Airflow

Airflow est spécifiquement conçu pour orchestrer des pipelines de données et des flux de travail complexes. Ses points forts principaux sont :

Support natif pour la planification et les dépendances entre les tâches Opérateurs intégrés pour le traitement des données et les services cloud Définition des flux de travail basée sur des Graphes Acycliques Dirigés (DAG) Interface utilisateur riche pour la surveillance des pipelines de données Meilleure gestion des nouvelles tentatives de tâches et du remplissage rétroactif

Considérez Airflow comme spécialisé dans les flux de travail de données (ETL, pipelines d'apprentissage automatique, etc.) tandis que Jenkins est plus généraliste pour les flux de travail de développement logiciel (construction, test, déploiement d'applications).

Voici le Docker compose pour lancer Airflow

Airflow Docker Compose Setup: Questions and Answers

Question 1: What services are defined in this Docker Compose setup?

Answer: The main services defined in this setup are:

- mlflow (ML experiment tracking)
- postgres (database)
- redis (message broker)
- airflow-webserver (Airflow UI)
- airflow-scheduler (DAG scheduling)
- airflow-worker (task execution)
- airflow-triggerer (trigger management)
- airflow-init (initialization service)
- airflow-cli (command line interface)
- flower (optional Celery monitoring)

Question 2: How are the Airflow images being built in this

setup?

Answer: Instead of using the default Apache Airflow image, this setup builds a custom image using a Dockerfile.airflow. This is specified in the build section under x-airflow-common:

```yaml
build:
  context: .
  dockerfile: Dockerfile.airflow
```

This approach allows for adding additional dependencies (like sklearn) to the base Airflow image.

Question 3: What executor is being used in this Airflow setup and what components does it require?

Answer: This setup uses the CeleryExecutor as specified in the environment variables:

```yaml
AIRFLOW__CORE__EXECUTOR: CeleryExecutor
```

This requires:

- Redis (as message broker)
- PostgreSQL (as result backend)
- Celery workers (for distributed task execution)

Question 4: How are the DAGs and data being mounted into the Airflow containers?

Answer: The setup uses volume mounting to make local directories available inside the containers:

```yaml
volumes:
  - /Users/alexis/work/ademe_mlops/dags:/opt/airflow/dags
  - /Users/alexis/work/ademe_mlops/data:/opt/airflow/data
```

Local DAGs and data directories are mounted to `/opt/airflow/dags` and `/opt/airflow/data` respectively inside the containers.

Question 5: What health checks are implemented in this setup and why are they important?

Answer: Health checks are implemented for multiple services:

- Webserver: Checks `http://localhost:8080/health`
- Scheduler: Checks `http://localhost:8974/health`
- Redis: Uses `redis-cli ping`
- PostgreSQL: Uses `pg_isready`
- Worker: Checks Celery worker status

These health checks ensure that services are running properly and help with container orchestration and automatic restarts when needed.

Question 6: What default credentials are set for the Airflow web interface?

Answer: The default credentials are set in the airflow-init service:

```yaml
_AIRFLOW_WWW_USER_USERNAME: ${_AIRFLOW_WWW_USER_USERNAME:-airflow}
_AIRFLOW_WWW_USER_PASSWORD: ${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
```

Both username and password default to "airflow" if not overridden by environment variables.

Question 7: What resource checks are performed during initialization?

Answer: The airflow-init service checks for:

- Memory: Minimum 4GB required
- CPU: Minimum 2 CPUs recommended
- Disk Space: Minimum 10GB recommended

If these requirements aren't met, warning messages are displayed but the service will still start.

Question 8: How is MLflow integrated into this setup?

Answer: MLflow is integrated as a separate service:

```yaml
mlflow:
  build:
    context: .
    dockerfile: Dockerfile.mlflow
  ports:
    - "5001:5000"
```

It's built from a custom Dockerfile and exposes port 5001 on the host machine, mapping to MLflow's

default port 5000 internally.

Question 9: What ports are exposed in this setup?

Answer: The main exposed ports are:

- 8080: Airflow webserver UI
- 5001: MLflow server
- 5555: Flower (Celery monitoring, optional)
- 6379: Redis (exposed internally only)

Question 10: How is data persistence handled in this setup?

Answer: Data persistence is handled through named volumes:

```yaml
volumes:
  postgres-db-volume:  # For PostgreSQL data
  mlflow-artifacts:    # For MLflow artifacts
```

Additionally, several directories are mounted from the host: - logs - dags - plugins - config - data

This ensures that data persists across container restarts and rebuilds.