Automation and Orchestration in ML and non ML processes

Airflow & Kubeflow

Technological Infrastructure for Data Science

Università degli Studi di Milano Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione

Guerini Paolo, Rocchi Niccolò, Scatassi Marco

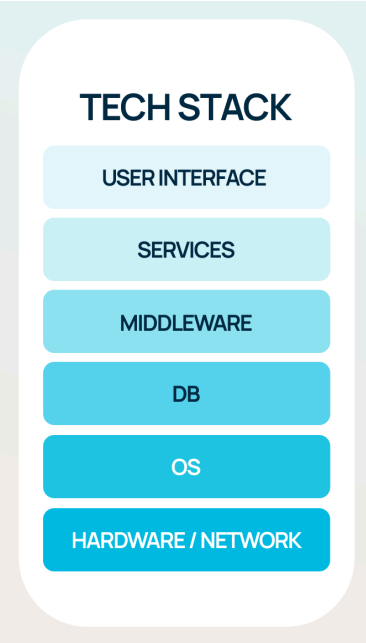
Agenda

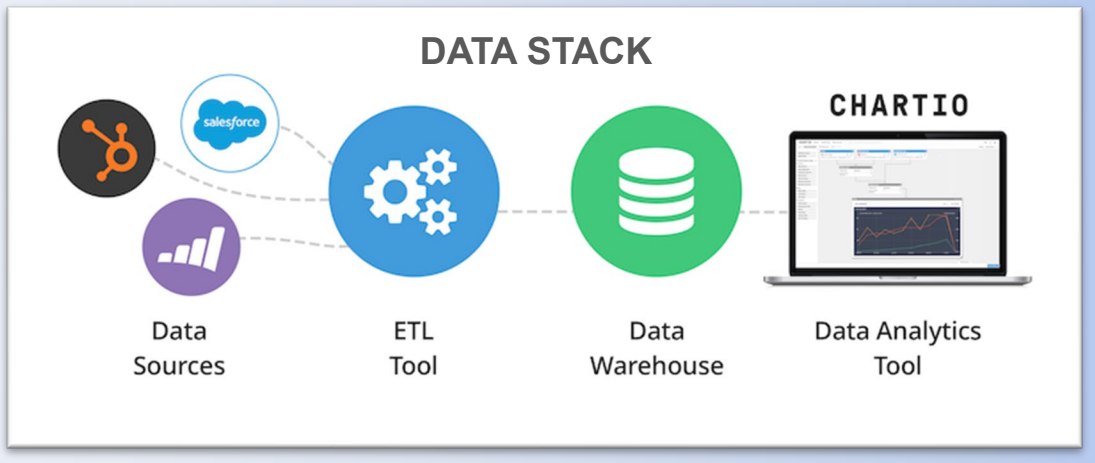
- Automation and orchestration
- Kubeflow
- Airflow

Before the Big Data Era

- Model-centric decision making for companies
- ETL/ELT just aim to serve data to storage, not to data apps and analytics
- Limited business value of data

From "inedible data" to "edible data" through Classical Data Stack (ETL and EDW)





The Big Data Era (~2000)

- Loads of new **multiple sources, types, velocity**
- Data as leading actor because of IoT
- Birth of **data-driven** strategies
- The more the data to dig into, the more governance, sync **schedules and processing problems**.
- Just adding new tools to the mix just increases complexity
- Birth of **value-chain**: how do we **extract business value?**

Classical Data Stack: **heavy**, time-consuming, difficult to maintain and manage

A need for a change

Biggest pain points for organizations:

- **Wasting time and resources** on manual coding
  - Lack of automation, thousands of servers to manage, manually fix bugs
- **Variety of data** churned out daily; **data siloed** and scattered
  - Heterogeneity of data formats and fragmented data sources
- Catching problems **after their impact**
  - We'd like to take automated data-driven decisions
- **Migrating** to the cloud and **interconnecting** the ecosystem
  - Too strict computation-storage relation. Lack of flexibility and agility

Today: **80%** of work in **ingesting and preparing data**.

And more than **87%** of businesses still have **low BI** and analytics maturity

The Modern Data Stack

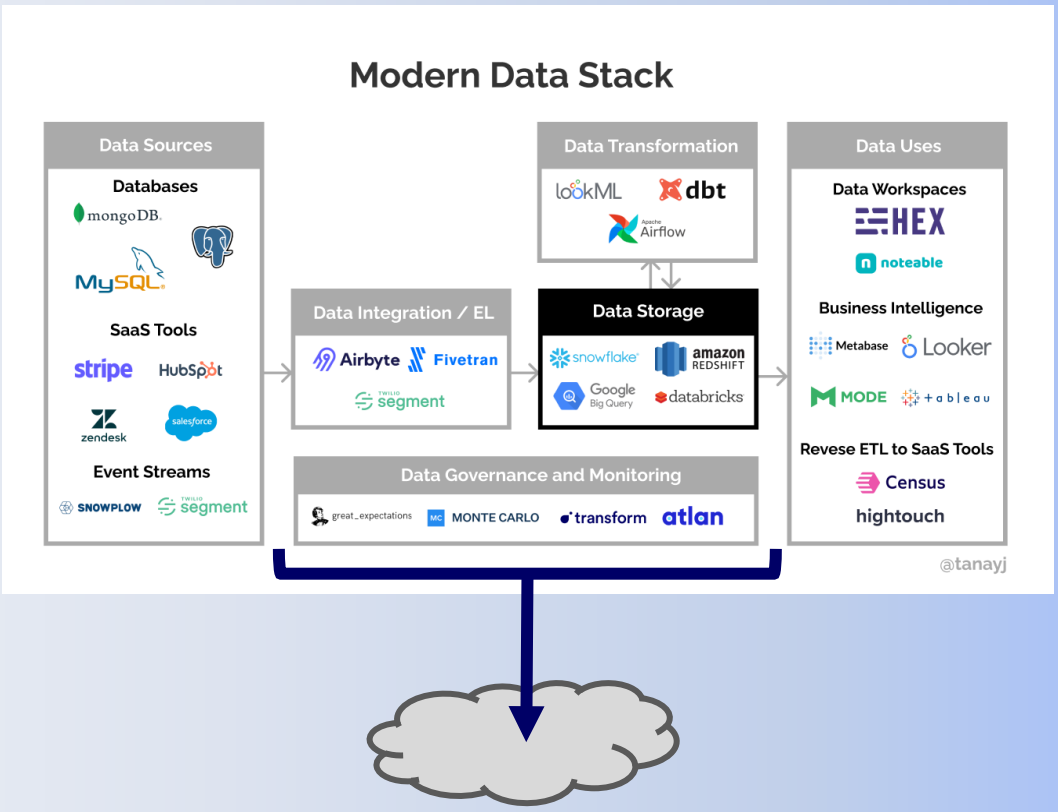
Raise of modern data stack concept:

- **Modularity** (standalone but integrated entities)
- **Cloud**-based solution
- Cost and time: **Agile**
- **Easy** to use and **re-use**

Requirements: **automation and orchestration**

Result: **more valuable data** as output for analytics.

Batch, streaming or a mix of both!



Automation or Orchestration?

Automation: automating a single process or a small number of related tasks

- One task per time
- Rule-based decision system
- No human involvement
- Single flow process
- No monitoring system

Output: end-to-end automated task

Orchestration: managing multiple automated tasks to create a dynamic workflow

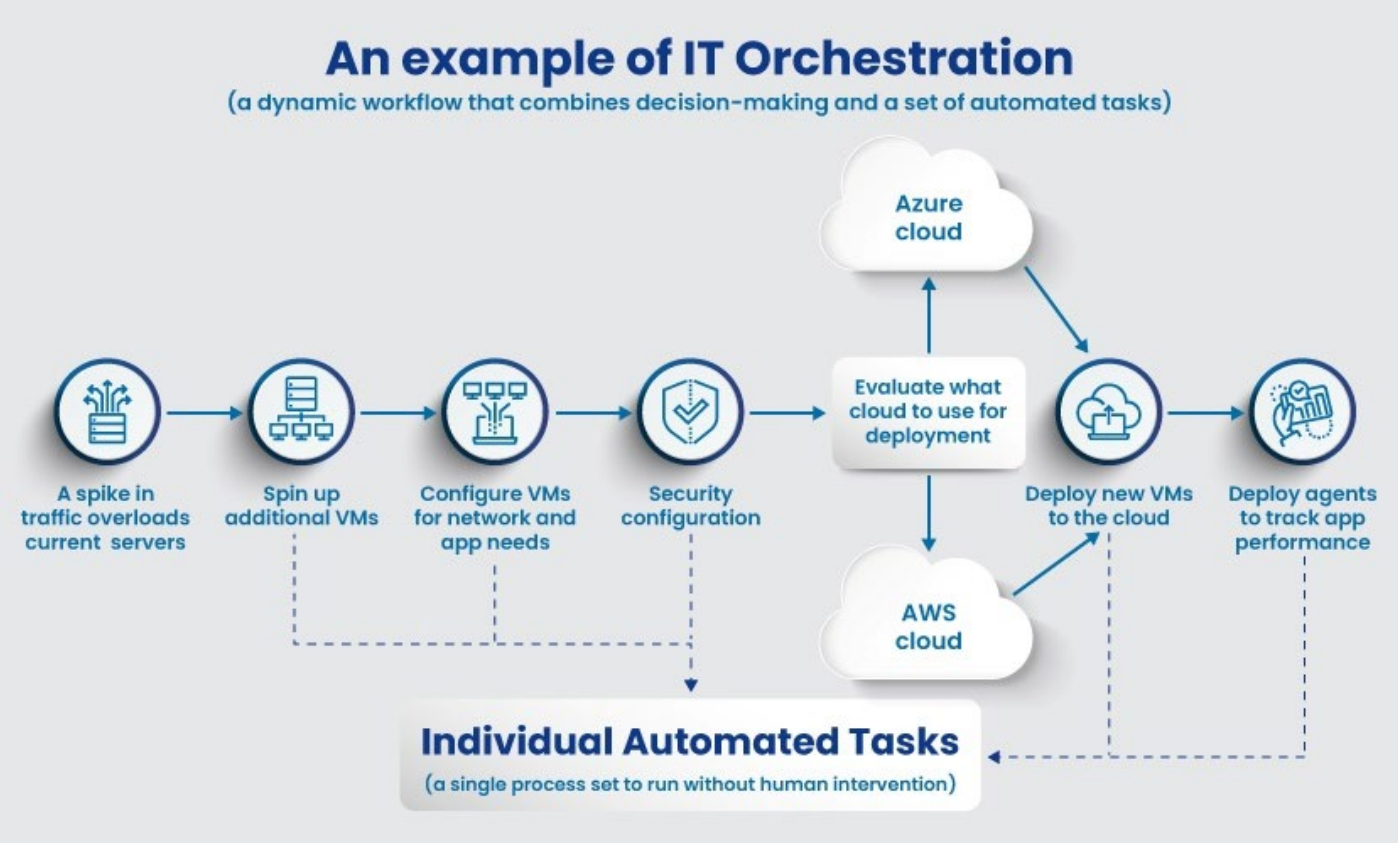
- More tasks at same time
- Decisions based on actions and outputs
- Human involvement
- More branches flow
- Monitoring and adapt to changes

Output: end-to-end coordinated, managed and tracked workflow

Automation or Orchestration?

Automation: the "boxes"

Orchestration: the "arrows"

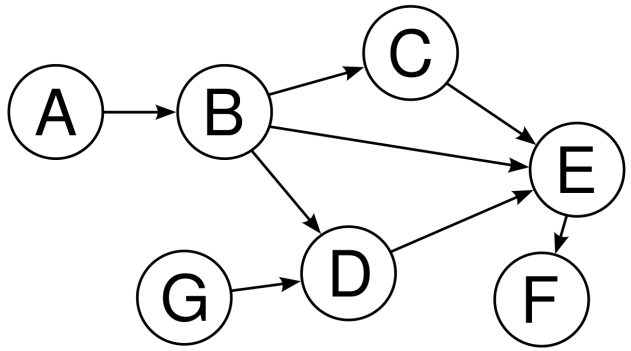


Some terminology

Actually, the term orchestration is broader:

- **Application** orchestration (SE)
- **Service** (or microservice) orchestration
- **Container** orchestration
- **Security** orchestration

Represented by a **DAG**



Our focus: **dataflow orchestration**. Namely the sub-workflow nodes that touches **data**. It can be:

- **Non-ML process:** from data sources to edible data for analysis
- **ML process:** from data sources to a deployed trained model

What does Orchestrator do?

The **delivery company** metaphor:

A complete delivery is a workflow running,
and each box is an automated task

Each box has different
contents and **shapes**

Fine-grained **visibility** of each package state, but with complete **privacy** of content

The company collects
delivery **history** through metrics

What does Orchestrator do?

You just "make your order by a click"

Scale: multiple deliveries at the same time, asynchronously

If a box was damaged, **retry** or **restart**.
You can also wrap them

You can **(re)schedule** the delivery

Different **vendors** for each box, different **sources** and delivery **addresses**.

Why should you care? Benefits

Orchestrator is the actual best solution for **Big Data Stacks**:

- **Automated** but potentially high-level **controlled** process
- Improve **data governance** and **visibilty**, infrastructure **management** and **fault tolerance**
- It **breaks data silos** from sources to storage improving **collection, preparation and unification**
- Acquire **real-time insights** and metrics

Why should you care? Benefits

Still...

- Leverage **fresher consumer data**, but ensuring privacy
- High quality data implies **high value** for company business, and **greater compliance**
- Allow to make **real-time valuable decisions** through delivery of already **activated data**
- A **DAG** is the most suited model to represent and train ML methods (Continous Retraining)

Orchestration and Data Science

Data Science → To create and automate blocks → Dataflow orchestration

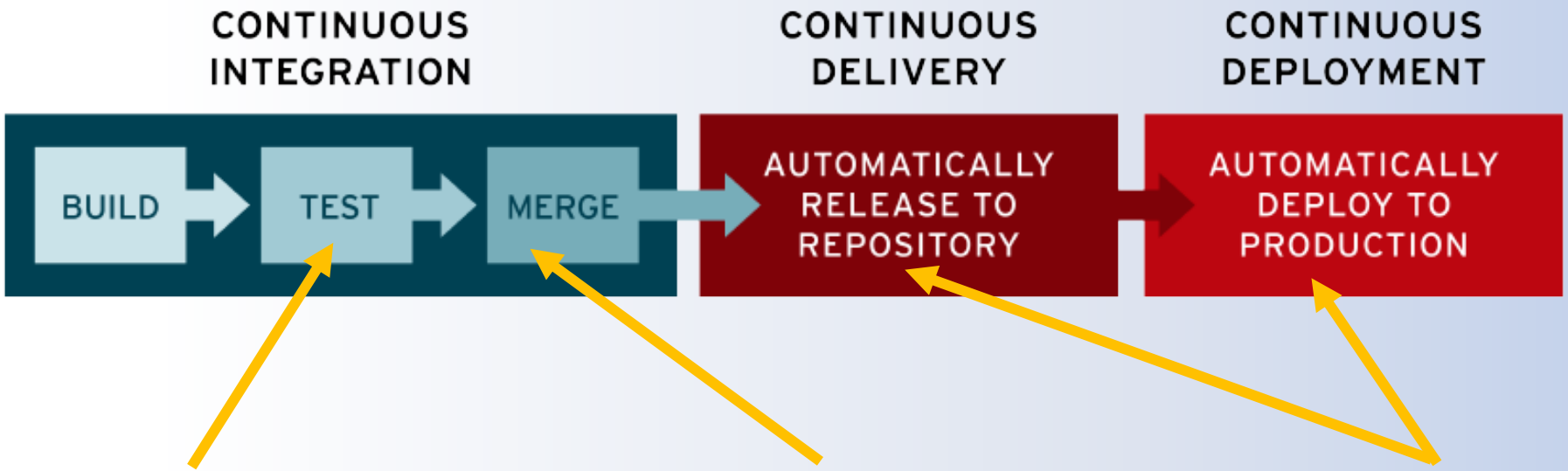
Loosely coupled but integrated **tasks**, automated by data architects, engeneers, analysts, scientists

Data Science ← As an important tool for analysis ← Dataflow orchestration

Easy tool for training ML models. Importance of having a "**stable**" data pipeline to work on

Role in DataOps & MLOps

The idea: we just want a solution to a **recurring problem**



What about **re-write tests** at every new integration?

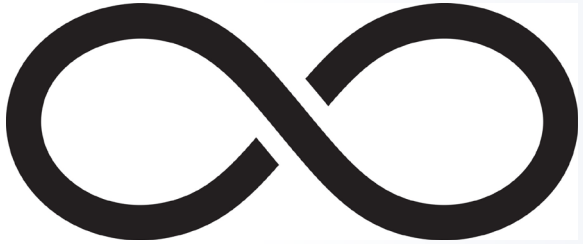
What about **manually merge** tasks each time?

What about if an **error** occurs here?

And what about **to schedule, scale, monitor**...?

Role in DataOps & MLOps

DataOps and MLOps work well as Python calls until dataset is static and small.



We further need to break it into sub-tasks and to use orchestration to:

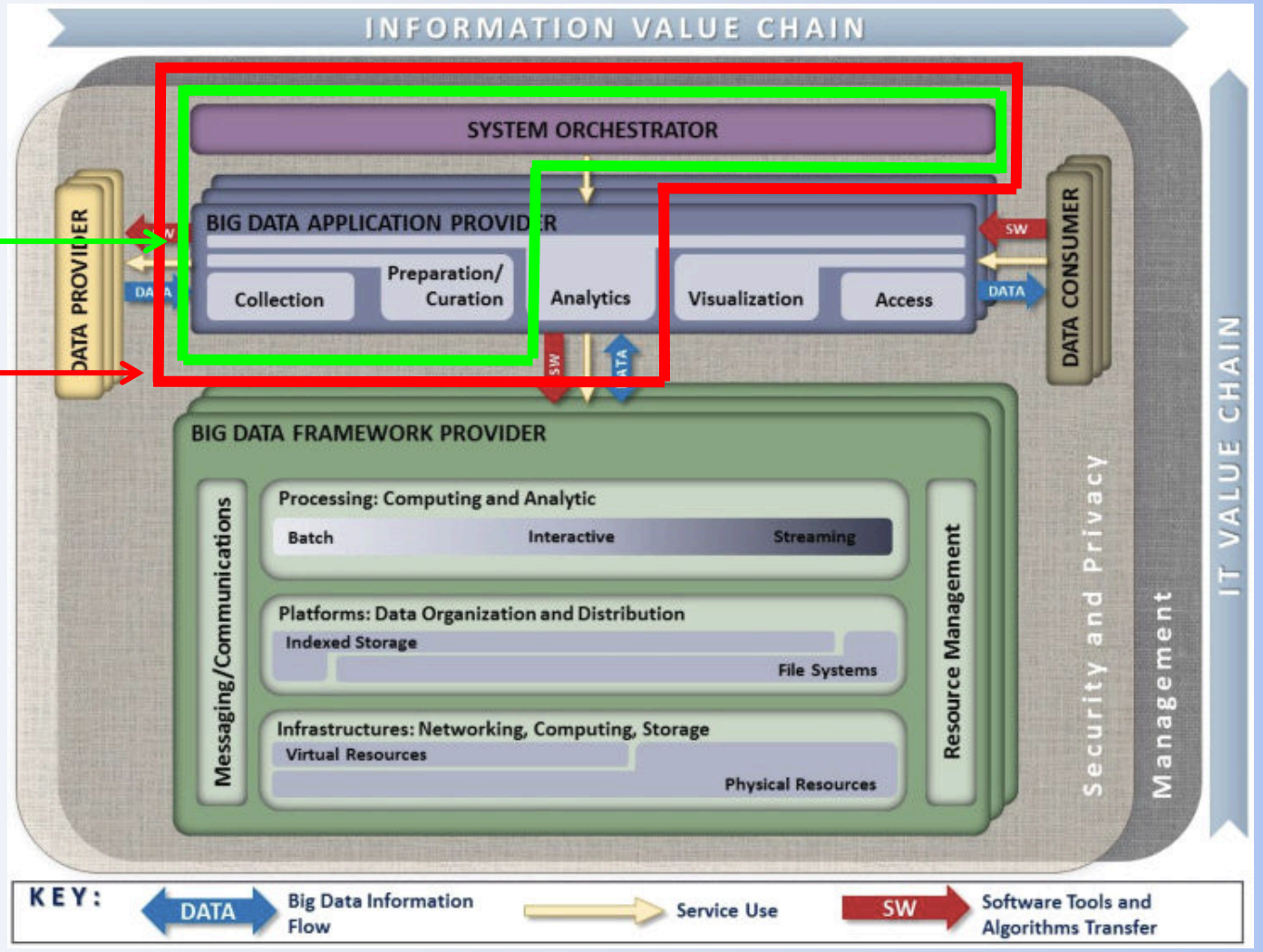
- Effectively and efficiently implement **CI/CD** or **CI/CD/CR**
- **Visually design** workflows: **transparency** to organization
- **Automatically** promote workflows between stages, in an **Agile** manner
- Enforce **feedbacks** with **observability**

Where are we?

According to the NIST reference:

Non-ML workflow

ML workflow



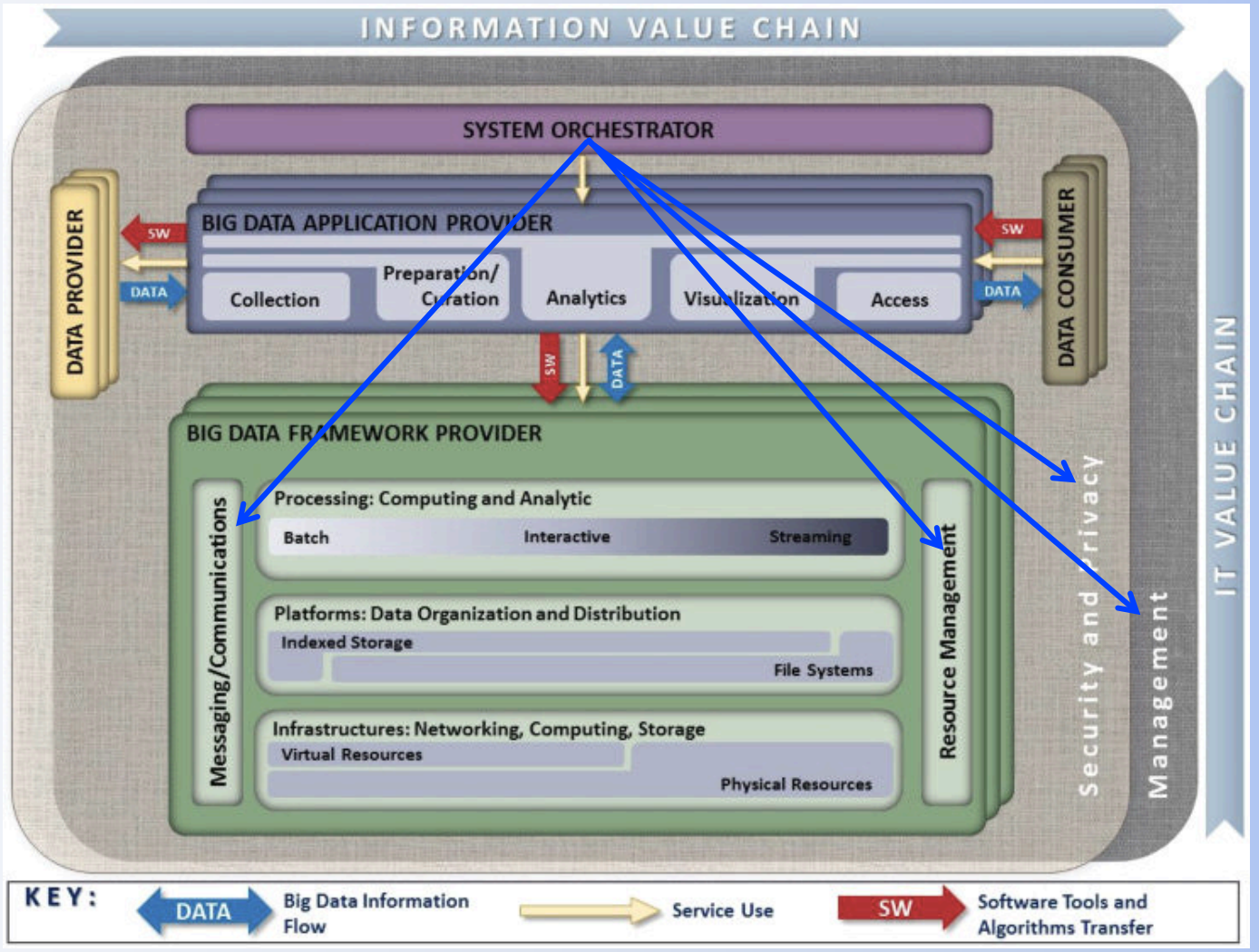
Where are we?

Orchestrator also deals with:

- Management and Security
- Resource Management and Messaging

It monitors if **requirements** are met and elastically **provides** phisycal or virtual resources

Cross-cutting roles and fabrics



Generations of tools

0. "Zero" Generation: Domain Specific Languages (DSL) for scheduling (like JSON or YAML)

1. First Generation: improved usability for data scientists through UIs and Python (e.g. Airflow, Luigi). They were task-driven, decoupling process (*what to do*) from its management (*how to do*)

2. Second generation: data-driven processes with improved awareness. They adapt to the data type by anticipating actions and performing ad-hoc tests (examples: Prefect, Flyte, Dagster)

Generations of tools

Two categories of **data-driven approaches**:

- **Active**: actively and intelligently split and transfer data through appropriate steps
- **Passive**: wait outside the DAG for an event to occur before triggering a task. Made possible by collecting tasks completion and data arrivals streams. It is suggested for complex workflows, e.g. ML-workflows with continous model-retraining (CR)

Let's now focus on...

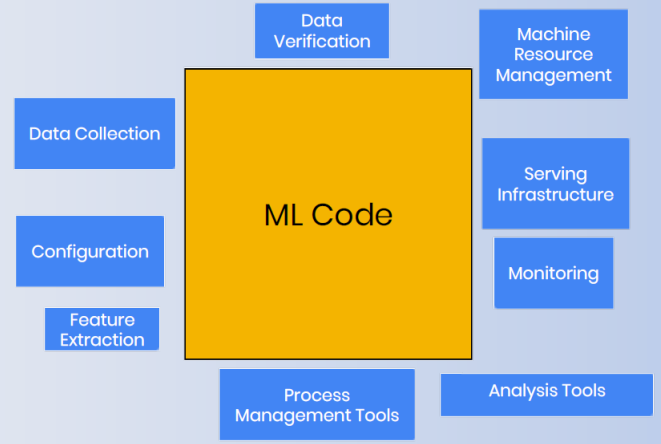
Agenda

- Automation and orchestration
- **Kubeflow**
- Airflow

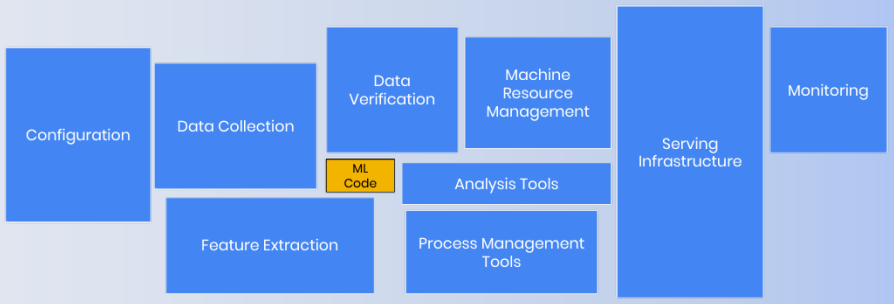
The hidden ML

- **Developing** ML systems is relatively **fast** and **cheap.**
- **Maintaining** them over time is **difficult** and **expensive**.

Perception

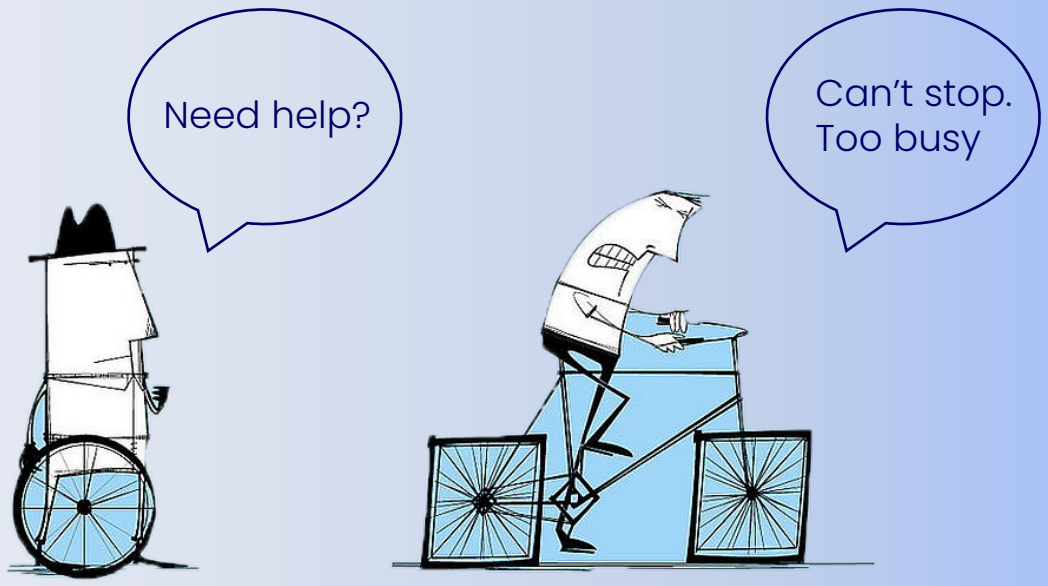


Reality



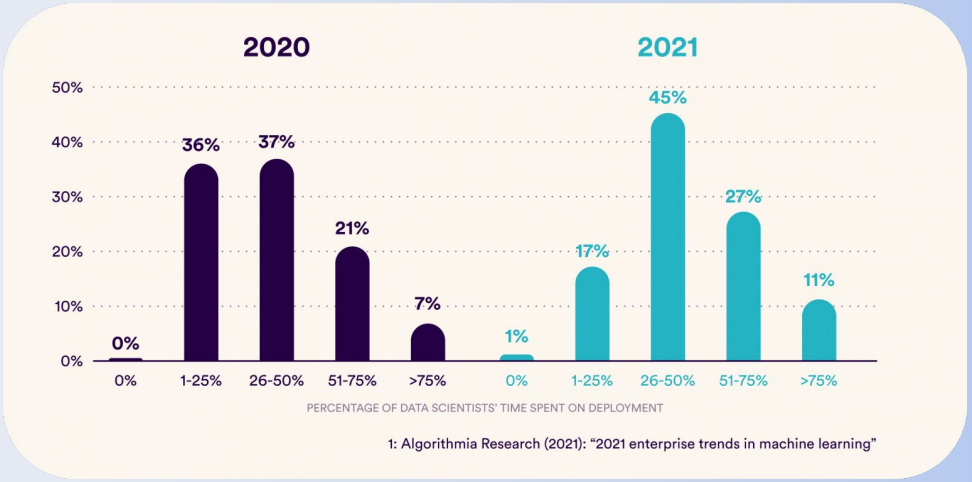
Technical debt in ML

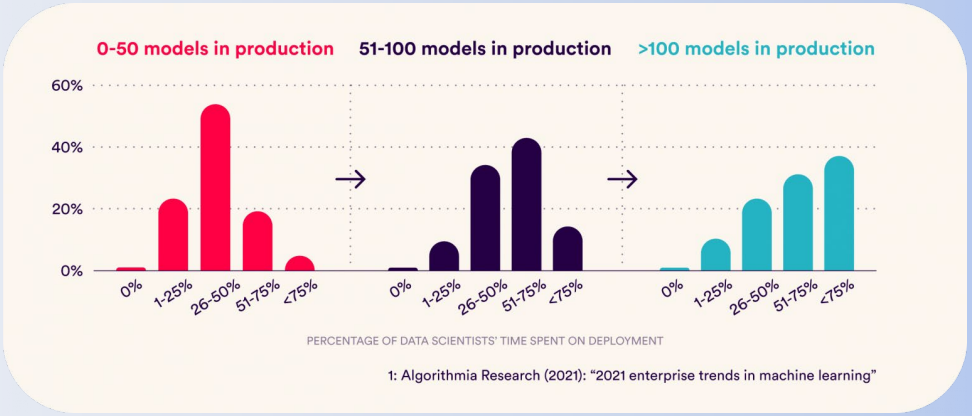
- **Traditional** issues:
  - **Tightly coupled** components
  - Lack of **software documentation**
  - Lack of **collaboration**, …
- **ML specific** issues:
  - Data **dependencies**
  - **Monitoring** debt
  - **Entanglement**, …



Model serving

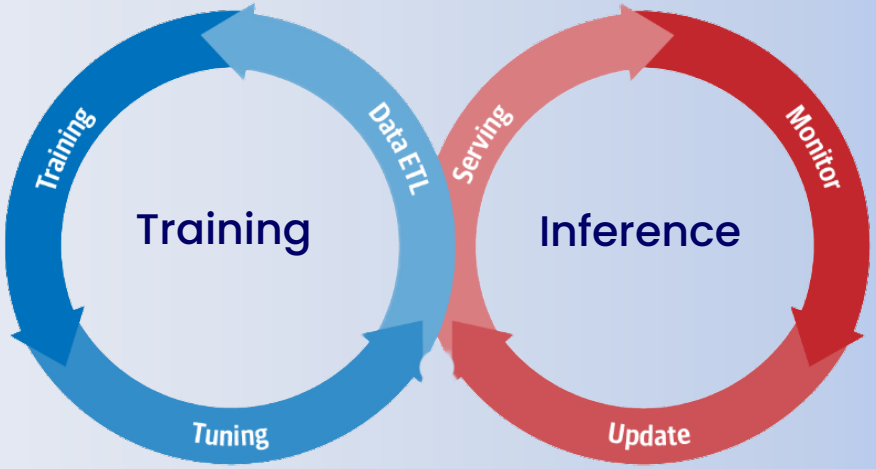
- Only models that are deployed to production provide **business value** to customers and users.
- Anywhere between 60%-90% of models **don't make it to production**, according to various analyses.
- The right mindset is to build an **automated pipeline**, where the problem is split into smaller components, where:
  - **data scientists** focus on the components
  - **software engineers** focus on the pipeline.





What is Kubeflow

- Kubeflow is an open-source**, Kubernetes-native** platform
  - for **developing**, **orchestrating**, **deploying**, and **running**
  - **scalable** and **portable** machine learning (ML) workloads.
- Main features
  - **Composability**
  - **Portability**
  - **Scalability**



Model development life cycle (MDCL)

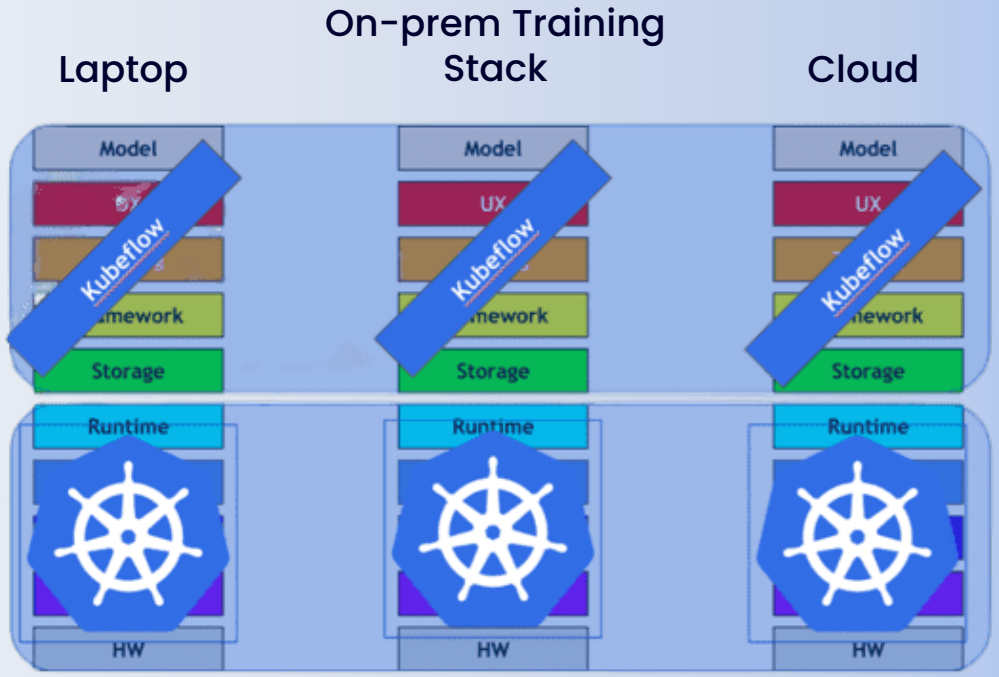
Composability

- Kubeflow is a collection of tools
  - Independent and **cloud native**
  - that can work seamlessly **together**
  - for all the stages of **Model development life cycle** (MDCL)
- allowing
  - **integrated end-to-end pipelines**
  - that connect all components of a **MDLC**.

Some components that run on Kubeflow

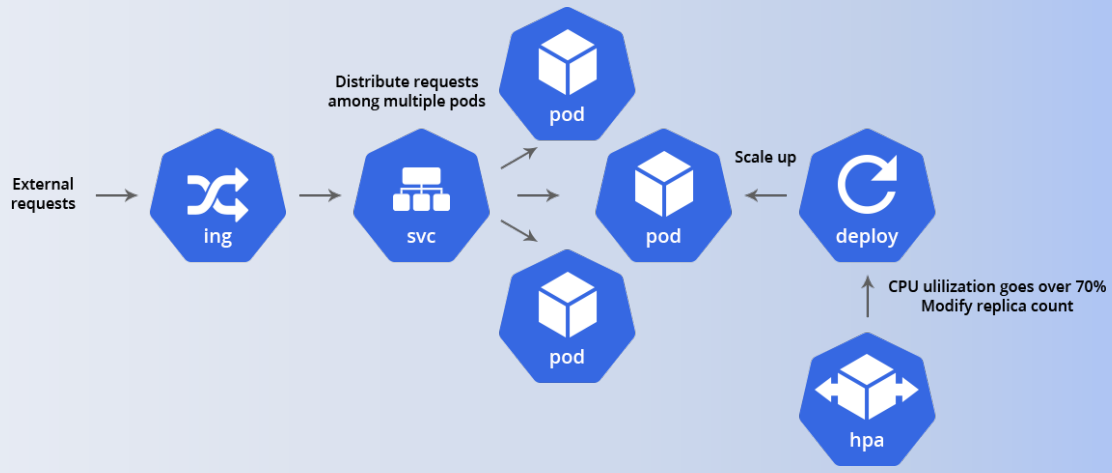
Portability

- Kubeflow generates an **abstraction layer** between your system and the ML project.
  - By means of a **container-based** design
  - Taking advantage of **Kubernetes**
- The ML project can be run **anywhere** you are using Kubeflow
  - You can experiment and prototype on your laptop and deploy to production effortlessly.



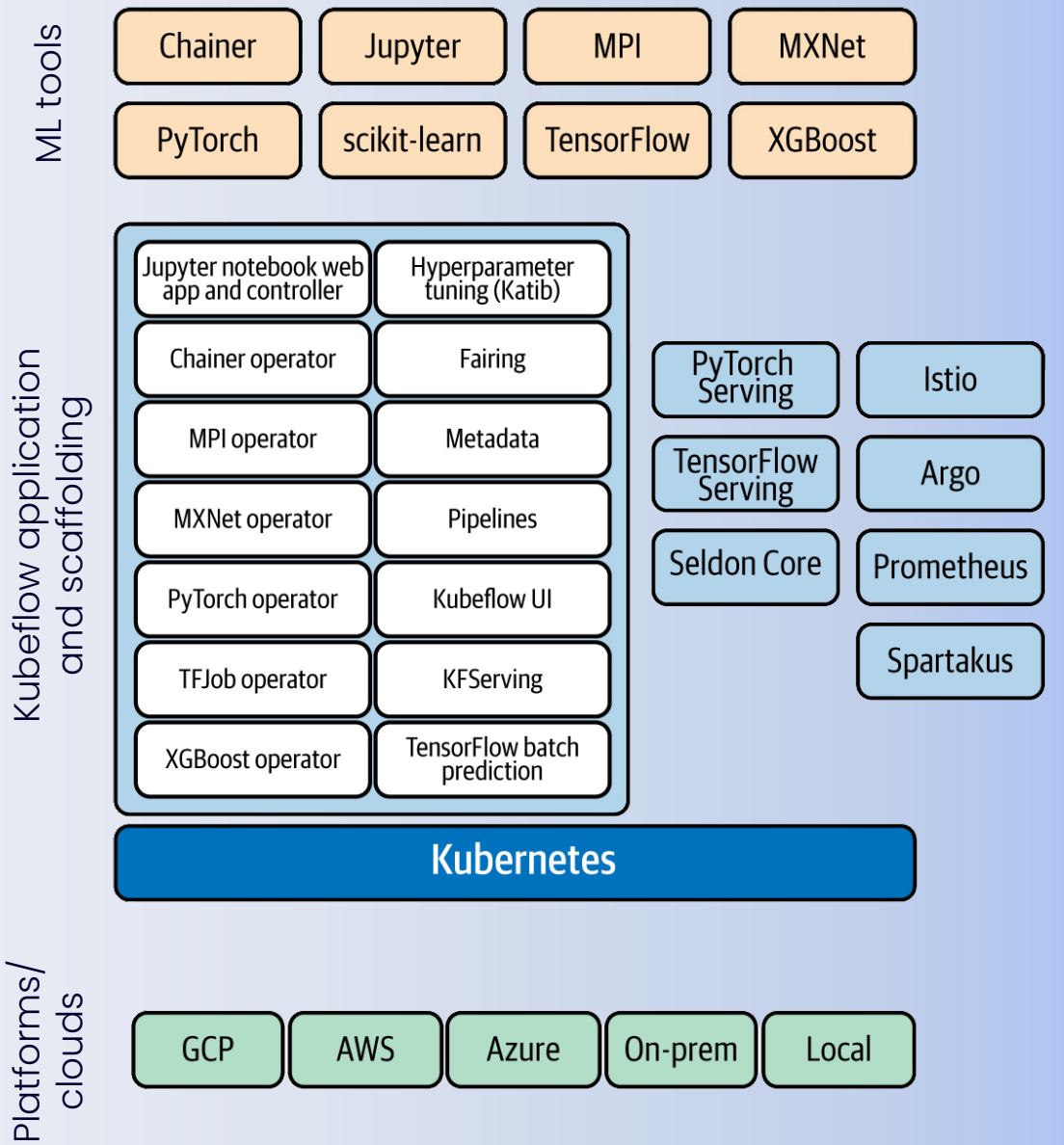
Scalability

- Kubeflow can **dynamically scale** according to the demand on your cluster
  - By using **Kubernetes**
  - Changing the number and size of the underlying containers and machines.



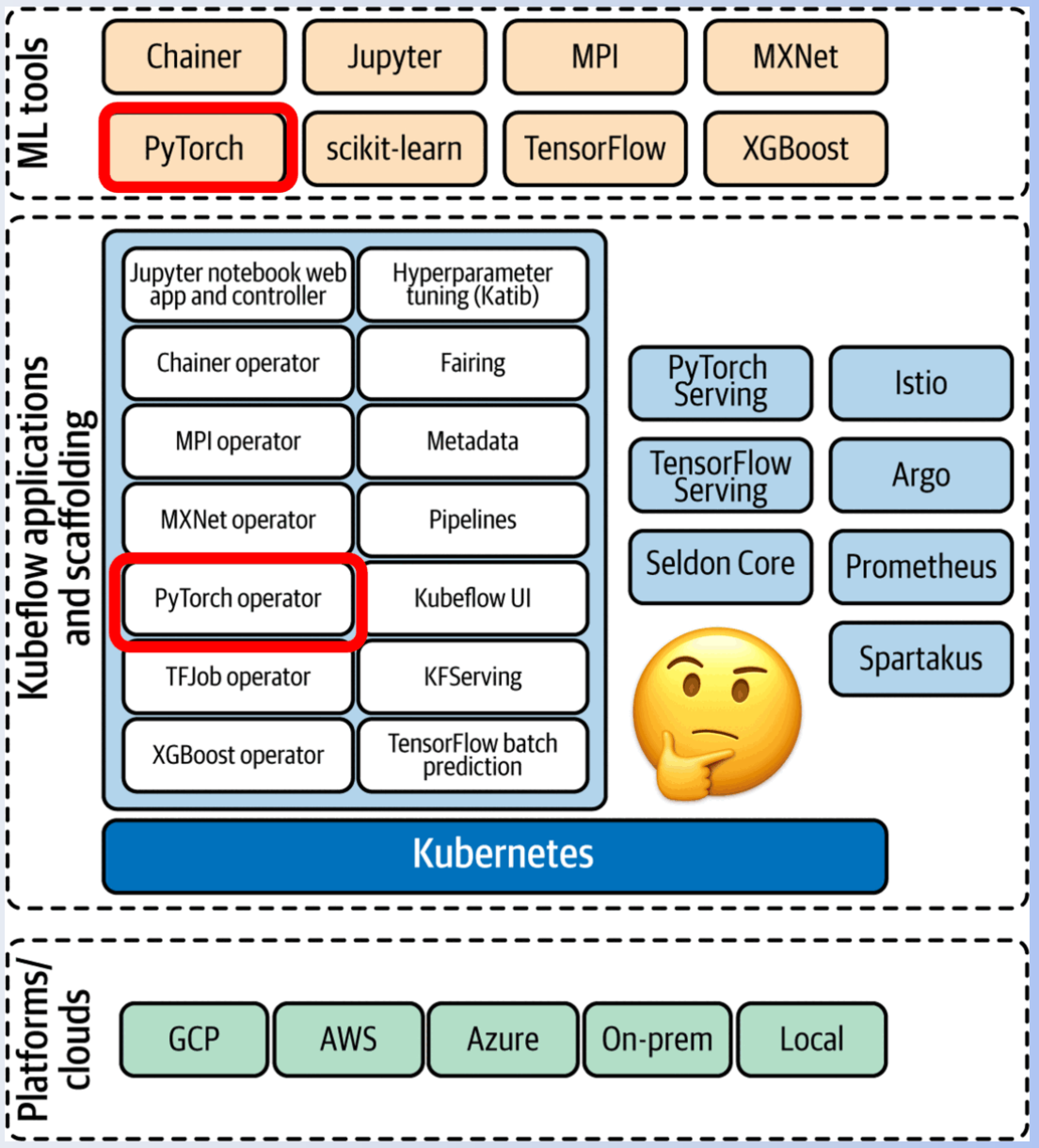
Architecture

- The Kubeflow architecture can be split in **three major blocks**
  - Machine Learning Tools
  - Applications and Scaffolding
  - Platforms and Clouds
- Application and services in ML tools and scaffolding are called **components**



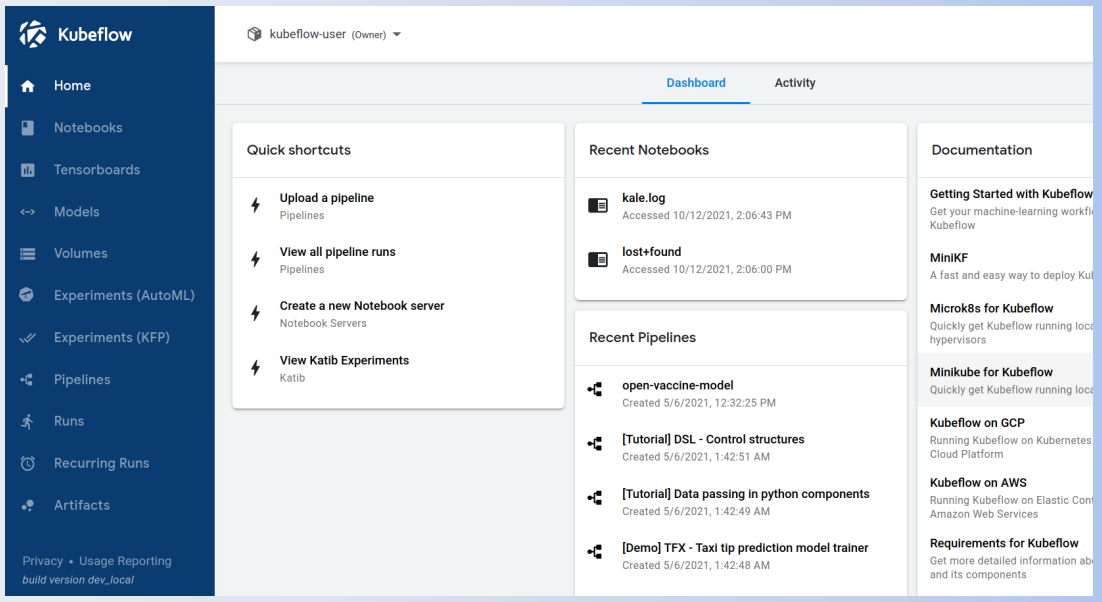
Training operator

- Which is the difference between an ML components and its **operator**?
  - The ML component simply represent a **framework** supported by Kubeflow;
  - Kubeflow **training operators** are
    - Kubernetes software extension
    - That make use of training Jobs custom resource
    - To run training task of ML models



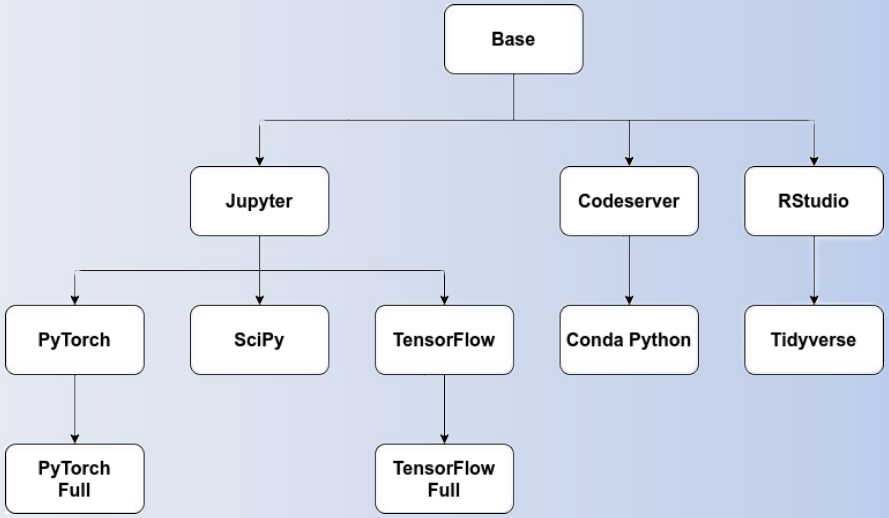
Central Dashboard

- The **central dashboard** provides quick access to the Kubeflow components deployed in your cluster.
- From the Kubeflow UI we can **visually performs** tasks such as
  - creating Pipelines
  - running hyperparameter optimization jobs
  - launching Jupyter Notebook servers
  - ...



Notebooks

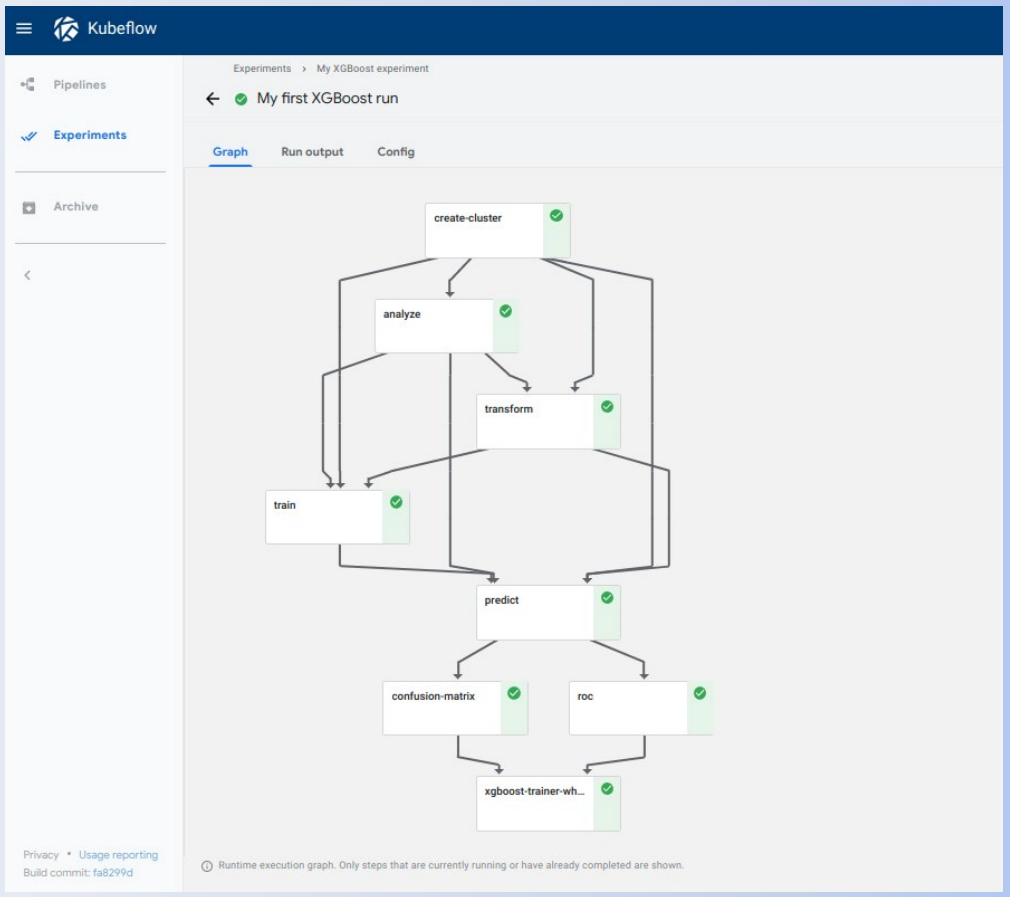
- **Kubeflow Notebooks**
  - provides a way to run **web-based development environments** inside your Kubernetes cluster by running them inside **Pods**.
- Some key features include:
  - Native support for **JupyterLab**, **RStudio**, and **Visual Studio Code** (code-server).
  - Admins can provide **standard notebook images** with required packages installed.
  - Enable **easier notebook sharing** across the organization.



Notebook image Dependency Chart

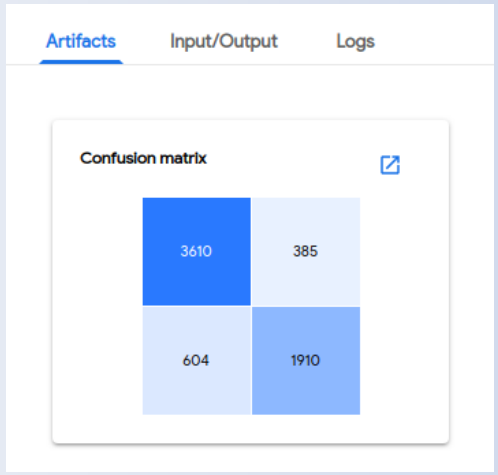
Pipelines

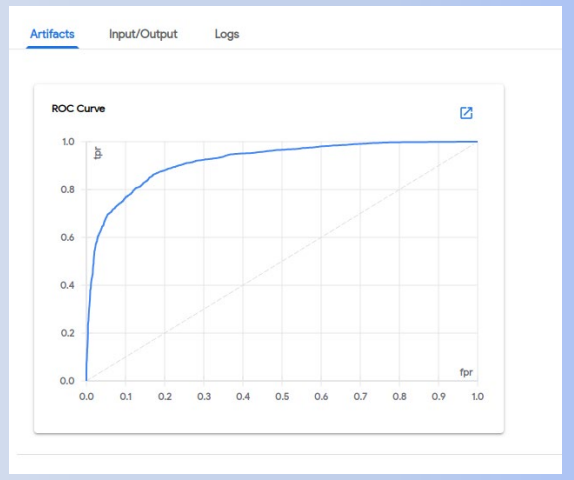
- Kubeflow Pipelines is a platform for building and deploying machine learning (ML) workflows.
- **Goals**:
  - End-to-end orchestration
  - Easy experimentation
  - Easy re-use

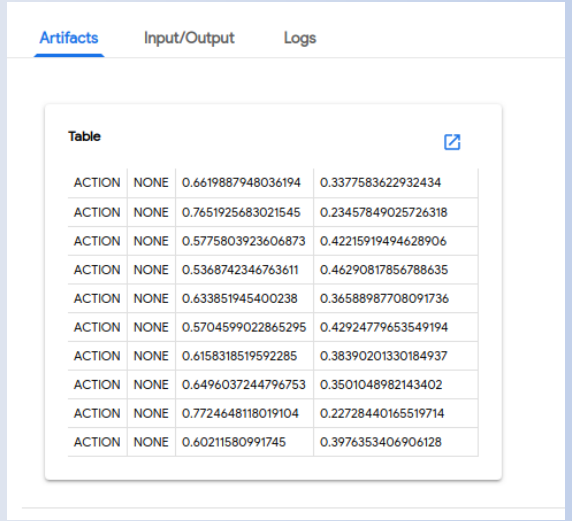


Pipelines

- A Kubeflow Pipeline is a **directed acyclic graph (DAG)** representing all the components in the workflow.
- A pipeline **component** is a **self-contained set of user code** packaged as a Docker image that performs one step in the pipeline. It can emit an **output artifact**.
- An **output artifact** is an output emitted by a pipeline component, which the Kubeflow Pipelines UI understands and can render as rich visualizations

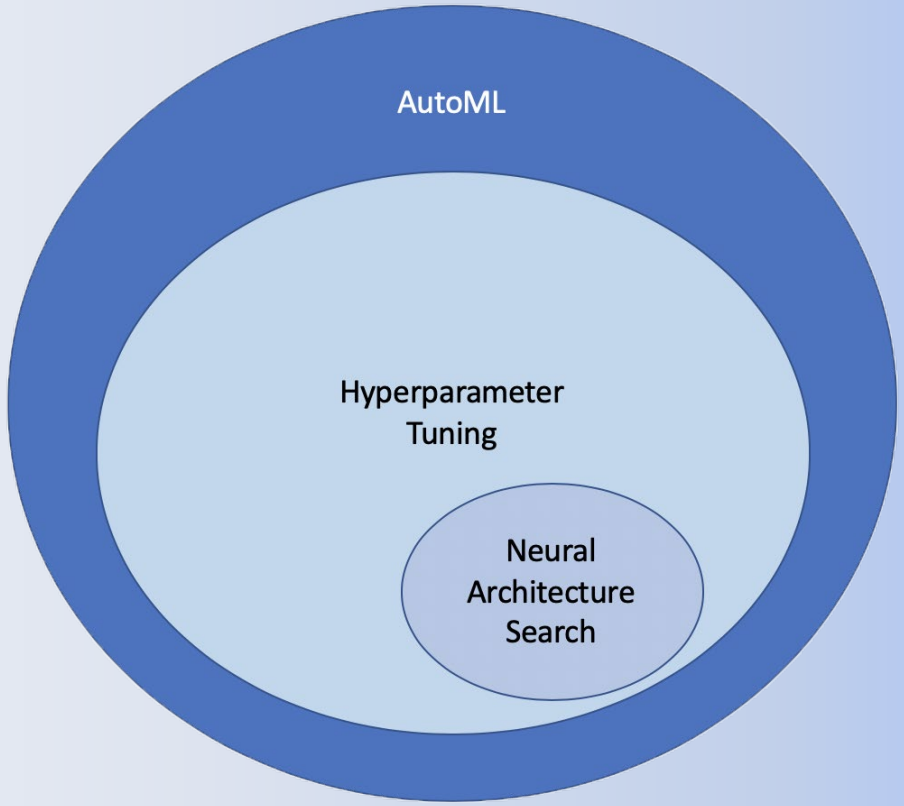






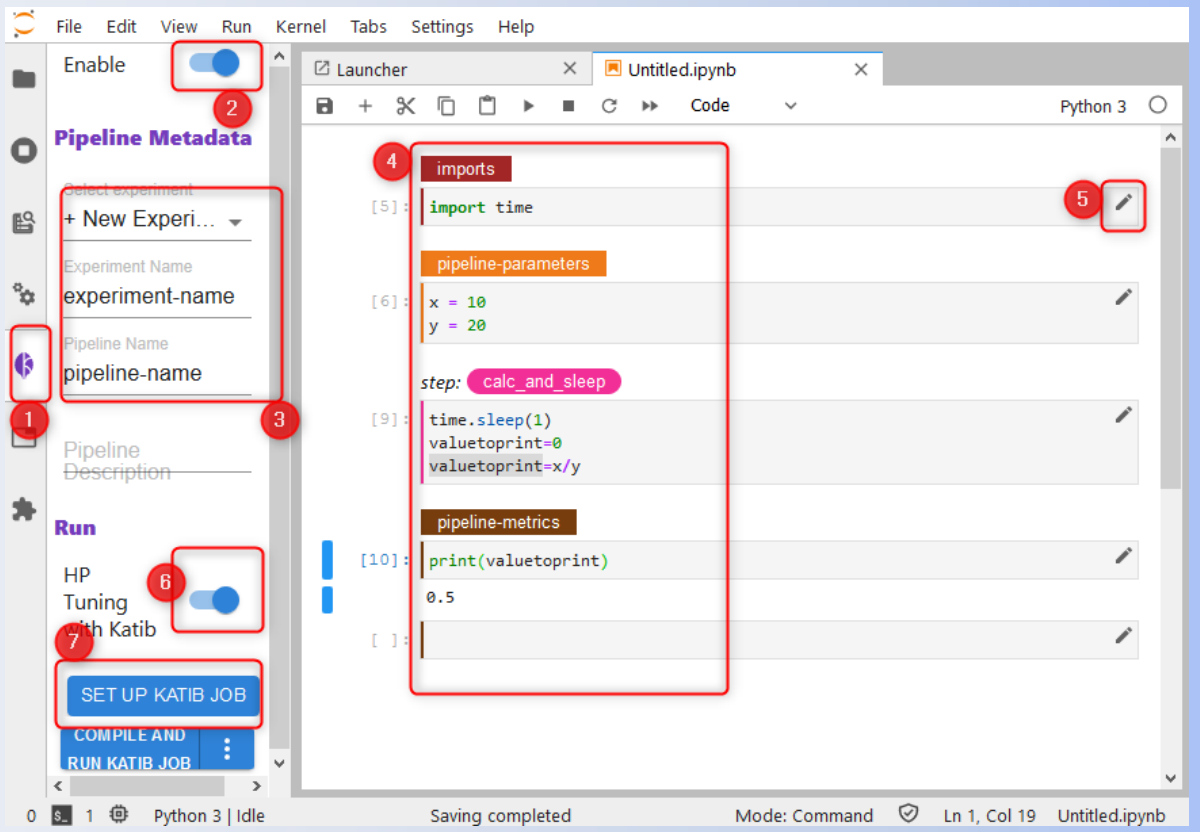
Katib

- Katib is a Kubernetes-native project for automated machine learning (**AutoML**).
- It supports
  - hyperparameter tuning,
  - early stopping and
  - neural architecture search (NAS)
- Katib natively supports many ML frameworks, such as
  - TensorFlow, MXNet, PyTorch, XGBoost, and others neural architecture search (NAS)



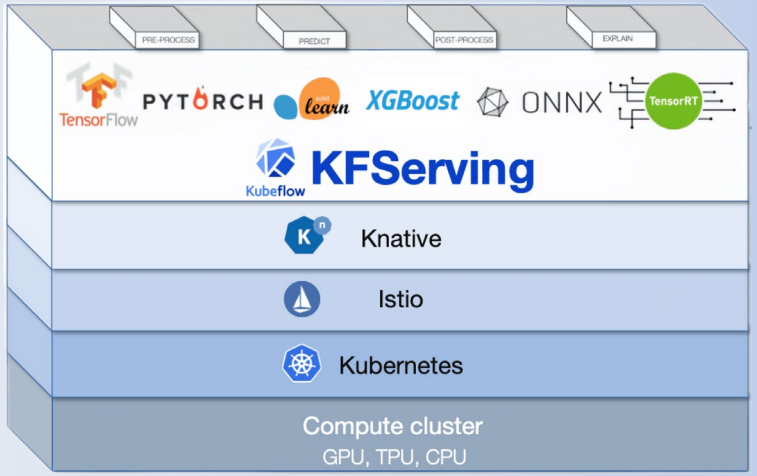
Kale

- Kale simplifies the use of Kubeflow, giving data scientists the tool they need to orchestrate end-to-end ML workflows.
- Kale provides a UI in the form of a JupyterLab extension. You can annotate cells in Jupyter Notebooks to define:
  - pipeline steps,
  - hyperparameter tuning,
  - GPU usage, and
  - metrics tracking.



Model serving

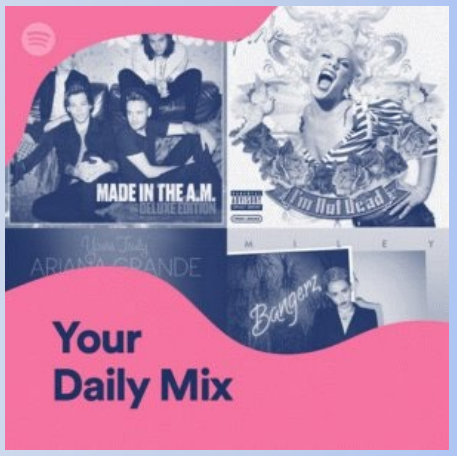
- Kubeflow supports **two model serving systems** that allow multi-framework model serving:
  - KFServing
  - Seldon Core.
- KFServing and Seldon Core are both open-source systems that allow **multi-framework model serving**
- Alternatively, you can use a **standalone** model serving system.



Spotify

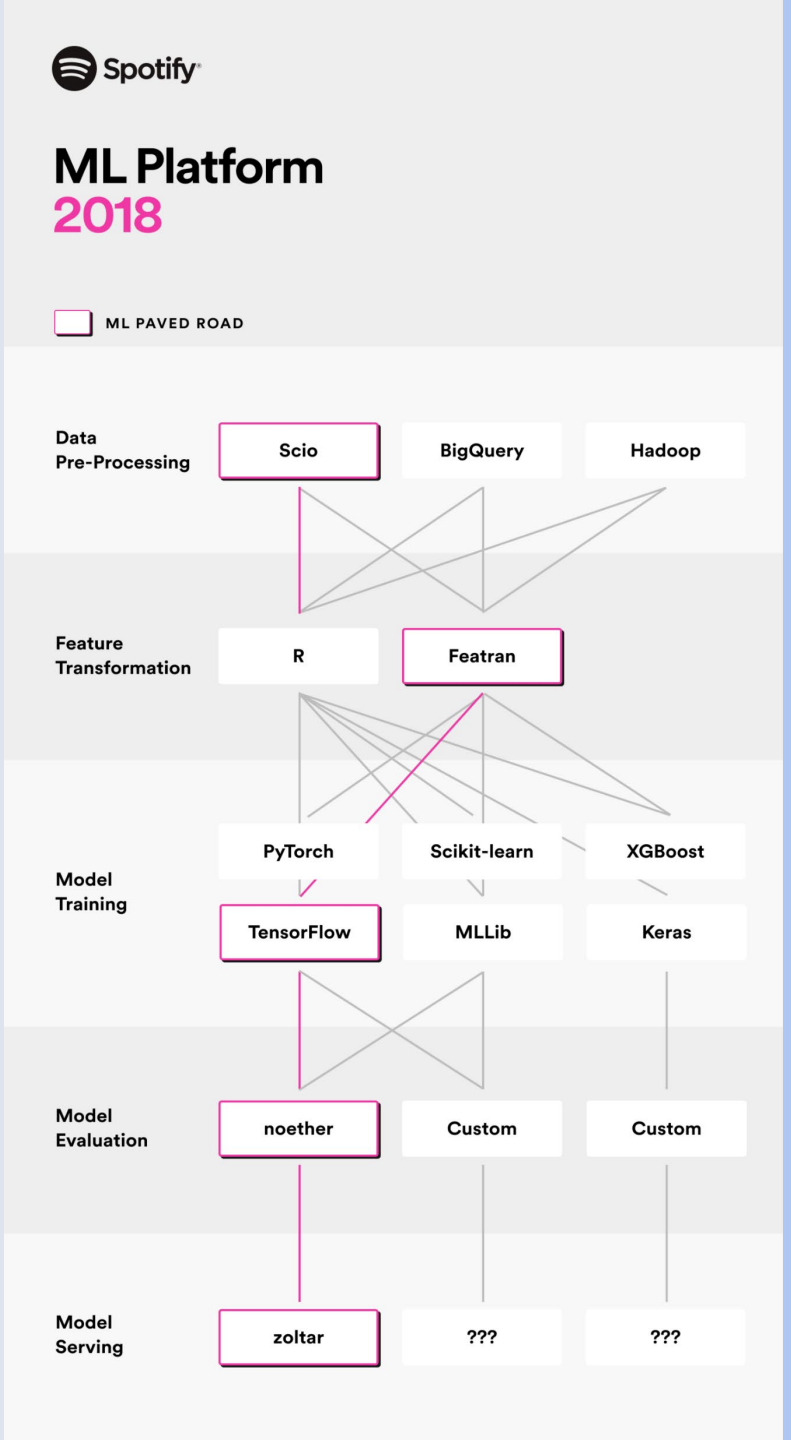
- Spotify is one of the largest **music streaming service providers** with over 489 million monthly active users
- It use ML, especially reinforcement learning, to personalize each Spotify user profile (➞)
- More users and more features led to more systems that relied on Machine Learning to scale inferences across a growing user base

- Necessity to **standardize best practices** and build tooling to bridge the gaps between
  - data, backend, and ML



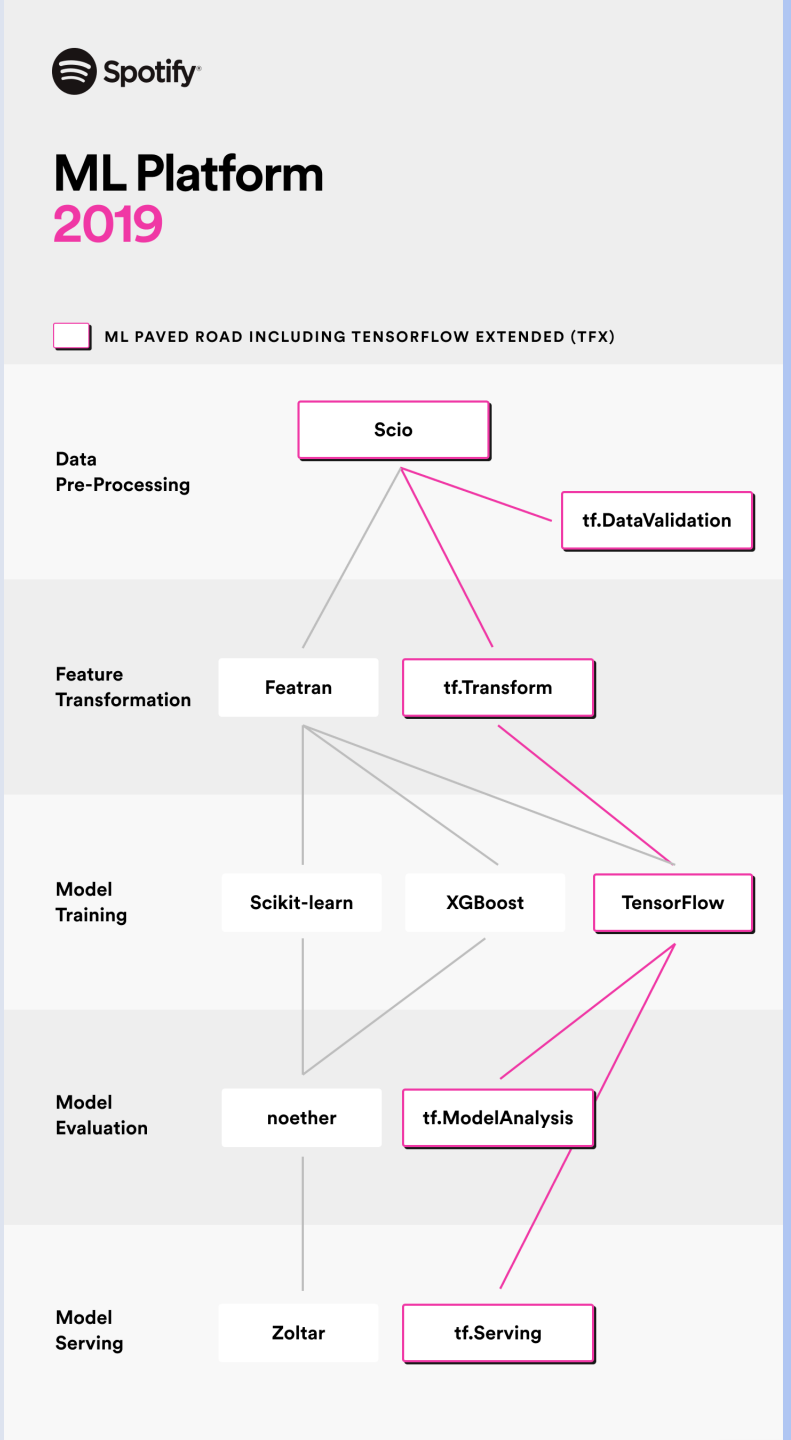
The First Iteration

- **Problem**
  - Spotify has had major ML systems in production long before it ever had a team dedicated to building tools for ML teams.
- **Solution**
  - meet their teams where they were
  - by building connectors amongst the most popular ML frameworks in active use
  - instead of dictating to teams which framework to use



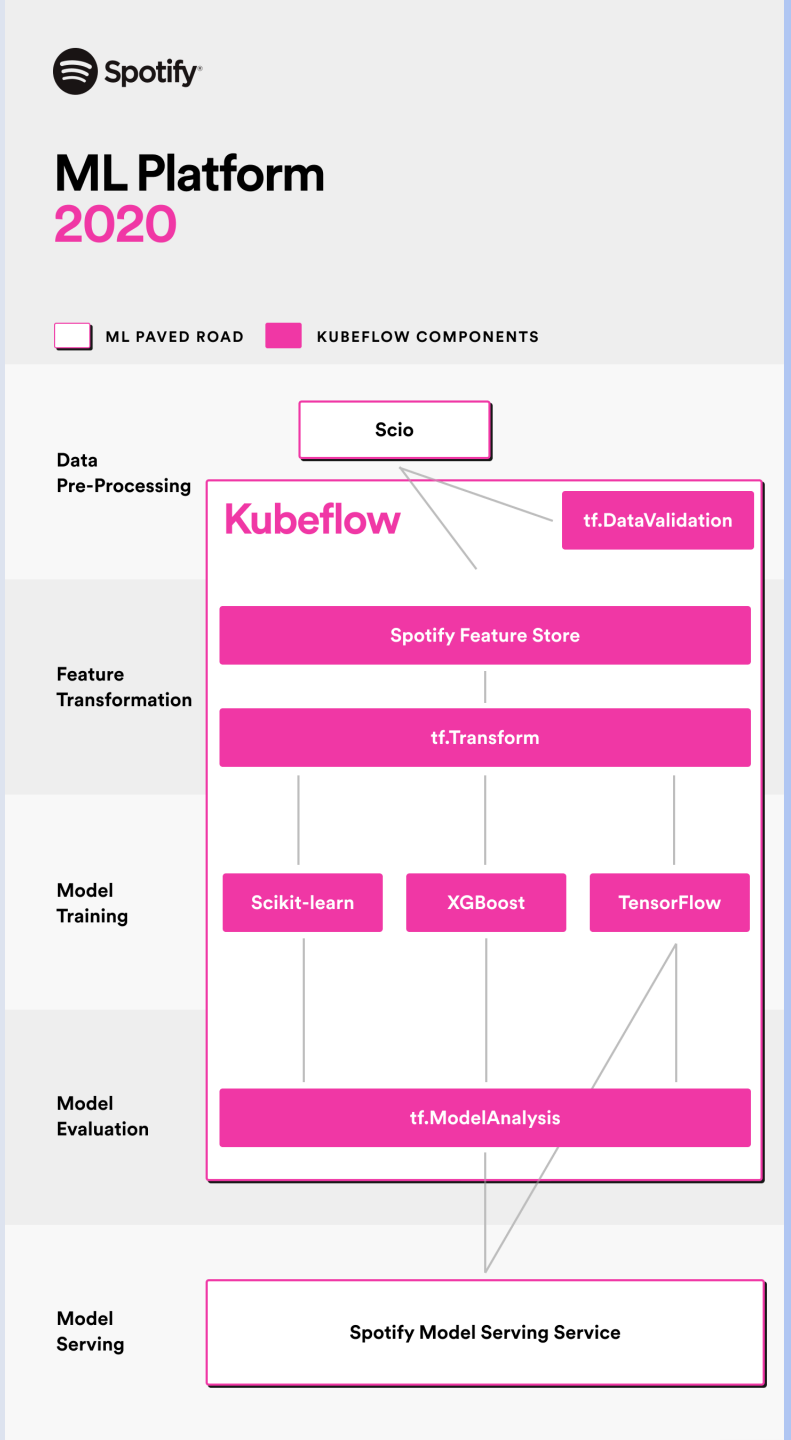
The Second Iteration

- **Problem**
  - Scala language was heavily used, but ML community was focusing on Python
- **Solution**
  - slowly shift Spotify ML engineers
  - starting from the use of some TFX open-source tools by Google



The Third Iteration

- **Problem**
  - nothing tied together the disparate tooling and provided the **end-to-end experience**
  - difficulties in **keeping track of independent machine learning experiments** and
  - difficulties in **comparing results from different approaches**
- **Solution**
  - Introduce and switch teams over to **Kubeflow Pipelines**



Agenda

- Automation and orchestration
- Kubeflow
- **Airflow**

Bits of history

- The Airflow Project was **started in October 2014** by Maxime Beauchemin at **Airbnb** and officially brought under the Airbnb GitHub in June 2015.
- The project **joined the Apache Software Foundation's Incubator program in March 2016** and the Foundation announced Apache Airflow as a Top-Level Project in January 2019.

Old Logo

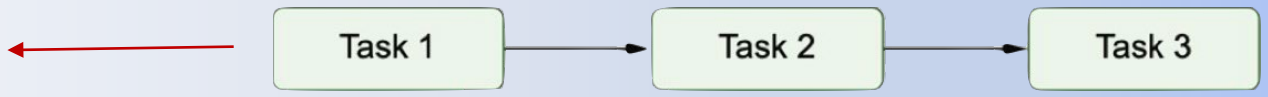
New Logo

What is Apache Airflow?

- **Batch-oriented** framework for developing, scheduling and monitoring **data pipelines.**
- **Python** coding for **dynamic** pipeline generation.
- **Open source** and **extensible** to various technologies.
- **Flexible** thanks to workflow **parameterization**.

What is Apache Airflow?

- **Batch-oriented** framework for developing, scheduling and monitoring **data pipelines.**
- **Python** coding for **dynamic** pipeline generation.
- **Open source** and **extensible** to various technologies.
- **Flexible** thanks to workflow **parameterization**.

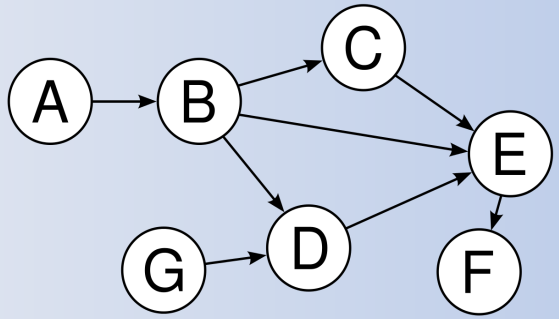


Continuous data flows

Discretized flows from Apache Kafka

What is Apache Airflow?

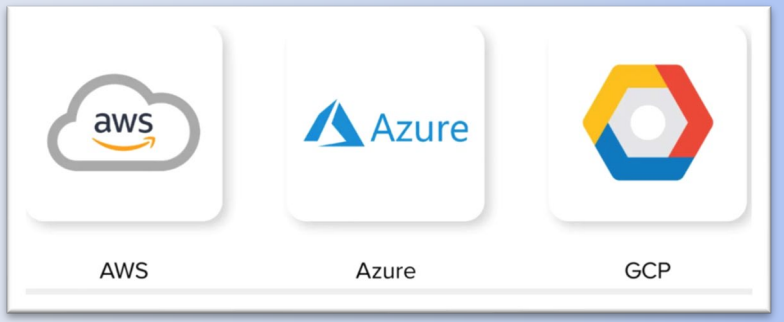
- **Batch-oriented** framework for developing, scheduling and monitoring **data pipelines.**
- **Python** coding for **dynamic** pipeline generation.
- **Open source** and **extensible** to various technologies.
- **Flexible** thanks to workflow **parameterization**.



- DAG tasks can vary based on the run
- Developer-friendly
- No drag-and-drop UI
- Large code gets complex very fast

What is Apache Airflow?

- **Batch-oriented** framework for developing, scheduling and monitoring **data pipelines.**
- **Python** coding for **dynamic** pipeline generation.
- **Open source** and **extensible** to various technologies.
- **Flexible** thanks to workflow **parameterization**.



- Completely free
- Active community & plug-ins
- No vendor lock-in

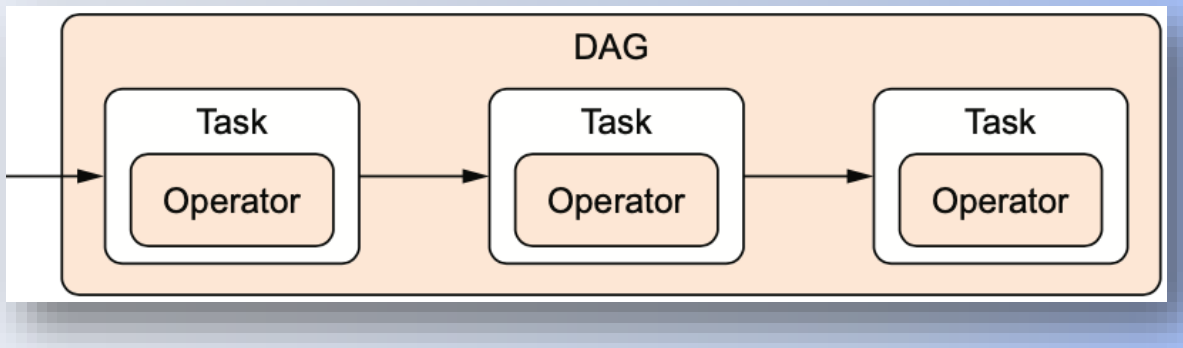
What is Apache Airflow?

- **Batch-oriented** framework for developing, scheduling and monitoring **data pipelines.**
- **Python** coding for **dynamic** pipeline generation.
- **Open source** and **extensible** to various technologies.
- **Flexible** thanks to workflow **parameterization**.

- Create placeholders for variables
- Set values later during rendering

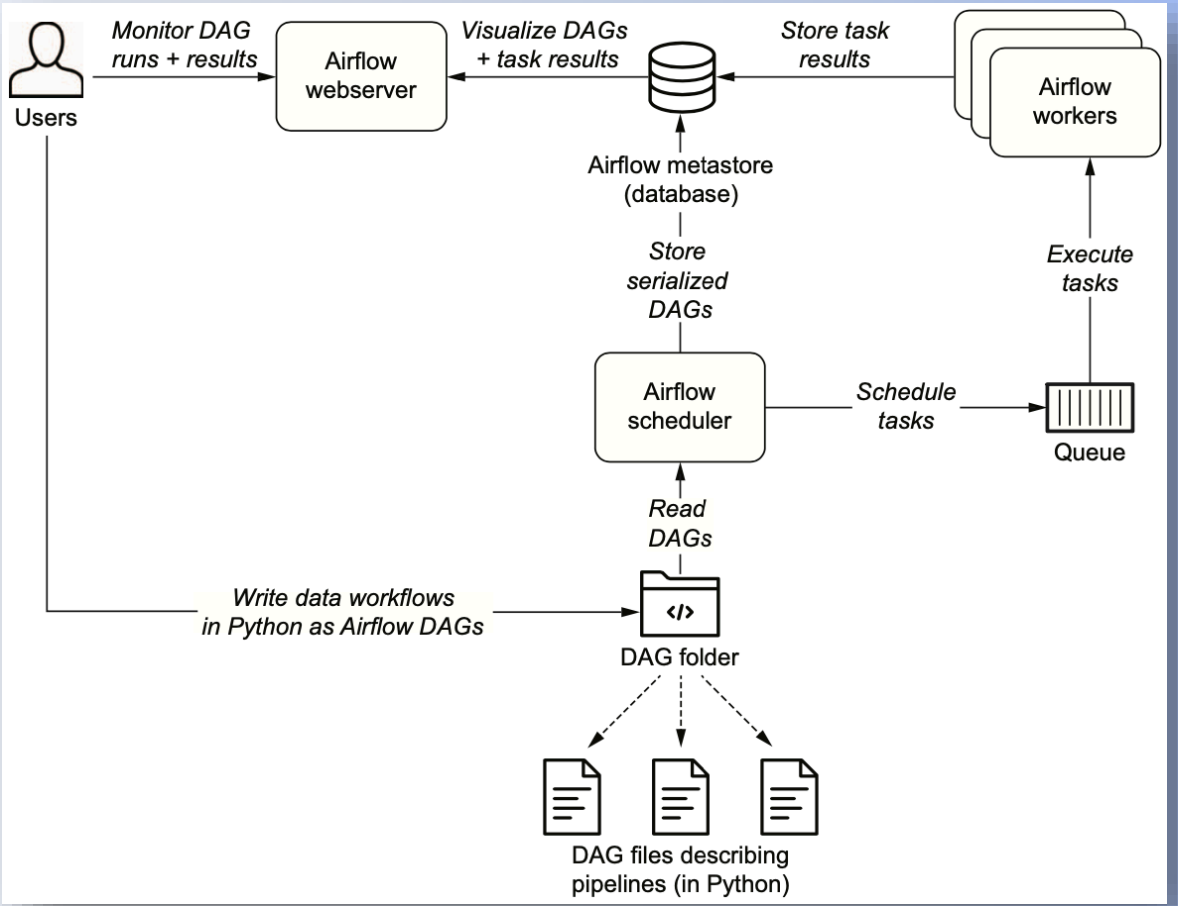
Basic terminology

- **Operator**: template that performs a single operation (e.g. *EmailOperator or PythonOperator*).
- **Task**: upon satisfaction of its dependencies monitors the execution of an operator.
- **DAG**: pipeline that contains a series of tasks that may branch but not loop.



Components and architecture

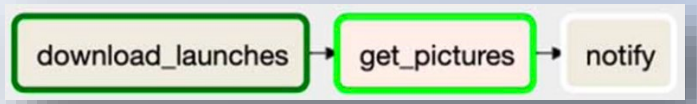
- **Scheduler**: parses DAGs and schedules task executions in a queue.
- **Executor**: assigns pending tasks in the queue to worker nodes for execution.
- **Webserver**: provides a user interface to monitor DAG runs and their results.
- **Metastore**: database (by default SQLite) that stores metadata about security, DAG configuration and logs.



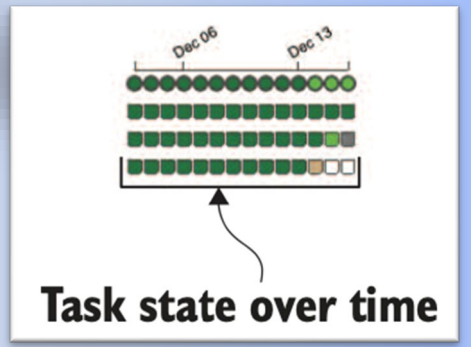
Our first DAG

- **Trigger**: runs a task if certain conditions are satisfied.
- **Linear dependencies**: tasks are run sequentially.

Success → Running → Wait

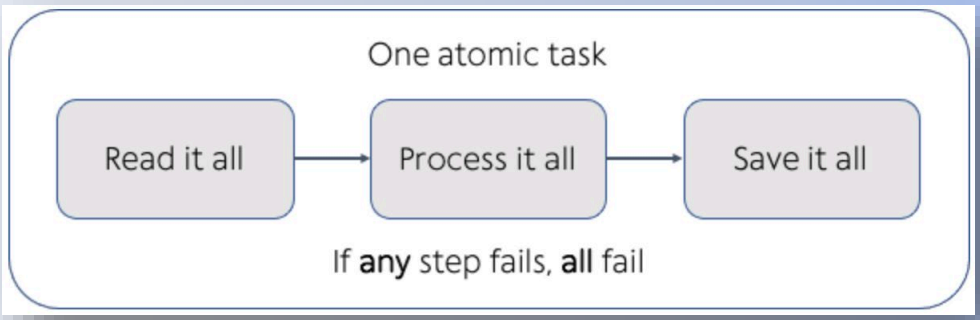


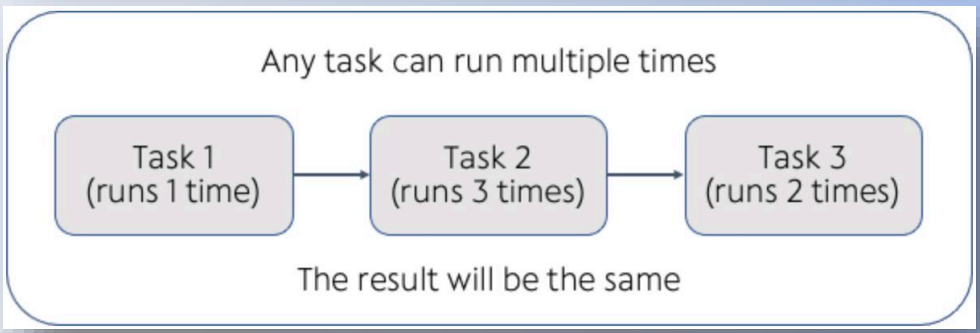
Tree view



Best practices

- **Atomicity**: each task must be composed of an all-or-nothing series of independent operations.
- **Idempotency**: running the same task multiple times with the same inputs must have no additional effect.
- Most of the operators in Airflow already are atomic and idempotent by design, but more flexible ones like PythonOperators and BashOperators are **responsibility of the developer**.

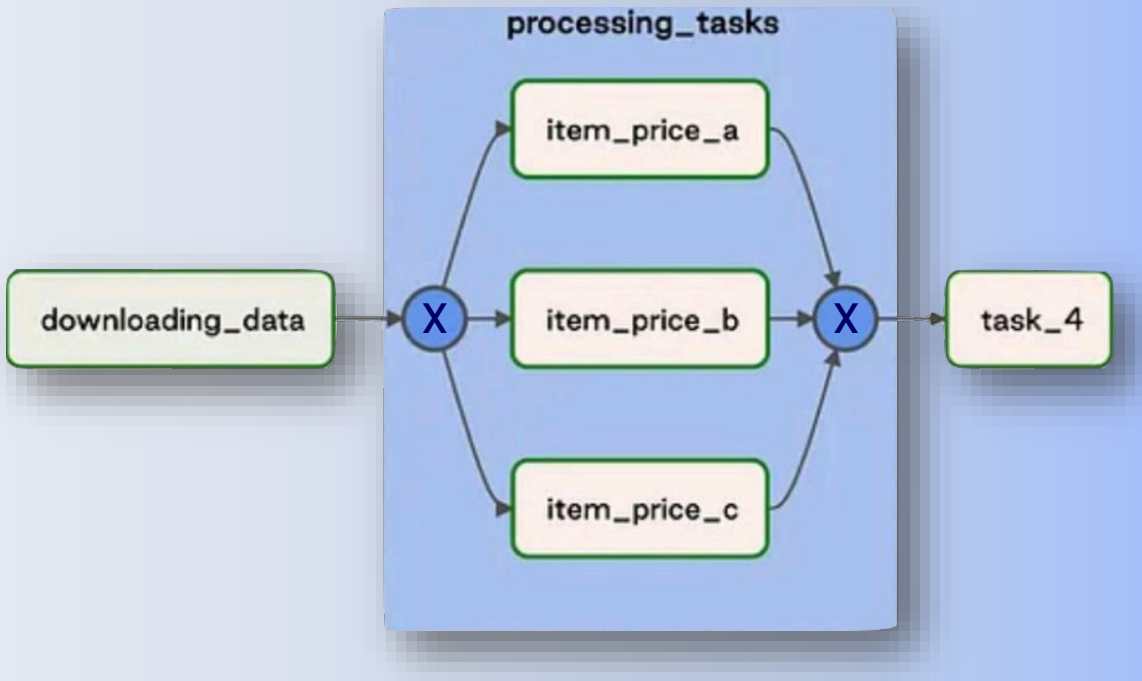




More complex DAGs

- **Fan-in/out** dependencies: allows for parallel task executions and multi-trigger conditions per task.
- **Conditional task**: triggers can prevent a task from being executed (DAG branches are pruned).
- **XCom** (cross-communications): shares small data objects (key-values with timestamp) between tasks.
- **Taskflow API**: for even easier data sharing capabilities between *PythonOperators*.

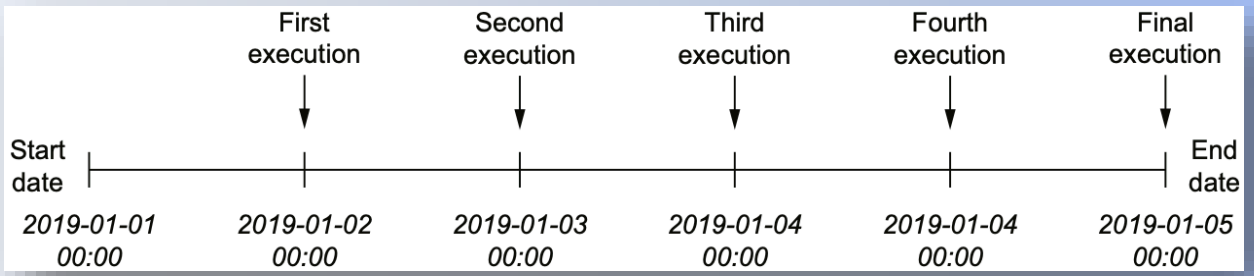
Fan-in/out DAG with XCOMS



Triggering DAGs

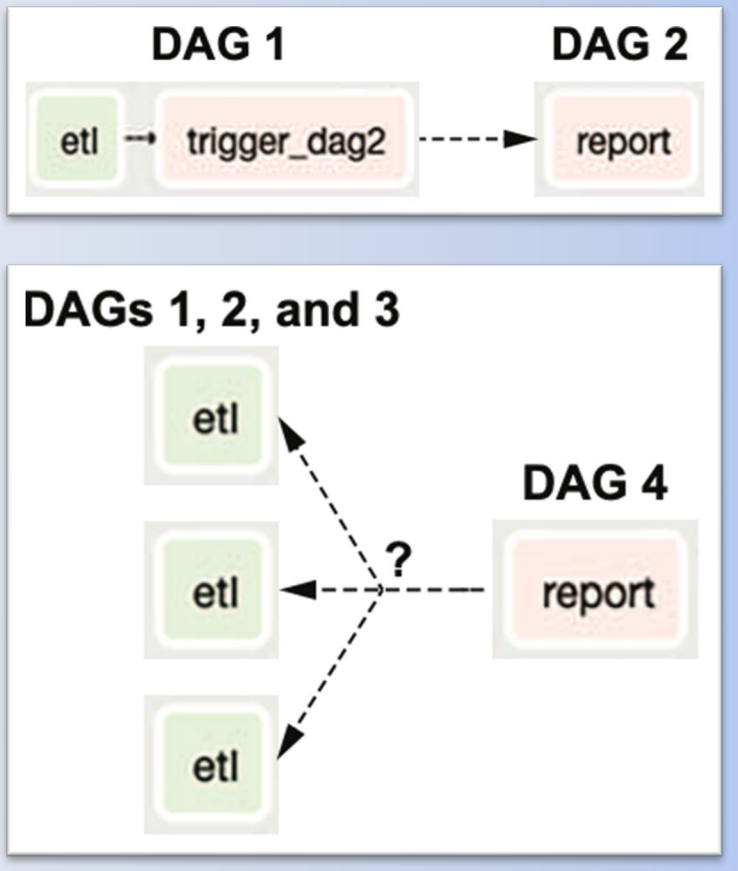
- **Event-based trigger**: the DAG is executed automatically in response to some event.
- **Schedule-based trigger**: the DAG is recurrently executed based on a schedule.
- **Manual trigger**: executes the DAG once disregarding any schedules.

Execution scheduled on a daily trigger



DAG event-based triggering

- An **ad-hoc operator** can used to run DAGs sequentially (*TriggerDagRunOperator*), as well as REST API or CLI calls.
- DAGs cannot create loops per se, but **a DAG can trigger another DAG multiple times**.
- Since DAGs are unaware of each other, **multi-trigger DAG execution needs to use sensors** (*ExternalTaskSensor*).
- **A sensor is a special operator that pokes a specific task** (identified by its ID and execution date) to check periodically its execution status.

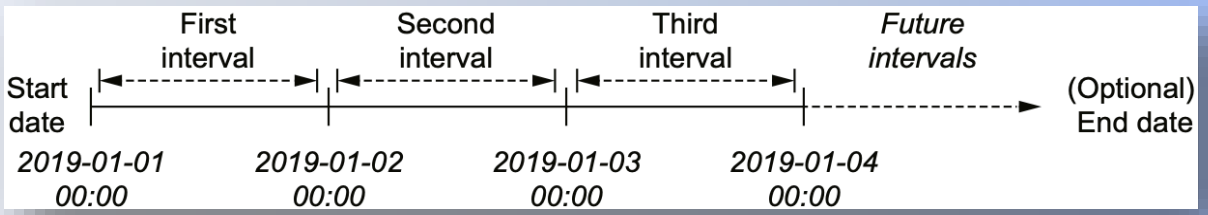


DAG schedule-based triggering

- **Interval-based scheduling**: frequency based (run at each timedelta).
  *Note that the first execution date effectively begins at **start_date + timedelta** and so on.*

- **Time point-based scheduling**: date based (run at each occurrence of the desired **Cron** expression). The implementation extends on *croniter* to add **time zone awareness**.

- Both can be scheduled with **backfilling** (*catchup=True*), which instantly executes in succession all the past (skipped) executions from the starting date.




```
┌─────── minute (0 - 59)
│ ┌─────── hour (0 - 23)
│ │ ┌─────── day of the month (1 - 31)
│ │ │ ┌─────── month (1 - 12)
│ │ │ │ ┌─────── day of the week (0 - 6) (Sunday to Saturday;
│ │ │ │ │       7 is also Sunday on some systems)
* * * * *
0 0 * * MON,WED,FRI = run every Monday, Wednesday, Friday at midnight
0 0 * * MON-FRI = run every weekday at midnight
0 0,12 * * * = run every day at 00:00 and 12:00
```

Execution parameters

- **Callback**: sends an email alert if the execution fails.
- **Timeout**: interrupts the execution if it takes longer than a specific threshold.
- **SLA**: sends an email alert if the execution takes longer than a specific threshold (but does not interrupt it). Counterintuitively, the elapsed time is calculated from the DAG starting date and not from the effective task execution date.
- Timeout and SLA can be applied **either to single tasks or the entire DAG** (timeout can be defined also for sensors).
- A failed task execution can be **restarted directly from the point of failure onwards**.

SLA alert curious default template

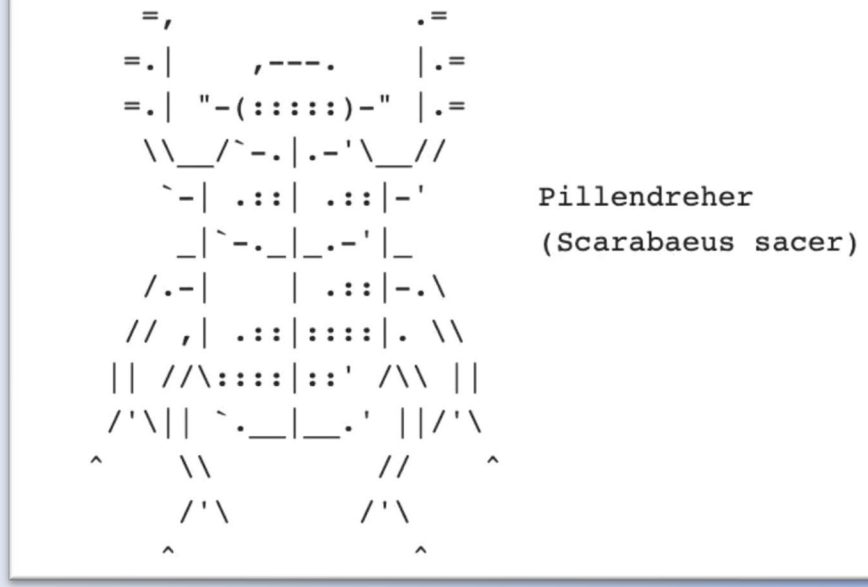
to me

Here's a list of tasks that missed their SLAs:

```
mid on 2019-10-15T17:13:27.135198+00:00
```

Blocking tasks:

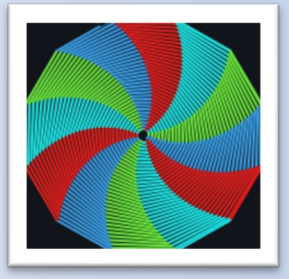
```
before on 2019-10-15T17:13:27.135198+00:00
```



DAG Testing

Parallel testing support

- **DAG integrity test**: verifies if the DAG is structurally correct (no cycles, unique IDs…).
- **Unit test**: verifies if there is any incorrect code in the DAG.
- **DTAP paradigm**: set four isolated environments (development, test, acceptance and production).



Whirl

Variable masking (simulation)

Docker Containers usage

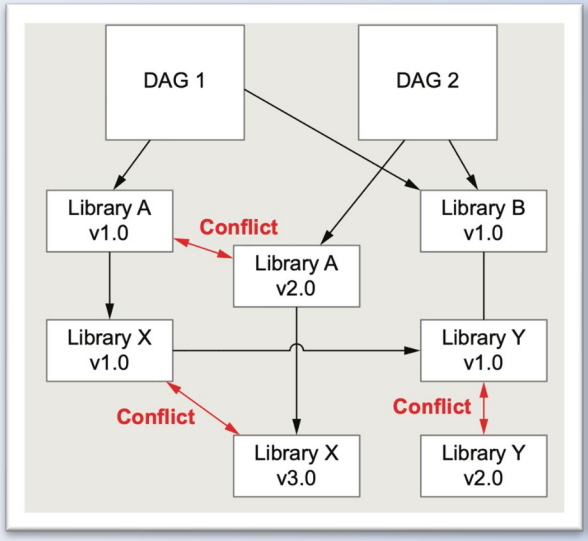
Versioning

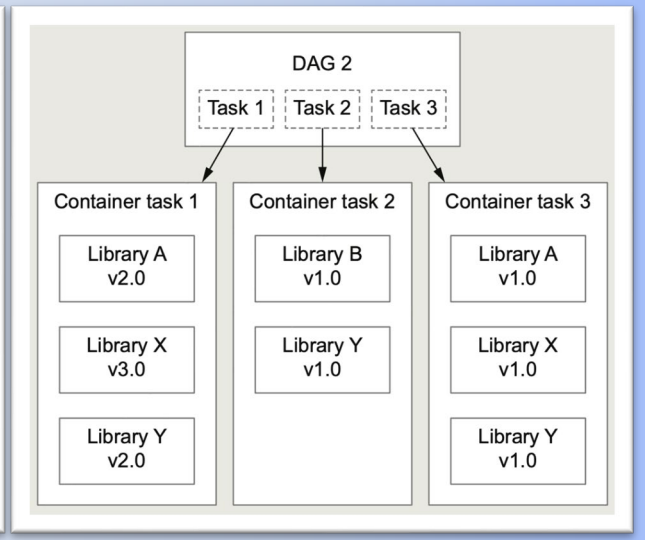
Collaboration

Airflow executor types

- **SequentialExecutor**: runs sequential tasks on a single machine (default option and not scalable).
- **LocalExecutor**: runs parallel tasks on a single machine (with FIFO method).
- **CeleryExecutor**: runs parallel tasks on multiple machines (using Celery).
- **KubernetesExecutor**: runs parallel tasks on multiple machines (using Kubernetes).

- Use only *KubernetesPodOperator* chains
- Work with Docker images
- Easier testing and version control
- Avoid library conflicts

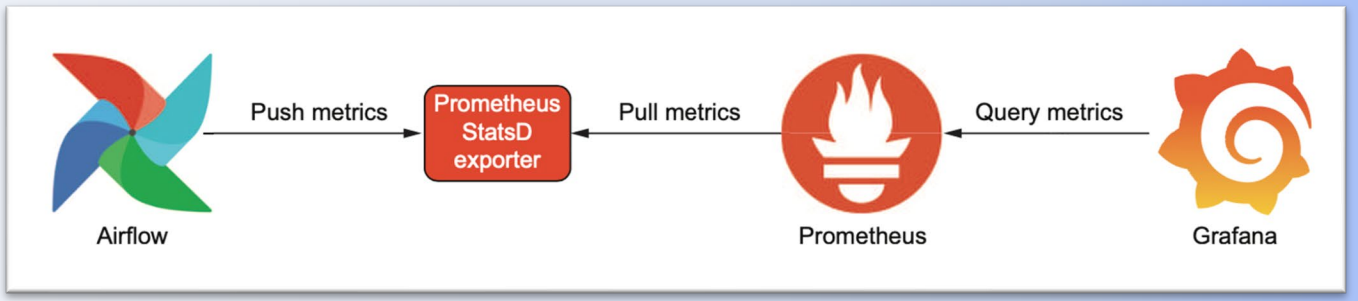




StatsD metric monitoring

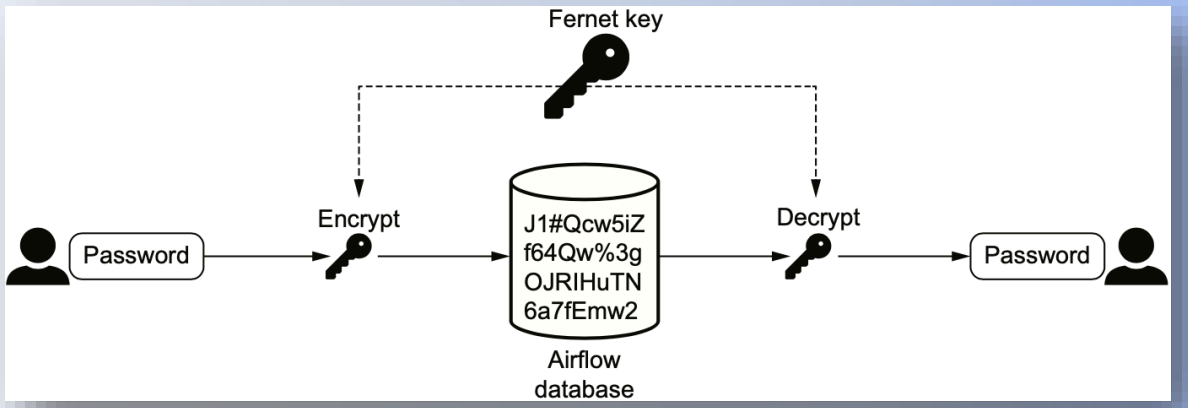
- **Latency**: request handling speed.
- **Traffic**: demand from the system.
- **Errors**: raised errors.
- **Saturation**: system capacity utilization.

StatsD can also used to integrate other tools



Security best practices

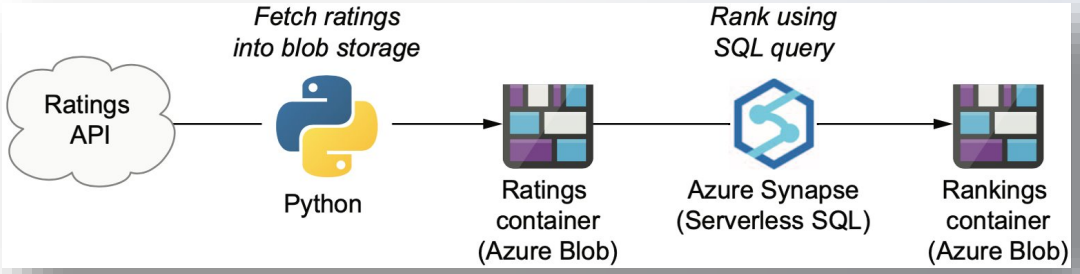
- Ensure the Airflow webserver is not accessible publicly (create a whitelist or avoid having an external IP).
- Use SSL certificates to enable encrypted connections via Fernet private key.
- Set role-based access control (RBAC) to restrict access, by defining roles and permissions.

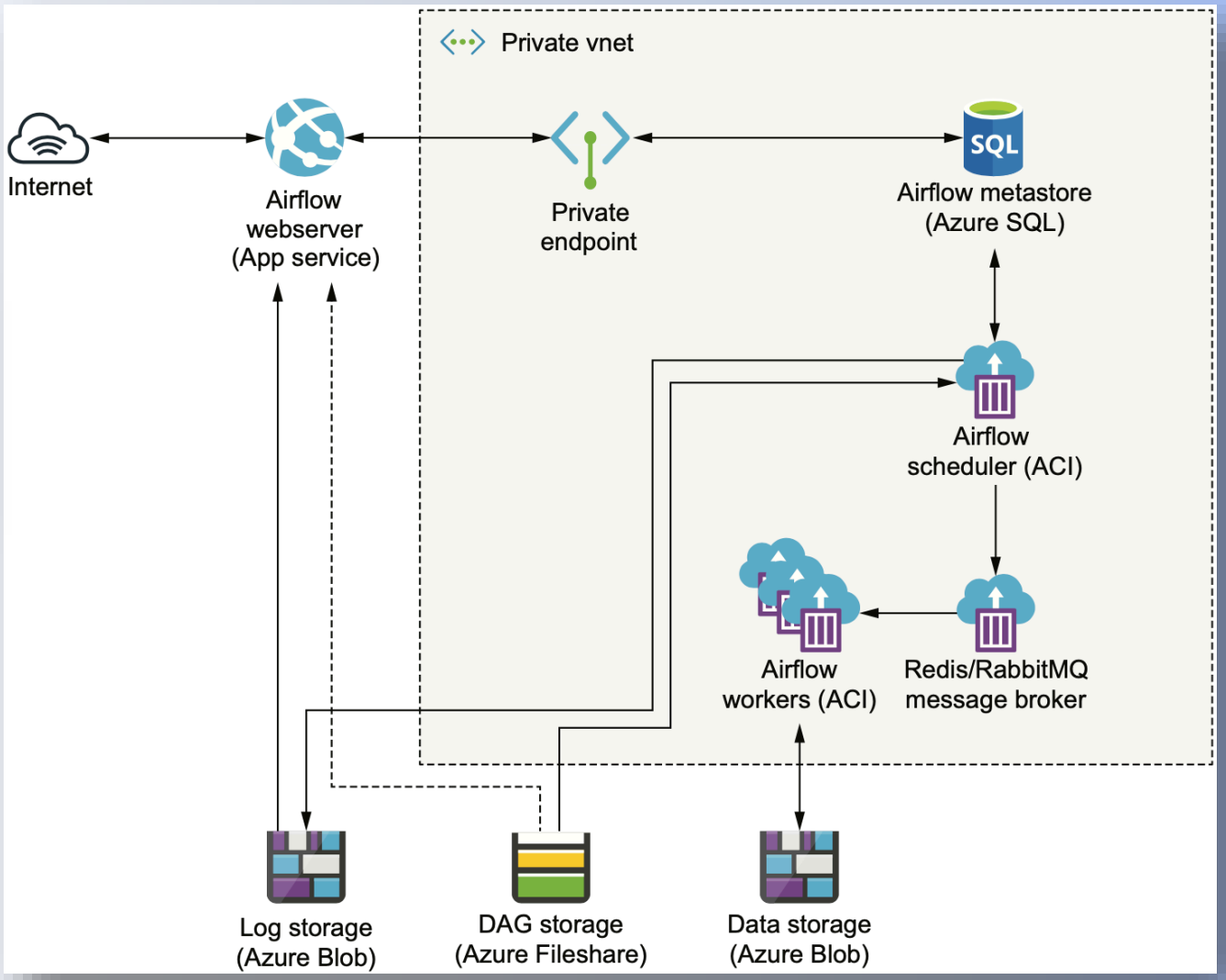


Cloud integration

- **Amazon Managed Workflows for Apache Airflow** (AWS).
- **Google Cloud Composer** (Google Cloud Platform).
- **Azure Data Factory Managed Airflow** (Azure).
- **Astro** (Astronomer).

Azure-based Airflow use case





Bibliography

1. Patterson, J., Katzenellenbogen, M., & Harris, A. (2020). *Kubeflow Operations Guide. O'Reilly Media*. https://books.google.it/books?id=bd0MEAAAQBAJ

2. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). *Hidden Technical Debt in Machine Learning Systems*. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 28). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

3. Wikipedia contributors. (2022, December 25). *Technical debt. Wikipedia*. https://en.wikipedia.org/wiki/Technical_debt

4. Kästner, C. (2022, January 5). *Technical Debt in Machine-Learning Systems* - Christian Kästner - Medium. Medium. https://ckaestne.medium.com/technical-debt-in-machine-learning-systems-62035b82b6de

5. Globaldots, A. (2021, August 8). *Kubeflow: Concepts, Use Cases and Starter's Guide*. GlobalDots. https://www.globaldots.com/resources/how-to/kubeflow-concepts-use-cases-and-starters-guide/#:~:text=Portability%20in%20Kubeflow%20means%20that,Training%20Rig%2C%20or%20the%20Cloud

6. Xu, J. (2021, June 4). *Why Implementing Kubernetes Operators Is a Good Idea!* Kubermatic. https://www.kubermatic.com/blog/why-implementing-kubernetes-operators-is-a-good-idea/

Bibliography

7. K. (2021, March 10). *Kubeflow Katib: Scalable, Portable and Cloud Native System for AutoML*. Kubeflow. https://blog.kubeflow.org/katib/

8. *What is Model Serving?* (8 B.C.E.). Iguazio. https://www.iguazio.com/glossary/model-serving/

9. *What is Model Deployment* | Iguazio. (2022, October 12). Iguazio. https://www.iguazio.com/glossary/model-deployment/

10. *What is Model Deployment*. (n.d.). https://valohai.com/model-deployment/

11. Engineering, S. (2022, January 27). *The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow* - Spotify Engineering %. Spotify Engineering. https://engineering.atspotify.com/2019/12/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

12. https://airflow.apache.org/docs/apache-airflow/stable/

13. https://github.com/apache/airflow/

Bibliography

16. https://github.com/godatadriven/whirl

17. https://jinja.palletsprojects.com/

18. https://docs.astronomer.io/

19. https://docs.pytest.org/

20. https://www.alibabacloud.com/blog/pull-or-push-how-to-select-monitoring-systems_599007

21. https://research.aimultiple.com/mlops-vs-dataops/

22. https://phoenixnap.com/blog/orchestration-vsautomation#:~:text=Automation%20refers%20to%20automating%20a,integrating%20it%20with%20other%20systems)

23. https://phoenixnap.com/blog/orchestration-vsautomation#:~:text=Automation%20refers%20to%20automating%20a,integrating%20it%20with%20other%20systems)

Bibliography

24. https://towardsdatascience.com/workflow-orchestration-vs-data-orchestration-are-those-differenta661c46d2e88
25. https://towardsdatascience.com/a-complete-guide-to-understanding-data-orchestration87a20b46297c
26. https://medium.com/memory-leak/data-orchestration-a-primer-56f3ddbb1700
27. https://www.astronomer.io/blog/what-is-data-orchestration/
28. https://www.databricks.com/it/glossary/orchestration
29. https://www.databricks.com/blog/2021/11/01/now-generally-available-simple-data-and-machine-learning-pipelines-with-job-orchestration.html
30. https://www.mongodb.com/basics/data-stack
31. https://www.mongodb.com/basics/technology-stack

Bibliography

32. https://www.talend.com/it/resources/elt-vs-etl/

33. https://www.techtarget.com/searchdatamanagement/definition/data-silo

34. https://www.stonebranch.com/blog/what-is-a-dataops-orchestration-solution

35. https://madewithml.com/courses/mlops/orchestration/