AutoML CI/CD Pipeline with Apache Airflow and GitLab

Ayman Osman Abubaker, Iosif Koen

8th May 2023

We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

1 Introduction

Coninuous Integration and Continuous Deployment practices are important for efficient software delivery in DevOps, enabling regular code integration, early issue identification, and automated deployment. As machine learning (ML) gains importance, the demand for automated ML pipelines has likewise grown. ML pipelines involve data processing, model training, evaluation, and deployment, often with complex dependencies and specialized orchestration tools which means implementing CI/CD poses extra challenges.

Apache Airflow and GitLab are popular tools for automating CI/CD ML pipelines. Airflow is an open-source platform for orchestrating data workflows using Directed Acyclic Graphs (DAGs), and GitLab is a web-based DevOps platform with features like version control, issue tracking, code review, and CI/CD. In this essay, we will demonstrate how to combine the power of Apache Airflow and GitLab to build a fully automated CI/CD ML pipeline, enabling data scientists and engineers to streamline their ML development process and achieve better collaboration, efficiency, and reliability in their projects.

2 Background

2.1 CI and CD in ML: Concepts and Challenges

In the context of ML, CI and CD present unique challenges due to the distinct nature of ML workflows. This section delves deeper into the concepts of CI and CD in ML and discuss the challenges associated with their implementations.

2.1.1 Continuous Integration in ML

CI is the practice of continuously merging code updates into a central repository, which is followed by automated testing to ensure the integrity of the codebase. The main goal of CI is to early on identify and resolve errors in the development process, reducing the risk of integration problems later on which in turn improves the quality of the software.

In ML, CI involves the automation of several key processes including but not limited to:

1. **Data validation**: This process consists of a series of checks to ensure that the data used for training and evaluation meets specific quality requirements. These requirements can for example be adherence to some schema constraint or data amount (1). .

2. **Feature engineering**: Feature engineering is the processing of raw data that is done with the intent of boosting model performance. Automating this process ensures that the latest data and features are incorporated into the model training process (2).

3. **Model training**: Automating model training allows developers to iterate on model architectures and hyperparameters faster which ultimately leads to higher quality models. This should also be done whenever code or data is updated.(3) .

4. **Model evaluation**: Evaluating the performance of trained models using relevant metrics, such as accuracy, F1-score, or mean squared error. Automating this process guarantees that the trained models meet some performance criteria before being deployed for production (4) .

2.1.2 Continuous Deployment in ML

CD is the practice of automatically deploying the code that is tested in the CI part into the relevant production environment. The aim here is to reduce the time between code changes and their deployment so that we can have faster iteration cycles and also make sure that any new features or improvements to our application are delivered to the users quickly.

For ML, CD involves these following key processes:

1. **Model packaging**: Trained models become converted into a relevant and suitable format for deployment such as SavedModel, TensorFlow or ONNX format. This step is important as it ensures that models can be deployed to a variety of runtime environments, such as edge devices, cloud services etc. (5).

2. **Model deployment**: The process of deploying these packaged models to the production systems, for example RESTful APIs or batch processing systems should also be automated to ensure that the most recent best model is available to the users and applications (4).

3. **Model monitoring**: Continuous and automated monitoring of the performance of these deployed models that are in production enables early detection of issues such as model/data drift and performance degradation (6).

4. **Model rollback**: Automatically rolling back to a previous version of a model if issues with performance is detected is an important feature for businesses (4).

2.2 A. Apache Airflow

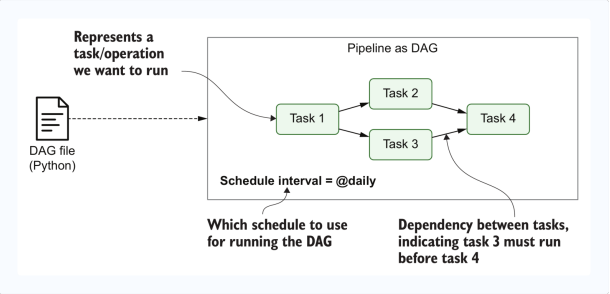

Figure 1: Diagram of a normal Airflow DAG pipeline. Image source: (7)

Apache Airflow is an open-source platform that offers a robust, scalable, and flexible solution for managing complex workflows and data pipelines (8). It is used a lot in the data engineering and data science communities for its extensive ability to orchestrate tasks, schedule jobs, and monitor their execution. The key functionalities of Apache Airflow include the following (8):

2.2.1 Workflow Management

Airflow provides a Python-based domain-specific language (DSL) to define and design workflows in the form of Directed Acyclic Graphs (DAGs). Each DAG represents a sequence of tasks with dependencies, where each such task is an instance of an operator (8).

2.2.2 Task Scheduling

Airflow also comes with a built-in scheduler that can trigger tasks in the DAG based on factors such as their dependencies and the specified schedule interval. The scheduler supports various scheduling options, such as cron expressions, fixed intervals, or manual triggers (8). We can see how the DAGs and scheduler functions in a typical Airflow pipeline in figure 1.

2.2.3 Monitoring

The web server of Airflow offers a user-friendly interface for monitoring the execution of tasks and DAGs. This interface allows the user to see many details such as the DAG's structure, task dependencies, and execution history, along with features to manage tasks, such as triggering, pausing, or marking them as successful.

The main components of Apache Airflow include (8):

- Directed Acyclic Graph (DAG): A DAG is a collection of tasks with specific relationships and dependencies that must be executed in a particular order.
- Operator: Operators are the building blocks of Airflow workflows. They define individual tasks and their execution logic.
- Task and TaskInstance: A task represents a single node in the DAG and is defined by an instance of an operator. A TaskInstance is a specific run of a task, with a unique execution date and state.
- Executor: Executors are responsible for running tasks in parallel, depending on the available resources and the specified concurrency limits.

2.3 GitLab CI/CD

GitLab CI/CD is a powerful feature that is included in the GitLab platform. As is expected of a CI/CD tool, it helps automate the process of building, testing, and deploying software applications and allows developers to seamlessly integrate code changes, detect errors, and deploy applications to production environments. GitLab CI/CD relies on pipelines that consist of multiple stages, each containing one or more jobs, to execute some predefined tasks.

Now we will describe some key components of GitLab CI/CD. These are the following (9):

1. **.gitlab-ci.yml**: This refers to the YAML-formatted file which is the core of GitLab CI/CD. This is what defines the pipeline structure, stages, and jobs to be executed on specific events, such as pushing to a repository or creating a merge request. So this file should be placed in the root directory of a project.
2. **Runners**: GitLab Runners are processes that execute CI/CD jobs on the specified environments. They can be installed on various operating systems including Linux, macOS, and Windows, and they can be appropriately configured to be executed on environments of Docker, Kubernetes, or other executors.
3. **Stages**: Stages define the order in which jobs are executed. GitLab provides three default stages: `build`, `test`, and `deploy`. But the user can customize these stages or create new ones entirely based on their specific needs. Each stage is executed in parallel, however jobs within a stage are executed in sequence.
4. **Jobs**: Jobs are individual tasks that are defined within a stage. They consist of some script of a sequence of commands that is to be executed by the runner, and could include additional configuration options, such as dependencies, artifacts, or environmental variables.
5. **Artifacts**: Artifacts are files/directories created by a job. They can be passed between jobs or stages and are used to store build outputs, test results, or deployment artifacts. They can also be configured to expire after a certain period to save memory.
6. **Environments**: As mentioned, GitLab CI/CD supports many environments and these can be defined in the `.gitlab-ci.yml` file, where each job can be assigned to a specific environment.
7. **Deployment**: GitLab CI/CD has a built-in support for deploying applications to various platforms, such as AWS, Google Cloud, or Kubernetes. This can be integrated with popular deployment tools like Helm, Capistrano, or Terraform to simplify the process of deploying and managing applications.

3 AutoML Pipeline with Airflow and Gitlab CI/CD

3.1 Workflow

To create a fully automated CI/CD ML pipeline, GitLab CI/CD and Apache Airflow can be combined to make use of the benefits of both tools (10; 11).

3.1.1 CI/CD Pipeline Configuration

The `.gitlab-ci.yml` file which is the core of Gitlab CI/CD is created and configured to define the stages and jobs for building, testing, and deploying ML models (10). This can include stages like linting, unit testing, integration testing, and deployment. For each stage, define jobs with relevant tasks and specify the required environment and dependencies to execute these tasks.

3.1.2 Airflow DAG Definition

After an Apache Airflow instance is setup in the chosen environment, a new Airflow DAG can be created for the ML pipeline. This DAG includes the tasks for data extraction, preprocessing, feature engineering, model training, evaluation, and deployment which can be defined using custom operators or built-in Airflow operators. The built-in scheduler can also be used specified with the `schedule_interval` parameter to trigger tasks based on some interval or through manual triggers which is independent of the CI/CD process (10).

3.1.3 Integrating GitLab CI/CD and Airflow

To integrate Airflow with Gitlab CI/CD, add a new job for each Airflow DAG in the `.gitlab-ci.yml` file. Also set up a webhook or API integration between Gitlab and Airflow to automate the pipeline execution after the deployment stage based on event-driven triggering mechanisms such as code pushes or merge requests. This implements CI to your pipeline whereas the CD is implicitly implemented by triggering the Airflow DAG which includes the deployment tasks. Finally, setup GitLab Runners that can run the CI/CD jobs (10).

3.1.4 Monitoring and Logging

Airflow's web server provides a user-friendly interface for monitoring the execution of tasks and DAGs, while GitLab CI/CD provides detailed logging and reporting on the build, test, and deployment stages.

3.2 Reflection

Although this autoML pipeline automates the whole ML process and fulfills most CI/CD requirements, it does not fulfill model rollback. While GitLab can be used to store and version the code associated with ML models, it is not the best tool for storing actual model files as they commonly are very large. A solution to this is to configure Gitlab so it stores the model files in external storage services such as Amazon S3, Google Cloud Storage etc. Another approach could be to use a dedicated model registry tool such as MLflow Model Registry (12) or Data Versino Control (DVS) (13) to manage model versions as these are specifically designed for ML projects.

3.3 Other State of the art solutions

There are various tools and platforms available for managing and automating machine learning (ML) pipelines. The choice between GitLab CI/CD and other CI/CD tools like Jenkins and Github Actions is mainly driven by factors like ease of integration with ML pipeline tools, level of customization required, and overall compatibility with the project infrastructure and requirements.

Airflow with GitLab CI/CD can perform well in various scenarios, especially when there is a need for custom, complex workflows that require integration with various data processing tools and services. The flexibility and extensibility of Airflow make it a suitable choice for managing sophisticated ML pipelines. GitLab CI/CD provides a seamless experience for automating the development and deployment process.

However, since Airflow is not specifically designed for ML pipelines, it could lack some specialized features available in platforms like Kubeflow which offers native support for deploying ML workflows on Kubernetes (14) and MLflow as aforementioned.

4 Conclusion

In conclusion, the concept of an AutoML CI/CD Pipeline with Apache Airflow and GitLab presents a promising approach to streamlining the development, testing, and deployment of ML models. By integrating these two powerful technologies, and including a method for storing model files, organizations can improve collaboration, efficiency, and the overall quality of their ML pipelines. But depending on the requirements of the projects, other platforms and tools may be more attractive.

References

[1] A. de Brébisson, S. Lapuschkin, A. Binder, K.-R. Müller, and W. Samek, "Automated data validation for machine learning," *arXiv preprint arXiv:2101.11943*, 2021.

[2] Y. Zhao, S. Huang, X. Feng, Z. Lin, R. Wang, J. Zhang, and J. Liu, "Autofeature: Automated feature engineering via tree-based pipeline optimization," *arXiv preprint arXiv:2012.04803*, 2020.

[3] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, pp. 2503–2511, 2015.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[6] V. Curcin, T. Woodcock, A. J. Poots, A. Majeed, and D. Bell, "Challenges and opportunities in data science for healthcare: a systematic review of the literature," *BMJ Open*, vol. 9, no. 8, p. e029635, 2019.

[7] B. P. Harenslak, J. R. de Ruiter, and F. van der Laan, *Data Pipelines with Apache Airflow*. Manning Publications, 2020.

[8] A. Airflow. Apache airflow documentation. https://airflow.apache.org/docs/. Accessed: 2023-05-03.

[9] GitLab Inc., "Gitlab documentation," https://docs.gitlab.com/, 2023, [Online; accessed May 3, 2023].

[10] P. DiLorenzo, F. Iorio, and R. Montella, "On the integration of ai pipelines with continuous integration and continuous deployment (ci/cd) in the cloud," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 100–106.

[11] I. C. Trelea, V. Aksakalli, A. Oztekin, and A. Yucel, "A machine learning and optimization based framework for predictive maintenance in large scale manufacturing," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1075–1088, 2020.

[12] MLflow. Mlflow documentation. https://mlflow.org/docs/latest/index.html. Accessed: 2023-05-06.

[13] D. Dmitriev, P. Korostelev, and A. Kibardin, "Data version control: Git for data models," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, 2018.

[14] Kubeflow, "Kubeflow," https://kubeflow.org/, 2021, accessed: 2023-05-05.