_METHOD, les logs peuvent être affichés dans la console ou enregistrés dans un fichier. Ce système permet une gestion des erreurs et des succès, facilitant ainsi le suivi du bon fonctionnement du programme. Le niveau des logs (INFO, ERROR, SUCCESS) est également configurable.

# Documentation du Projet

Après avoir finalisé la mission principale, j'ai documenté le projet en utilisant **Obsidian**, un outil de prise de notes basé sur Markdown. Cette documentation comprend la description des fonctions importantes, les instructions d'installation etc.

Par la suite, j'ai reçu une nouvelle mission : tester **Docusaurus** en vue d'une potentielle migration de MediaWiki vers Docusaurus pour la documentation technique de l'entreprise. Mon travail a consisté à :

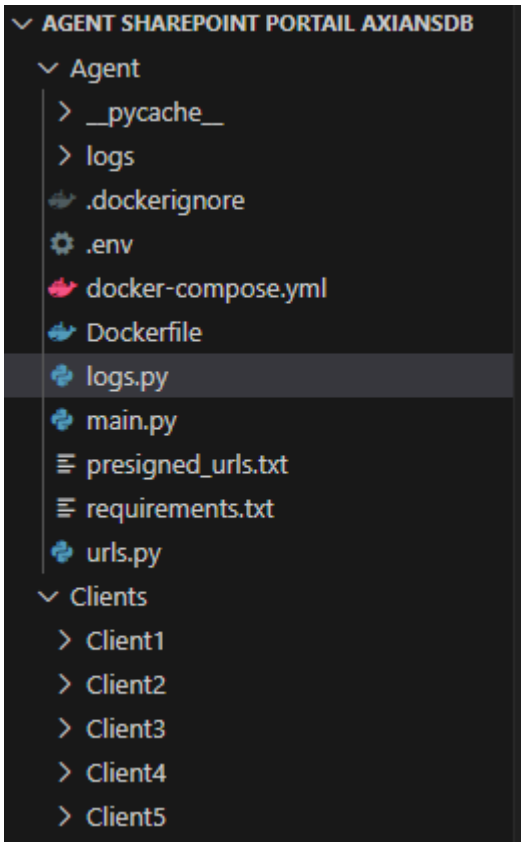
- Explorer les fonctionnalités de Docusaurus.
- Comparer les avantages et inconvénients.
- Tester l'utilisation de **Markdown** pour structurer la documentation.
- Étudier l'organisation des fichiers et répertoires (dossiers, sous-dossiers, fichiers).
- Explorer les possibilités d'extension via **HTML, Node.js, etc**.
- Tester l'installation et l'ajout de plugins pour enrichir la documentation.
- Préparer une présentation pour l'entreprise.

# Conclusion

Ce stage m'a permis d'acquérir une solide compréhension des technologies suivantes : **Docker, AWS S3, PostgreSQL et Python**. La mise en place d'un environnement de travail flexible via Docker et la gestion des variables d'environnement m'ont permis de créer une solution robuste et adaptable pour la synchronisation des fichiers entre un répertoire local et le cloud.

En parallèle, l'étude de **Docusaurus** m'a offert une expérience précieuse dans l'évaluation et le test de nouvelles solutions pour la documentation technique. Cette mission m'a permis de comprendre les défis liés à la documentation dans un environnement technique et d'explorer des solutions modernes basées sur Markdown.

**Quelques impressions écrans :**


AGENT SHAREPOINT PORTAIL AXIANSDB
Agent
__pycache__
logs
.dockerignore
.env
docker-compose.yml
Dockerfile
logs.py
main.py
presigned_urls.txt
requirements.txt
urls.py
Clients
Client1
Client2
Client3
Client4
Client5

Agent > docker-compose.yml

```yaml
Run All Services
services:
  Run Service
  agent-portail:
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      - AWS_REGION=${AWS_REGION}
      - AWS_BUCKET_NAME=${AWS_BUCKET_NAME}
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - LOG_METHOD=CONTAINER
      - LOG_PATH=/app/logs/agent-portail.log
      - LOG_LEVEL=INFO
      - DB_HOST=postgres
      - DB_NAME=portail_axiansdb
      - DB_USER=xx
      - DB_PASSWORD=xx
    volumes:
      - ../Clients:/sharepoint/
      - ./logs/agent-portail.log:/app/logs/
    stdin_open: true
    tty: true

  Run Service
  postgres:
    image: postgres:latest
    environment:
      POSTGRES_DB: portail_axiansdb
      POSTGRES_USER: xx
      POSTGRES_PASSWORD: xx
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  Run Service
  urls-generator:
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      - AWS_REGION=${AWS_REGION}
      - AWS_BUCKET_NAME=${AWS_BUCKET_NAME}
```

```yaml
  ▷ Run Service
  urls-generator:
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      - AWS_REGION=${AWS_REGION}
      - AWS_BUCKET_NAME=${AWS_BUCKET_NAME}
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
    volumes:
    - ./presigned_urls.txt:/app/presigned_urls.txt
    command: python /app/urls.py

  ▷ Run Service
  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana
    depends_on:
      - postgres

volumes:
  grafana-storage:
  postgres_data:
```

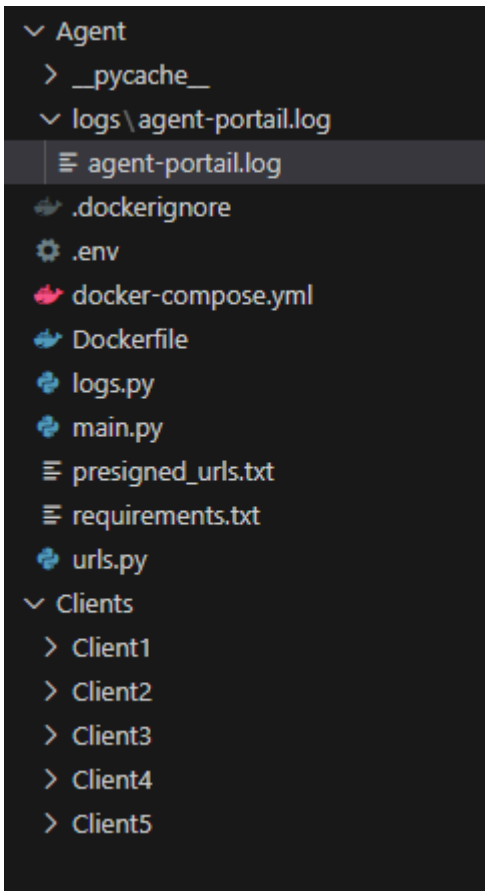
Agent > Dockerfile > ...

```dockerfile
# Utilise l'image Python 3.9-slim comme base
FROM python:3.9-slim

# Défini le répertoire de travail à /app dans le conteneur
WORKDIR /app

# Copie le fichier requirements.txt dans le répertoire de travail
COPY requirements.txt /app/

# Installer les dépendances à partir du fichier requirements.txt
RUN pip install --no-cache-dir -r requirements.txt


# Copie tous les fichiers du répertoire local dans le répertoire de travail du conteneur
COPY . /app/

# Lance le programme Python
CMD ["python", "/app/main.py"]

```

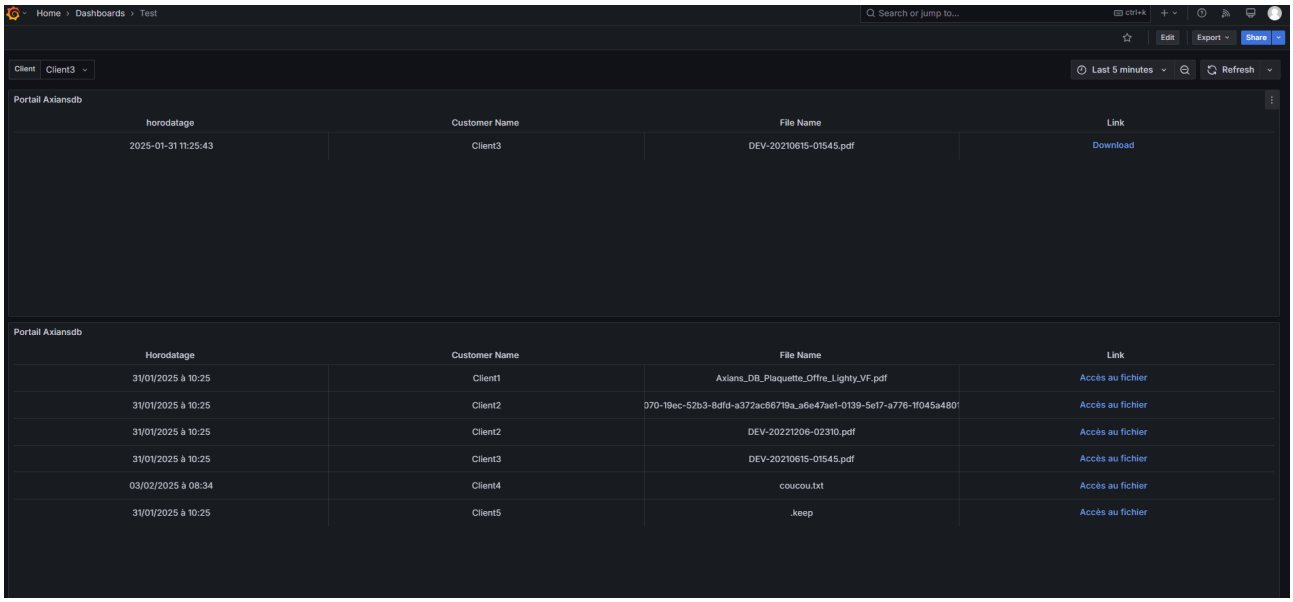
Agent > main.py > ...

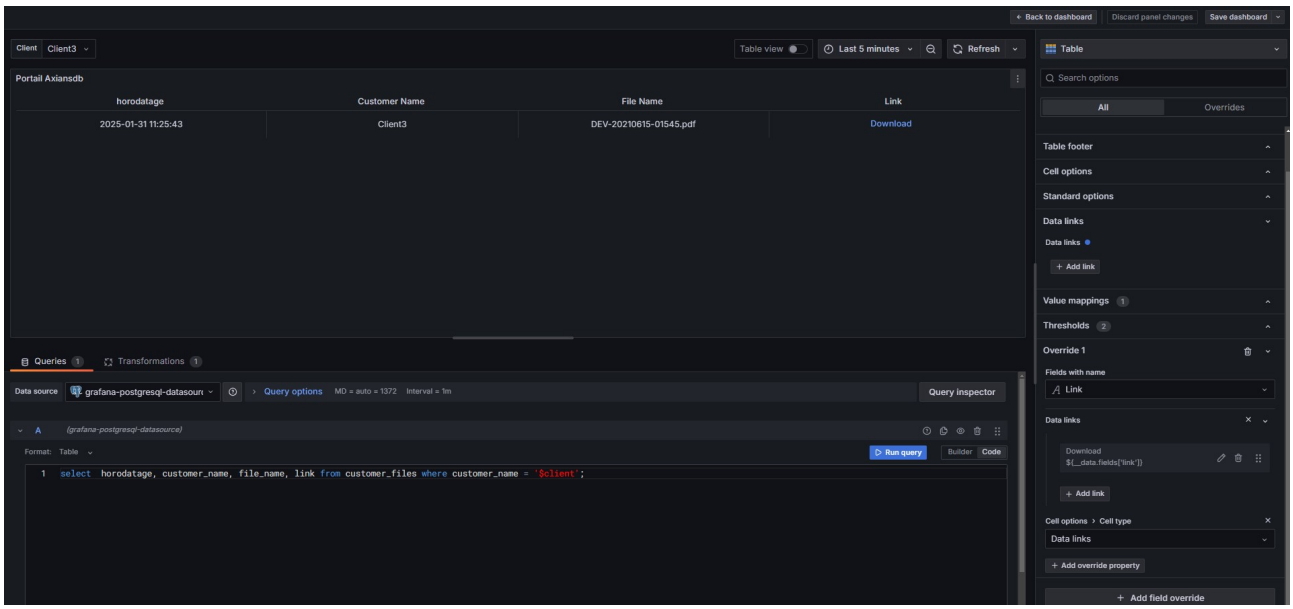
```python
import hashlib
import os
import boto3
import time
from botocore.exceptions import ClientError
from datetime import datetime
import sys
from logs import logger, log_message

# Affichage des variables d'environnement pour validation
def check_environment_variables():...

# Fonction pour vérifier les chemins locaux requis
def check_required_local_paths(local_paths):...

# Fonction de vérification de la connexion S3
def check_s3_connection(s3):...

# Appel des fonctions de vérification au début du programme
check_environment_variables()

# Configuration AWS
AWS_REGION = os.getenv('AWS_REGION', 'eu-west-3')
bucket_name = os.getenv('AWS_BUCKET_NAME', 'axiansdbdata')
local_directory = '/sharepoint/'
required_folder_name = 'Portail - Documents partagés'

# Chargement des variables d'environnement depuis le fichier .env
AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')

# Vérifier la présence des répertoires locaux nécessaires
check_required_local_paths([local_directory])

# Initialisation du client S3
s3 = boto3.client(
    's3',
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
    region_name=AWS_REGION
)

# Vérifier la connexion à S3
check_s3_connection(s3)
```

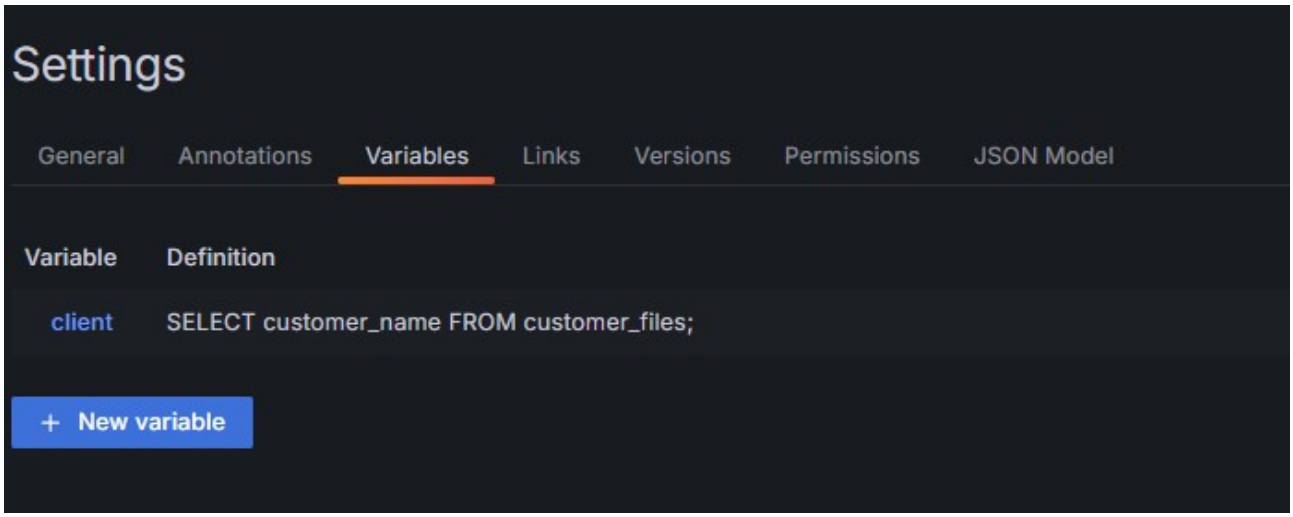
```python
# Vérifier la connexion à S3
check_s3_connection(s3)

# Calcul MD5 pour un fichier local
def calculate_md5(file_path):...

# Fonction de validation et de configuration du mode de logging selon la variable d'environnement LOG_METHOD
def check_log_method():...

# Appeler la vérification des logs en début de programme
check_log_method()

# Vérification des fichiers locaux et comparaison avec ceux sur S3
def get_s3_file_md5(s3_key):...

# Récupération de la liste des fichiers sur S3 pour un client donné
def get_files_from_s3_for_client(customer_folder):...

# Vérification de la présence du dossier client en local
def get_customers_list(local_path):...

# Vérification de la présence du dossier requis sur le dossier client
def ensure_local_directory_is_present(directory_path, required_folder_name, createFolder=True):...

# Récupération de la liste des fichiers présents sur S3 dans le bucket
def get_s3_files():...

# Suppression des fichiers qui existent sur S3 mais pas localement
def delete_files_from_s3(local_directory, customers_list):...

# Fonction de synchronisation des fichiers vers S3
def synchronize_with_s3(customers_list):...

# Lancement du programme
if __name__ == "__main__":...
```




| Name | Container ID | Image | Port(s) | CPU (%) | Last started | Actions |
|---|---|---|---|---|---|---|
| my-website | - | - | - | 0% | 1 hour ago | |
| docusaurus-1 | 6e635edd9819 | my-website-docusaurus | 3000:3000 | 0% | 1 hour ago | |
| agent | - | - | - | 1.09% | 10 minutes ago | |
| postgres-1 | 23a0c9bb5880 | postgres:latest | 5432:5432 | 0% | 10 minutes ago | |
| grafana-1 | 6f15f5a2f3a0 | grafana/grafana | 3000:3000 | 1.09% | 10 minutes ago | |
| urls-generator-1 | 88f98d032791 | agent-urls-generator | | 0% | 10 minutes ago | |
| agent-portail-1 | e3db98d0af6d | agent-agent-portail | | 0% | 10 minutes ago | |




Back to dashboard
Discard panel changes
Save dashboard
Client
Client3
Table view
Last 5 minutes
Refresh
Portail Axiansdb
horodatage
Customer Name
File Name
Link
2025-01-31 11:25:43
Client3
DEV-20210615-01545.pdf
Download
Table
Search options
All
Overrides
Table footer
Cell options
Standard options
Data links
Data links
+ Add link
Value mappings
Thresholds
Override 1
Fields with name
Link
Data links
Download
${__data.fields['link']}
+ Add link
Cell options › Cell type
Data links
+ Add override property
+ Add field override
Queries
Transformations
Data source
grafana-postgresql-datasoun
Query options
MD = auto = 1372
Interval = 1m
Query inspector
A
(grafana-postgresql-datasource)
Format: Table
Run query
Builder
Code
1 select horodatage, customer_name, file_name, link from customer_files where customer_name = '$client';


Settings
General
Annotations
Variables
Links
Versions
Permissions
JSON Model
Variable
Definition
client
SELECT customer_name FROM customer_files;
+ New variable