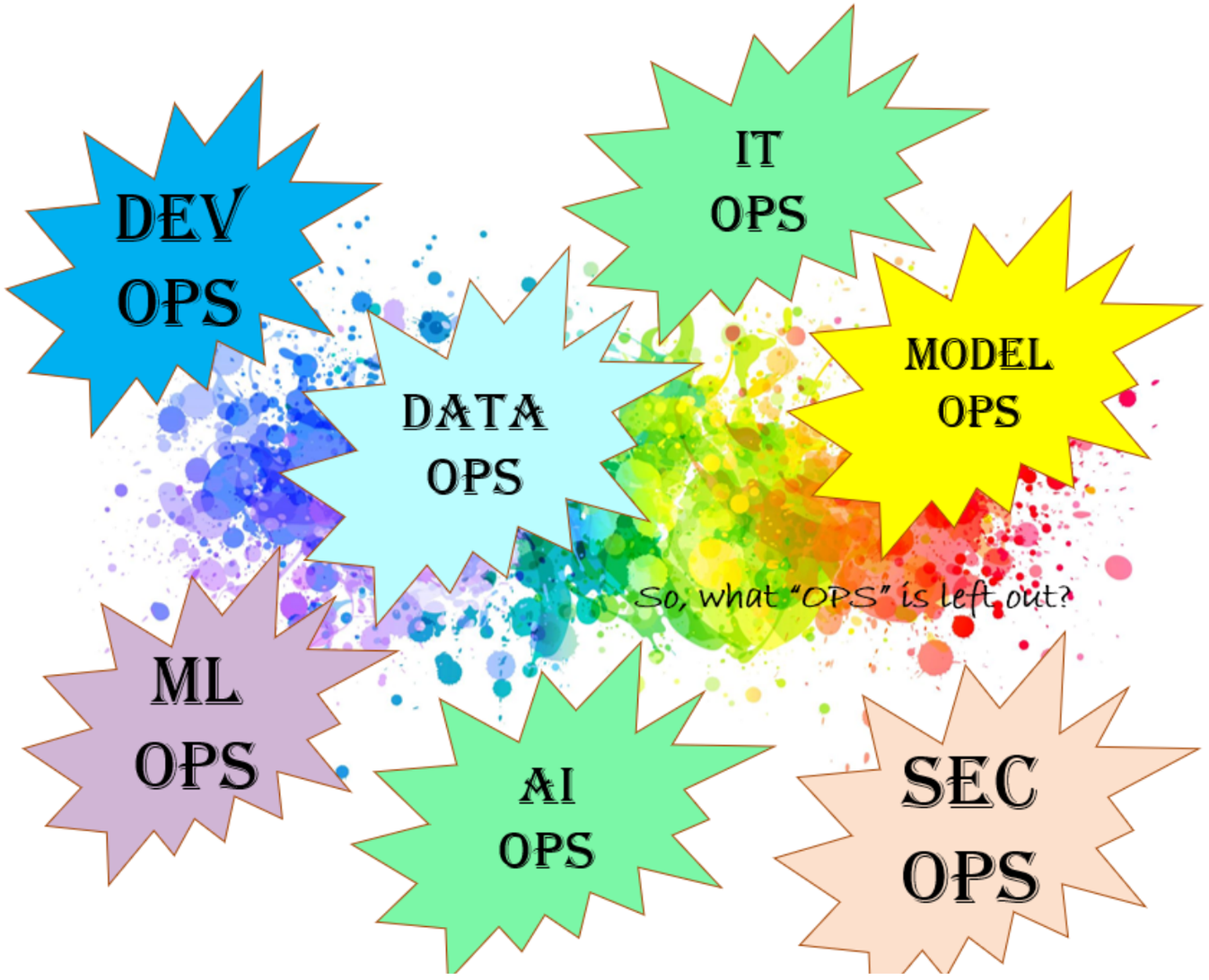
Because of you,

AGILE,

I Need to Learn --->



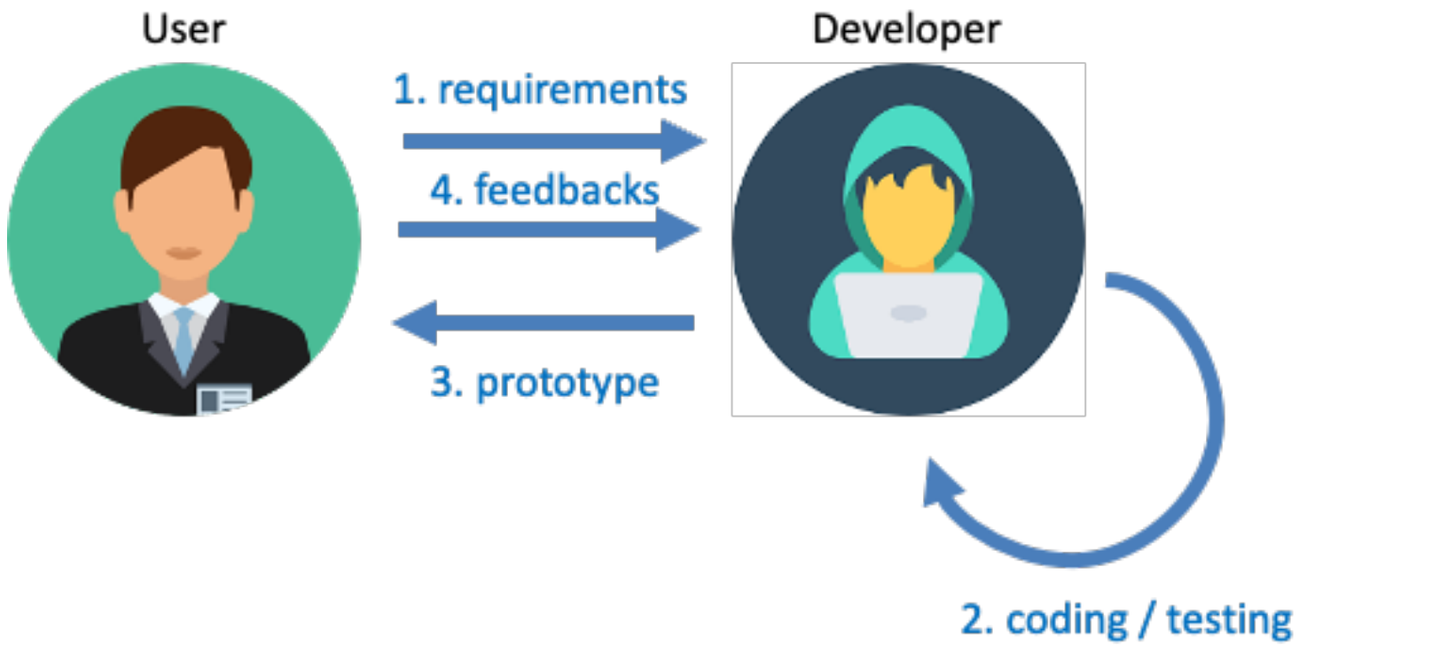
2110446 - Data Science and Data Engineering

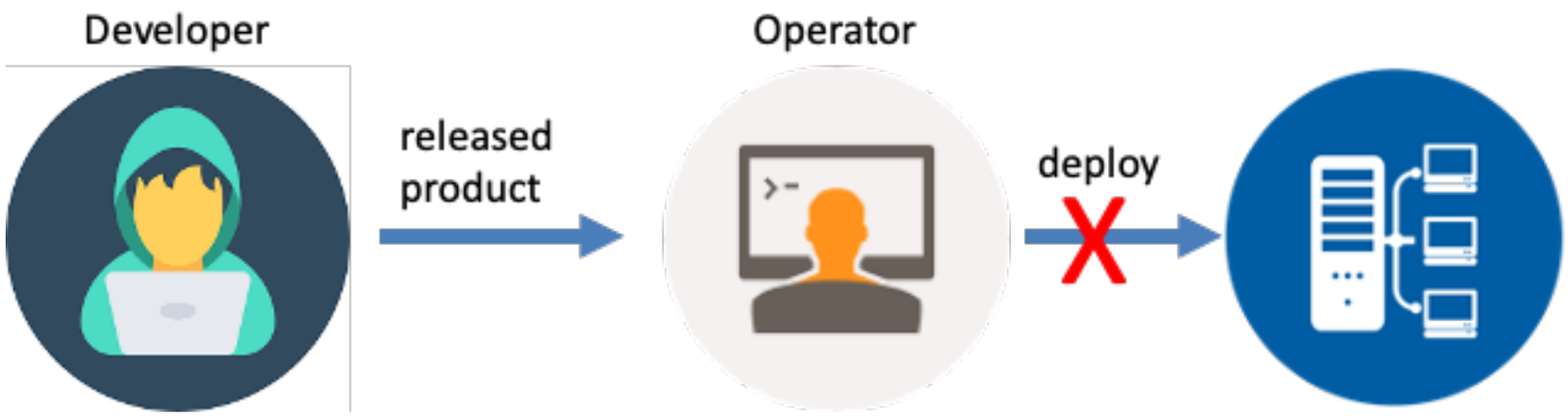
Ops Stars

Asst.Prof. Natawut Nupairoj, Ph.D.
Department of Computer Engineering
Chulalongkorn University
natawut.n@chula.ac.th

CHULA ENGINEERING
Foundation toward Innovation

Agile Development and the Real World Problems

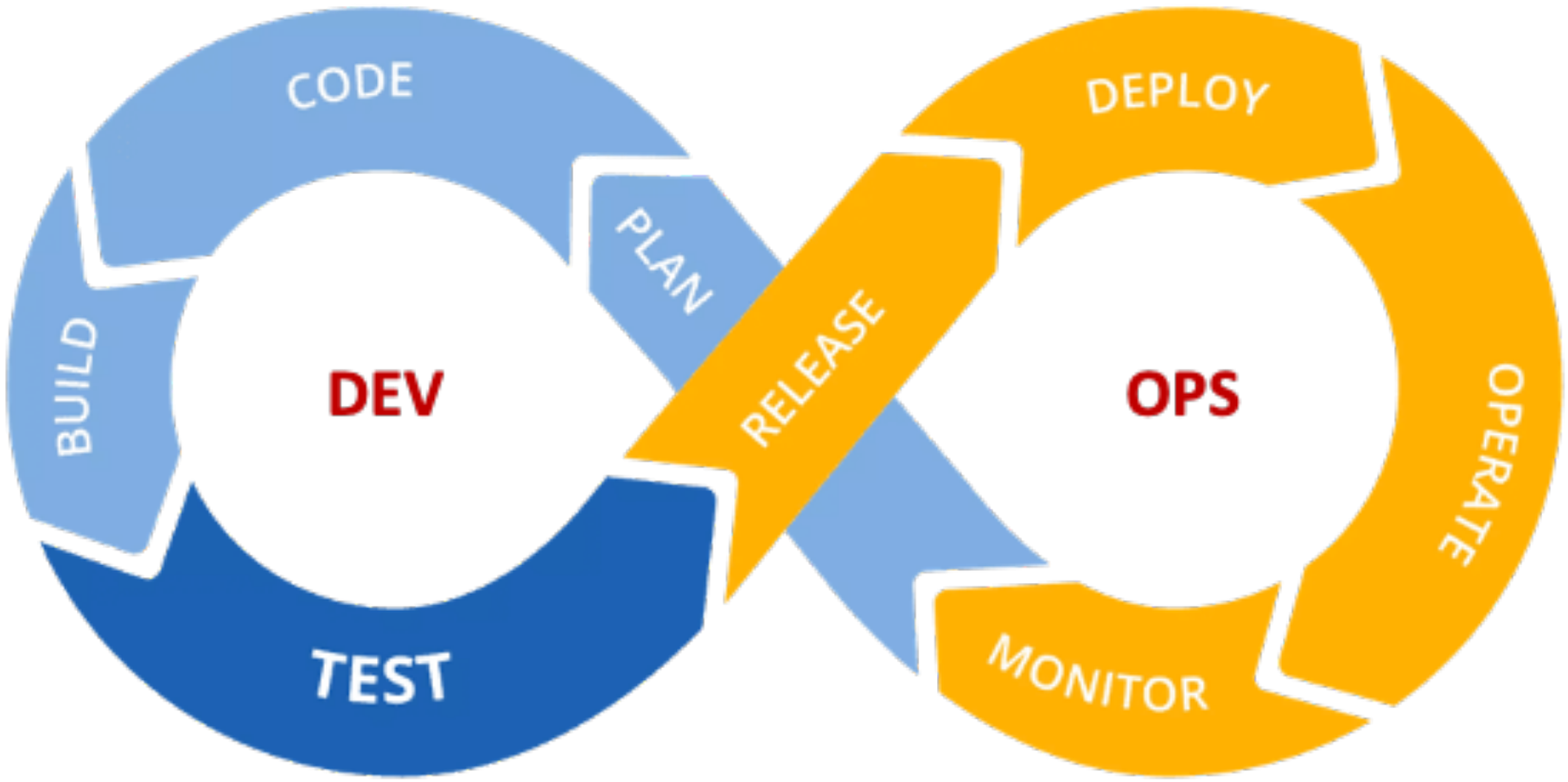




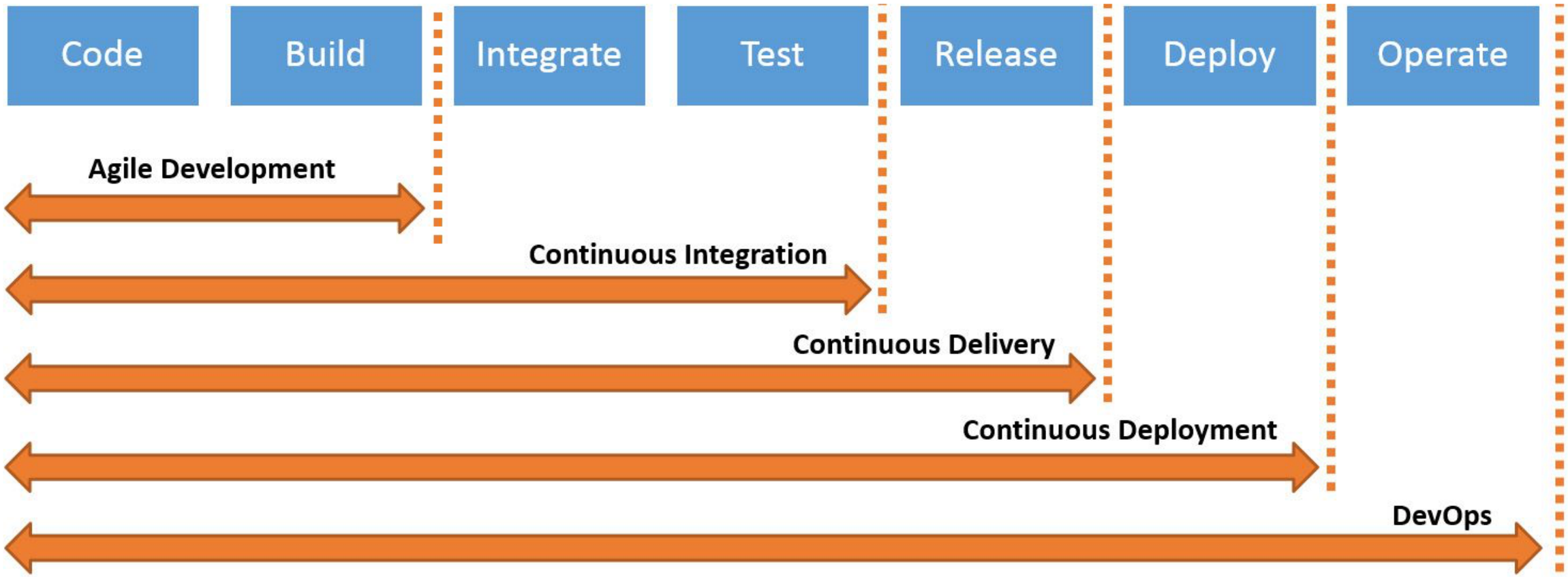
Solution: DevOps

- A set of practices that combines software development (Dev) and IT operations (Ops)
- Aim to shorten SDLC with continuous delivery and high software quality
  - Tackle deployment architecture at early stages
  - Involve everyone in SDLC in a continuous fashion
  - Automate all necessary steps as much as and as early as possible
- Software-defined system (Infrastructure as Code) plays a very important role to ensure smooth integration and delivery

The Infinite Lifecycle

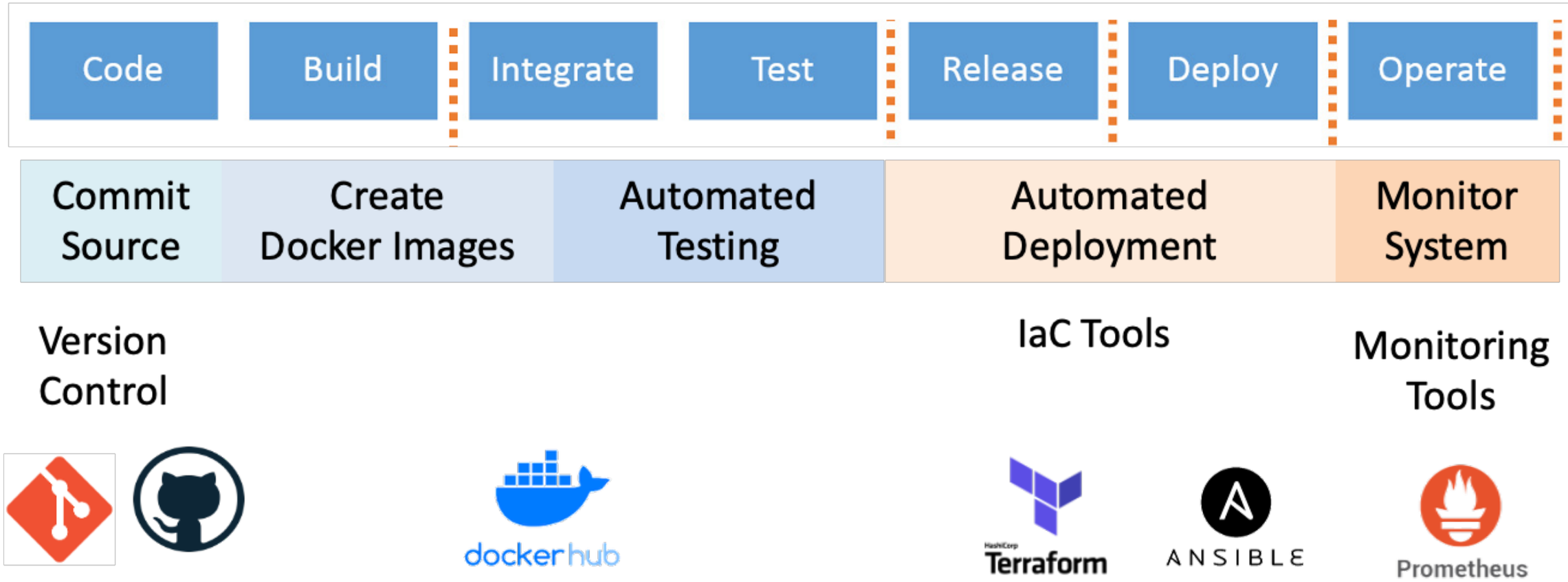


DevOps Automation

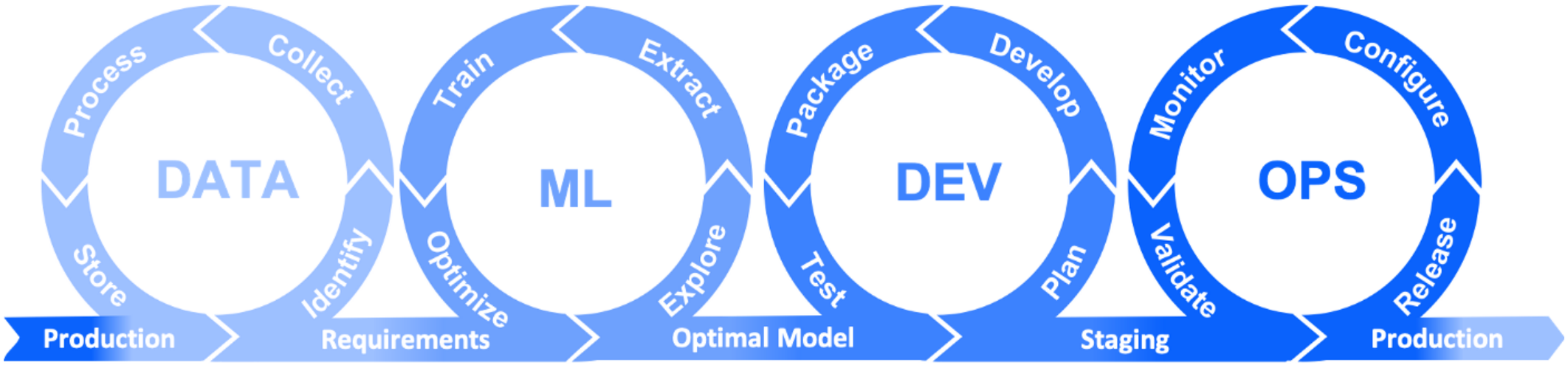


Source: https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/

DevOps requires End-to-End Automation Toolchain

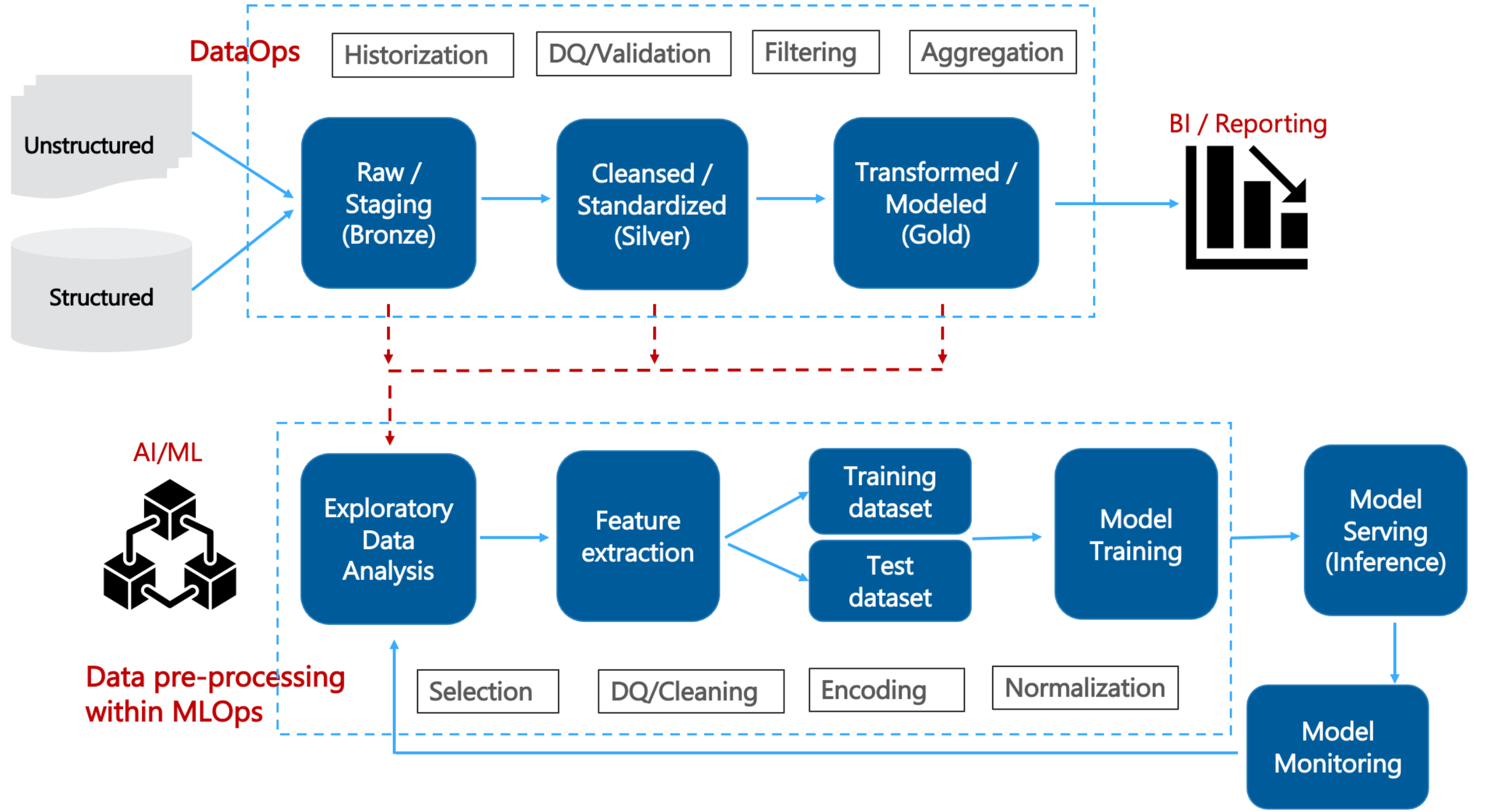


With ML and Data Science, these cycles become more complicated



Source: https://medium.com/@bhandaripallavi.98/mlops-ease-collaboration-between-data-scientists-and-it-ops-e7e3ccac9eed

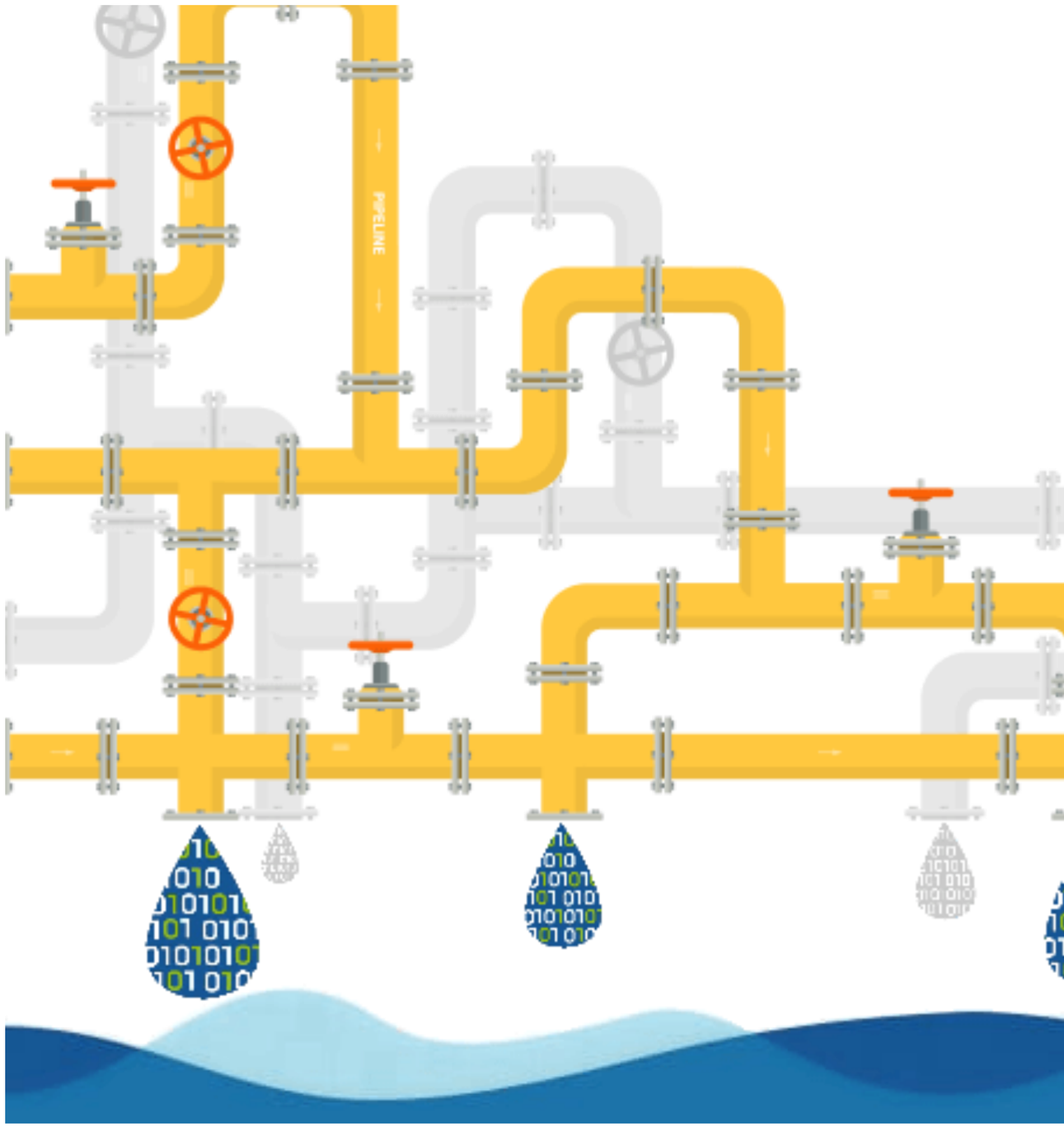
DataOps and MLOps



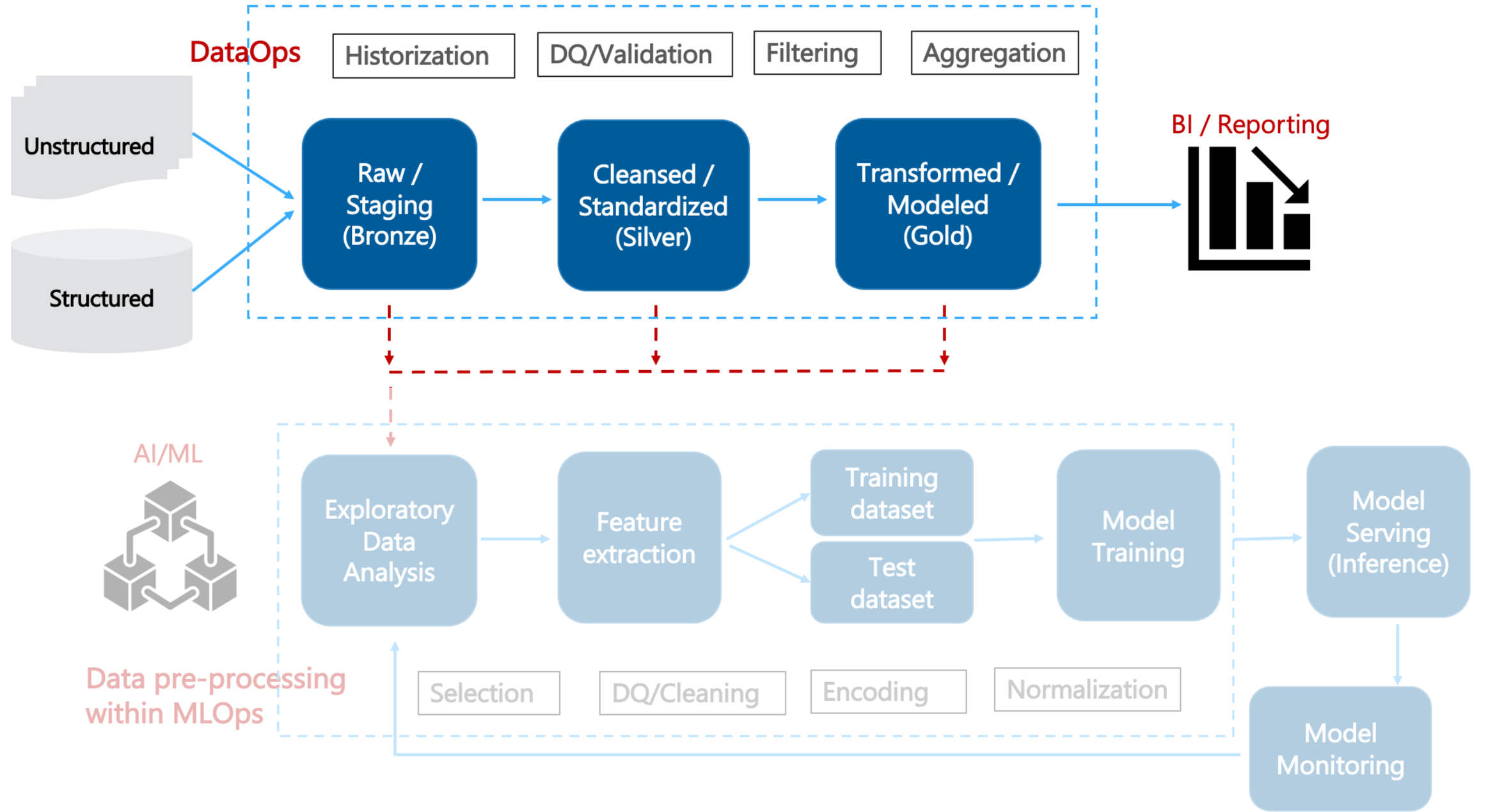
Source: https://towardsdatascience.com/bridging-dataops-mlops-301f010caf30

Orchestrating Data Pipeline for DataOps

Ops Stars



DataOps and MLOps

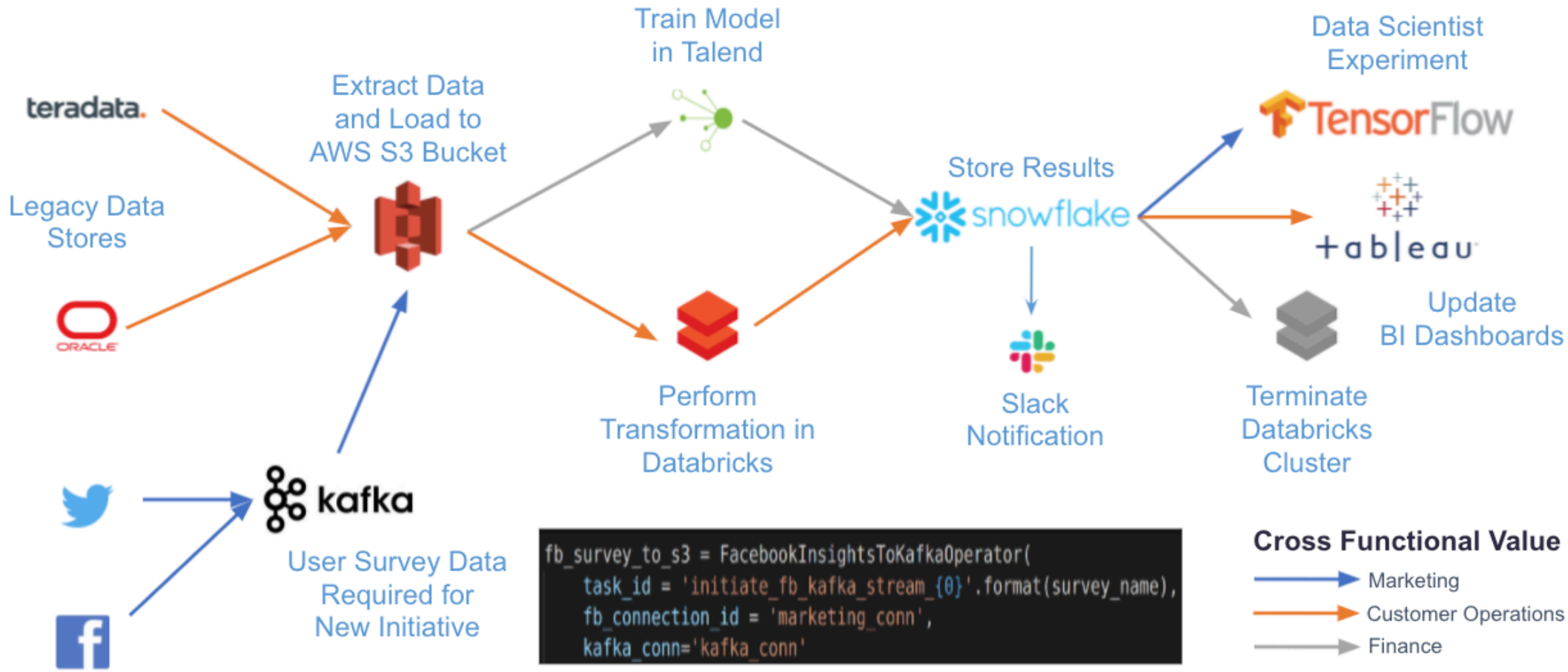


Source: https://towardsdatascience.com/bridging-dataops-mlops-301f010caf30

Apache Airflow

- Open-source workflow management platform for data engineering pipelines, started by Airbnb in 2014
- Designed under the principle of "configuration as code”
  - Uses directed acyclic graphs (DAGs) to manage workflow orchestration
    - Tasks and dependencies are defined in Python (possible for just one file)
    - DAG can be started by defined calendar schedule or triggered by external event
    - Airflow manages the scheduling and execution
- Provide managing and monitoring facility with web-based UI

Flexibility of Data Pipelines as Code



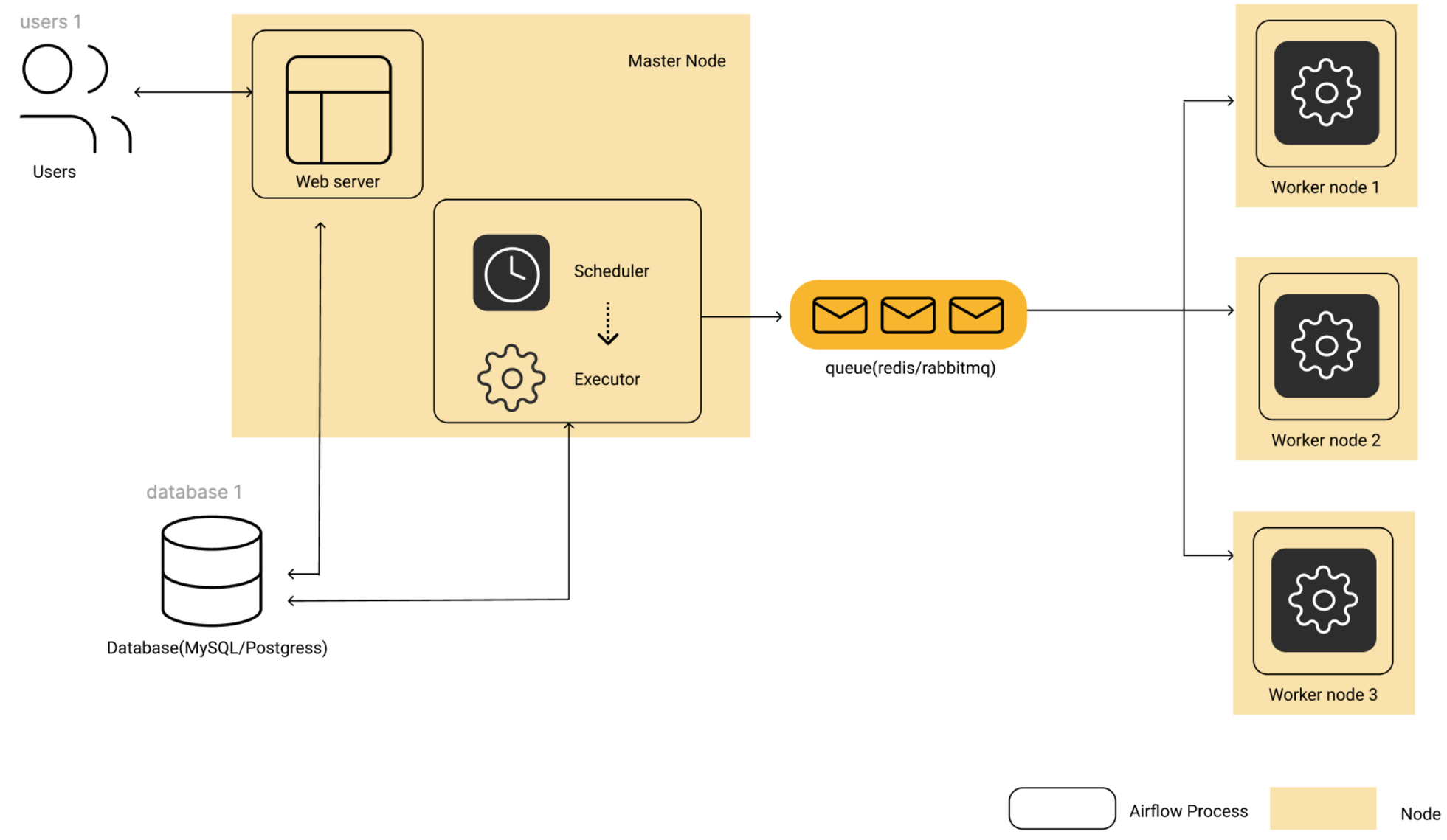
Complex pipeline can be implemented using operators, provider packages, and external services — with minimal code writing

Provider Packages

Providers packages include integrations with third party projects. They are updated independently of the Apache Airflow core. Read the documentation >>

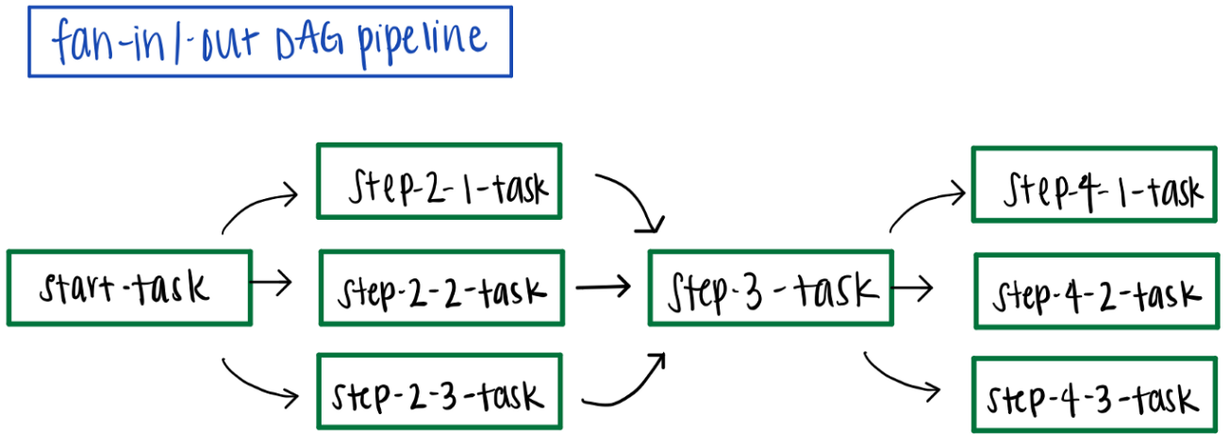
- Airbyte
- Alibaba
- Amazon
- Apache Beam
- Apache Cassandra
- Apache Drill
- Apache Druid
- Apache HDFS
- Apache Hive
- Apache Kylin
- Apache Livy
- Apache Pig
- Apache Pinot
- Apache Spark
- Apache Sqoop
- Asana
- Celery
- IBM Cloudant
- Kubernetes
- Databricks
- Datadog
- DBT cloud
- Dingding
- Discord
- Docker
- Elasticsearch
- Exasol
- Facebook
- File Transfer Protocol (FTP)
- Github
- Google
- gRPC
- Hashicorp
- Hypertext Transfer Protocol (HTTP)
- Influx DB
- Internet Message Access Protocol (IMAP)
- Java Database Connectivity (JDBC)
- Jenkins
- Jira
- Microsoft Azure
- Microsoft PowerShell Remoting Protocol (PSRP)
- Microsoft SQL Server (MSSQL)
- Windows Remote Management (WinRM)
- MongoDB
- MySQL
- Neo4J
- ODBC
- OpenFaaS
- Opsgenie
- Oracle
- Pagerduty
- Papermill
- Plexus
- PostgreSQL
- Presto
- Qubole
- Redis
- Salesforce
- Samba
- Segment
- Sendgrid
- SFTP
- Singularity
- Slack
- Snowflake
- SQLite
- SSH
- Tableau
- Telegram
- Trino
- Vertica
- Yandex
- Zendesk

Airflow Architecture



Source: https://www.qubole.com/tech-blog/understand-apache-airflows-modular-architecture/

Airflow Components



- DAG (directed acyclic graphic) is a collection of tasks in pipeline
- Task is the basic unit of execution, an instantiated operator object with unique task id, and can be arranged into DAG to define dependencies among tasks
- Operator is a predefined task template being used in many situations e.g. BashOperator, PythonOperator, EmailOperator, Sensors, TaskFlow, and others (database operators, cloud storage operators, etc.)
- Others: hooks, coms, trigger, pool, variables, etc.

Airflow Common Operators

- BashOperator - executes a bash command
- PythonOperator - calls an arbitrary Python function
- EmailOperator - sends an email
- SimpleHttpOperator - sends an HTTP request
- MySqlOperator, SqliteOperator, PostgresOperator, MsSqlOperator, OracleOperator, JdbcOperator, etc. - executes a SQL command
- Sensor - an Operator that waits (polls) for a certain time, file, database row, S3 key, etc…
- Other specific operators: DockerOperator, HiveOperator, S3FileTransformOperator, PrestoToMySqlTransfer, SlackAPIOperator, etc.

How to run airflow

- Two recommended approaches, standalone and docker

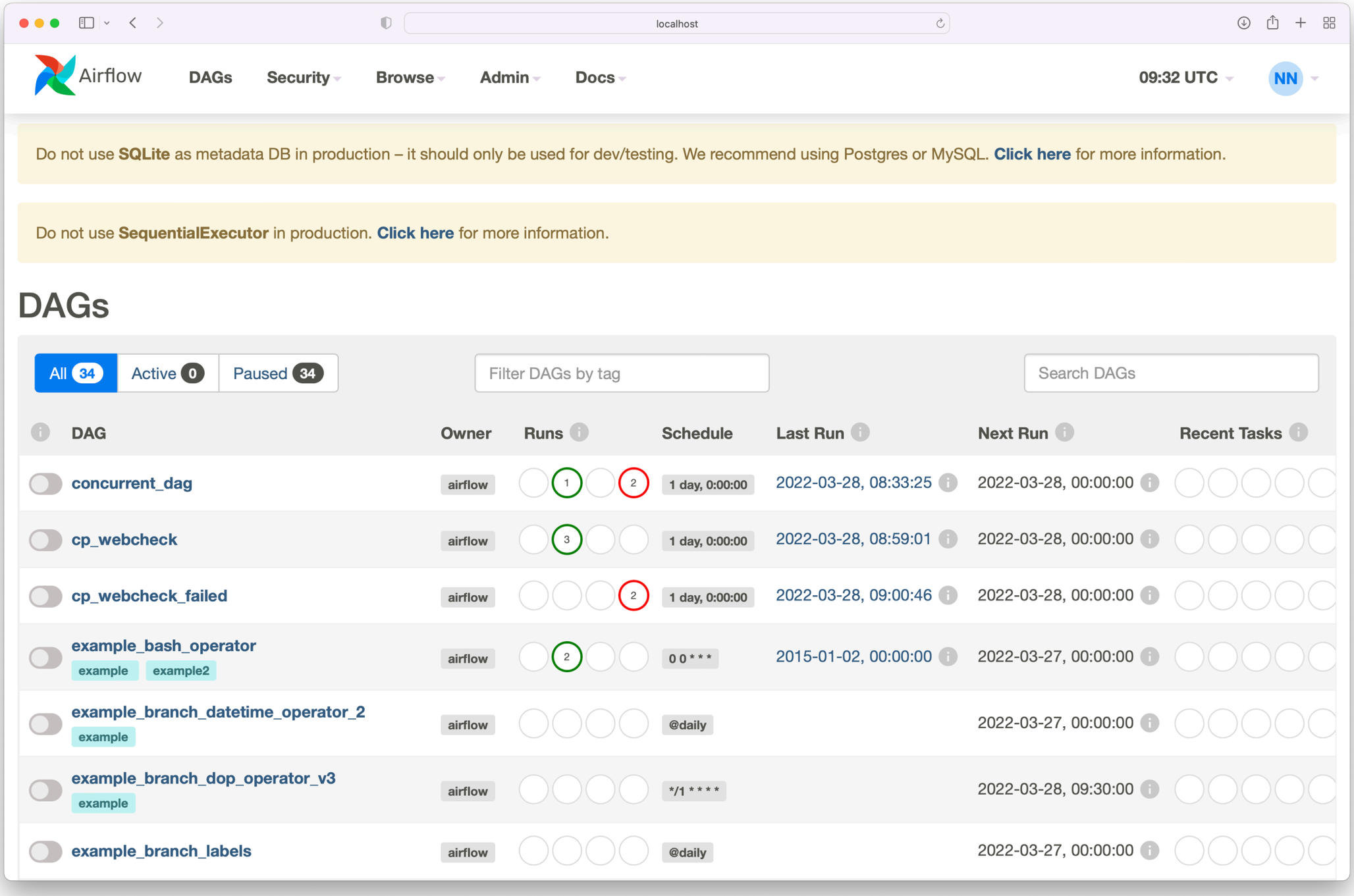
```
pip install apache-airflow

airflow db init

airflow users create --username admin --firstname Natawut --
lastname Nupairoj --role Admin --email natawut.n@chula.ac.th

airflow standalone
```

Configure airflow

- Default airflow config/database directory is at ~/airflow
- You can add new dags in ~/airflow/dags
- You should modify ~/airflow/airflow.cfg to make airflow checking for new dags more frequent

```
dag_dir_list_interval = 300
```

Airflow UI


localhost
Airflow
DAGs
Security
Browse
Admin
Docs
09:32 UTC
NN
Do not use SQLite as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. Click here for more information.
Do not use SequentialExecutor in production. Click here for more information.
DAGs
All 34
Active 0
Paused 34
Filter DAGs by tag
Search DAGs
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
concurrent_dag
airflow
1
2
1 day, 0:00:00
2022-03-28, 08:33:25
2022-03-28, 00:00:00
cp_webcheck
airflow
3
1 day, 0:00:00
2022-03-28, 08:59:01
2022-03-28, 00:00:00
cp_webcheck_failed
airflow
2
1 day, 0:00:00
2022-03-28, 09:00:46
2022-03-28, 00:00:00
example_bash_operator
example
example2
airflow
2
0 0 * * *
2015-01-02, 00:00:00
2022-03-27, 00:00:00
example_branch_datetime_operator_2
example
airflow
@daily
2022-03-27, 00:00:00
example_branch_dop_operator_v3
example
airflow
*/1 * * * *
2022-03-28, 09:30:00
example_branch_labels
airflow
@daily
2022-03-27, 00:00:00

BashOperator

- BashOperator allows Airflow to run a task using bash script
- The exit code of the program/script will determine the result of the task execution
  - 0 - Success
  - 99 - Skipped
  - Something else - Failed
- Parameterizing bash_command is possible using Jinja templates (refer to https://airflow.apache.org/docs/apache-airflow/stable/templates-ref.html for more details
- Airflow captures the last line of the output from the script in ‘output’

Example: Simple BashOperator

simple_bashoper.py

```
from airflow.utils.dates import days_ago
from airflow import DAG

from airflow.operators.bash import BashOperator

dag = DAG('dsde_simplebash', start_date=days_ago(1))

echo = BashOperator(task_id='echo_template', bash_command='echo "run_id =
{{ run_id }} and ds = {{ ds }}"', dag=dag)
```

echo

Example: Check if website runs

cp_webcheck.py

```
from airflow.utils.dates import days_ago
from airflow import DAG

from airflow.operators.bash import BashOperator

dag = DAG('dsde_webcheck', start_date=days_ago(1))

ping = BashOperator(task_id='http_check', bash_command='curl -s https://
www.cp.eng.chula.ac.th > /dev/null', dag=dag)

inform = BashOperator(task_id='inform_status', bash_command='echo "CP
website still works!"', dag=dag)

ping >> inform
```

Example: Concurrent Tasks

concur_webcheck.py

```
from airflow.utils.dates import days_ago
from airflow import DAG

from airflow.operators.bash import BashOperator
from airflow.operators.empty import EmptyOperator

# 'with' enables DAG to become context managers; automatically assign new operators to that DAG
with DAG('dsde_concurrent', start_date=days_ago(1)) as dag:
    start = EmptyOperator(task_id='start_task')
    ping = BashOperator(task_id='cp_check', bash_command='curl https://www.cp.eng.chula.ac.th')
    ping2 = BashOperator(task_id='eng_check', bash_command='curl https://www.eng.chula.ac.th')
    inform = BashOperator(task_id='inform_status', bash_command='echo "CP website still works!"')

    # creating DAG dependencies can be a long flow or multiple short flows
    # start >> [ping, ping2] >> inform
    start >> [ping, ping2]
    ping >> inform
    ping2 >> inform
```

PythonOperator / PythonVirtualenvOperator

- PythonOperator provides more flexibility of running python code e.g. perform data conversion, train a model

- Input parameters can be passed via op_kwargs/op_args and XCom

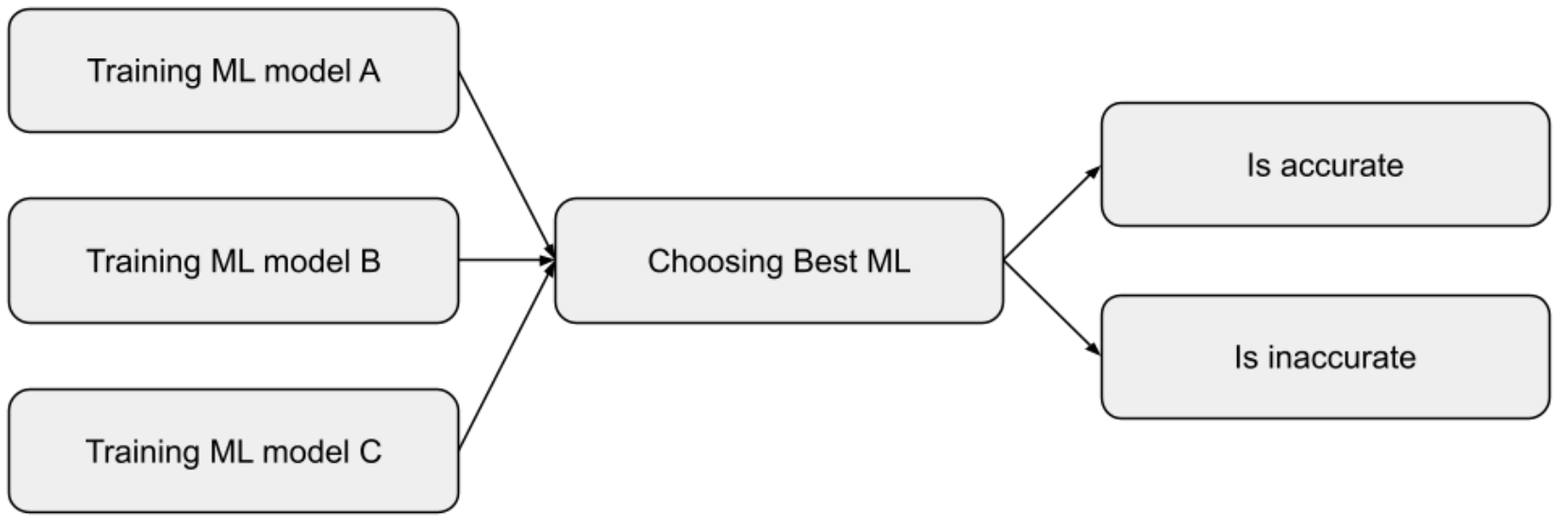
- Return values are captured in ‘output’ variable

Example: PythonOperator

python_operator.py

```
from airflow.utils.dates import days_ago
from airflow import DAG

from airflow.operators.bash import BashOperator
from airflow.operators.empty import EmptyOperator
from airflow.operators.python import PythonOperator


def show_status(web1, web2):
    print('All websites are running!')
    print('web1 = ', web1)
    print('web2 = ', web2)
    print('--- DONE ---')


# 'with' enables DAG to become context managers; automatically assign new operators to that DAG
with DAG('dsde_python', start_date=days_ago(1)) as dag:
    start = EmptyOperator(task_id='start_task')
    ping = BashOperator(task_id='cp_check', bash_command='curl https://www.cp.eng.chula.ac.th')
    ping2 = BashOperator(task_id='eng_check', bash_command='curl https://www.eng.chula.ac.th')
    inform = PythonOperator(task_id='inform_status', python_callable=show_status,
op_args=[ping.output, ping2.output])

    # creating DAG dependencies can be a long flow or multiple short flows
    # start >> [ping, ping2] >> inform
    start >> [ping, ping2]
    ping >> inform
    ping2 >> inform
```

BranchPythonOperator



- Allows to follow a specific path according to a condition, evaluated in a python callable function
- Executes a Python function returning a list of task ids of the next tasks to execute

Example: PythonBranchOperator

branch_operator.py

```python
from airflow import DAG
from airflow.operators.python import BranchPythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime


def _choose_best_model(accuracy):
    if accuracy > 0.8:
        return 'accurate'
    return 'inaccurate'


with DAG('dsde_branch', start_date=datetime(2021, 1, 1), catchup=False) as dag:
    choose_best_model = BranchPythonOperator(task_id='choose_best_model',
python_callable=_choose_best_model, op_args=[0.75])

    accurate = EmptyOperator(task_id='accurate')
    inaccurate = EmptyOperator(task_id='inaccurate')

    choose_best_model >> [accurate, inaccurate]
```

Example: PythonBranchOperator

complex_branch.py

```python
"""Example DAG demonstrating the usage of the BranchPythonOperator."""

import random
from datetime import datetime

from airflow import DAG
from airflow.operators.empty import EmptyOperator
from airflow.operators.python import BranchPythonOperator
from airflow.utils.edgemodifier import Label
from airflow.utils.trigger_rule import TriggerRule

with DAG(
    dag_id='example_branch_operator',
    start_date=datetime(2021, 1, 1),
    catchup=False,
    schedule_interval="@daily",
    tags=['example', 'example2'],
) as dag:
    run_this_first = EmptyOperator(
        task_id='run_this_first',
    )

    options = ['branch_a', 'branch_b', 'branch_c', 'branch_d']

    branching = BranchPythonOperator(
        task_id='branching',
        python_callable=lambda: random.choice(options),
    )
    run_this_first >> branching

    join = EmptyOperator(
        task_id='join',
        trigger_rule=TriggerRule.NONE_FAILED_MIN_ONE_SUCCESS,
    )

    for option in options:
        t = EmptyOperator(
            task_id=option,
        )

        dummy_follow = EmptyOperator(
            task_id='follow_' + option,
        )

        # Label is optional here, but it can help identify more complex branches
        branching >> Label(option) >> t >> dummy_follow >> join
```

Sensors

- a special type of Operator that are designed to wait until something happens, and then succeed so their downstream tasks can run

- Can be time-based, or waiting for a file, or an external event

- Sensors have three different modes of running

  - poke (default): The Sensor takes up a worker slot for its entire runtime

  - reschedule: The Sensor takes up a worker slot only when it is checking, and sleeps for a set duration between checks

  - smart sensor: There is a single centralized version of this Sensor that batches all executions of it

More about DAG

- DAG can be declared with the following parameters
  - start_date - when to start, can be exact date or relative e.g. days_ago(1)
  - schedule_interval - how often should this DAG run
  - catchup - should scheduler kick off a DAG Run for any data interval that has not been run since the last data interval
  - default_args - args to get passed on to each operator in this DAG, can override them on a per-task basis
- Airflow 2.0 introduces TaskFlow API with DAG and Task decorators

Other Components

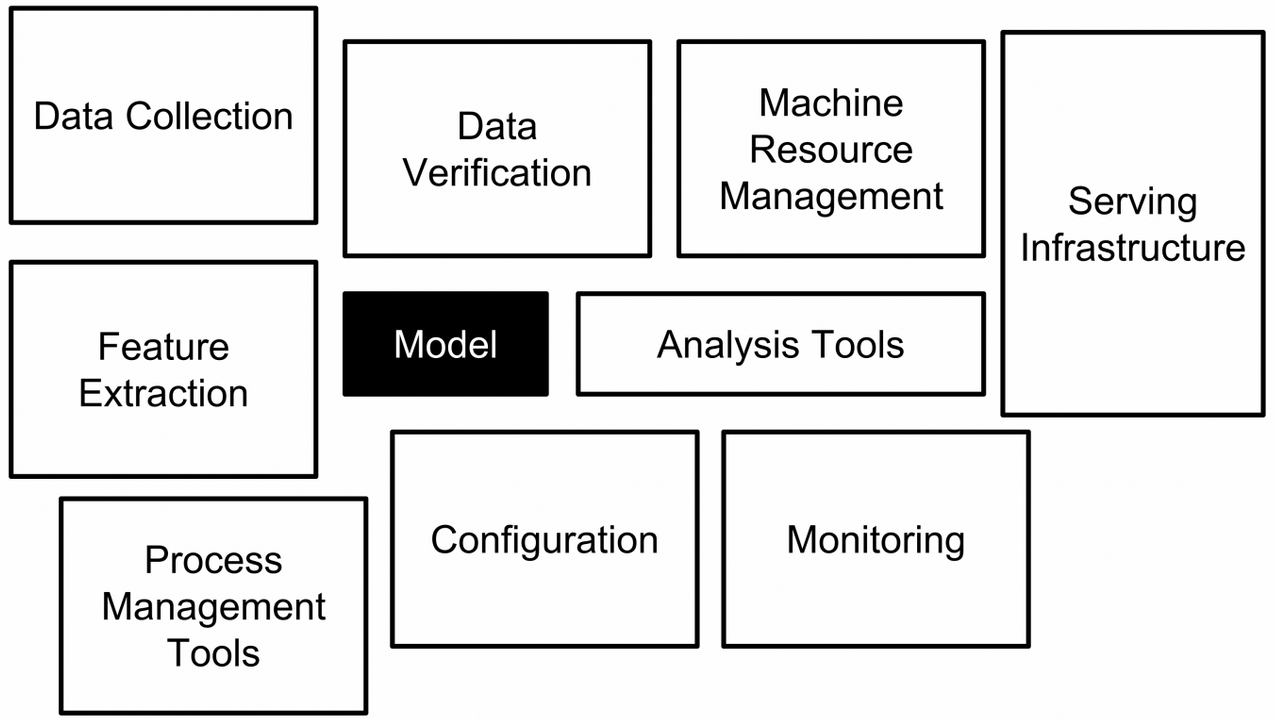
- Connectors - repository of configurations to connect to external systems
- Hooks - high-level interface (driver) to an external platform
- Variables - generic way to store and retrieve configured values (key-value pair), similar to environment variables
- XCom - operator cross-communication - let tasks exchange information
- Executors - mechanism how task instances get to run (default = LocalExecutors)
- Trigger - run DAG from CLI, REST API, Web UI, TriggerDagRunOperator

Use Cases

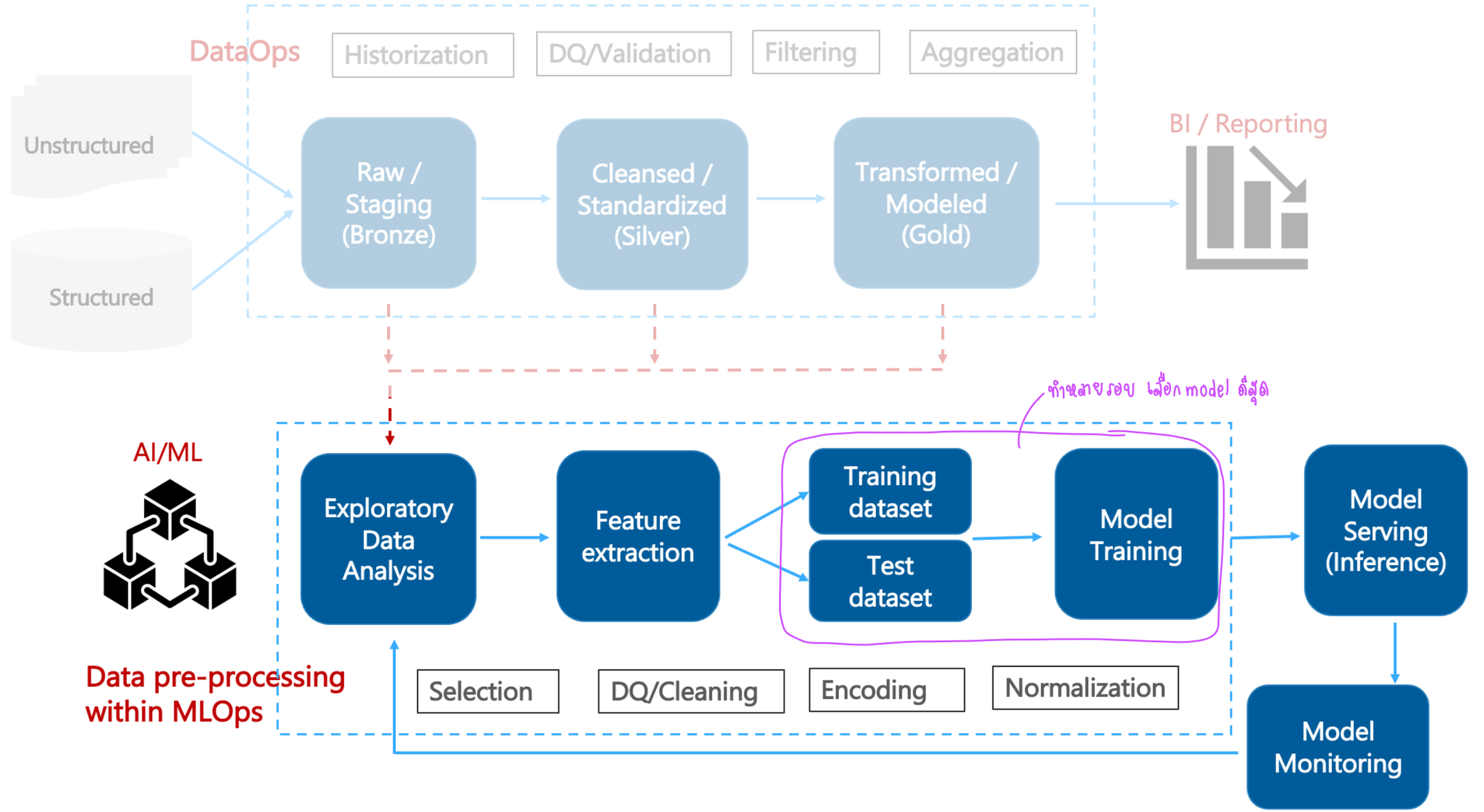
- ETL Pipelines that extract data from multiple sources and run Spark jobs or any other data transformations
- Collecting Sensor data and move to Data Lake/Data warehouse
- Training machine learning models
- Orchestrating automated testing
- Report generation

Orchestrating ML Pipeline

Ops Stars



DataOps and MLOps

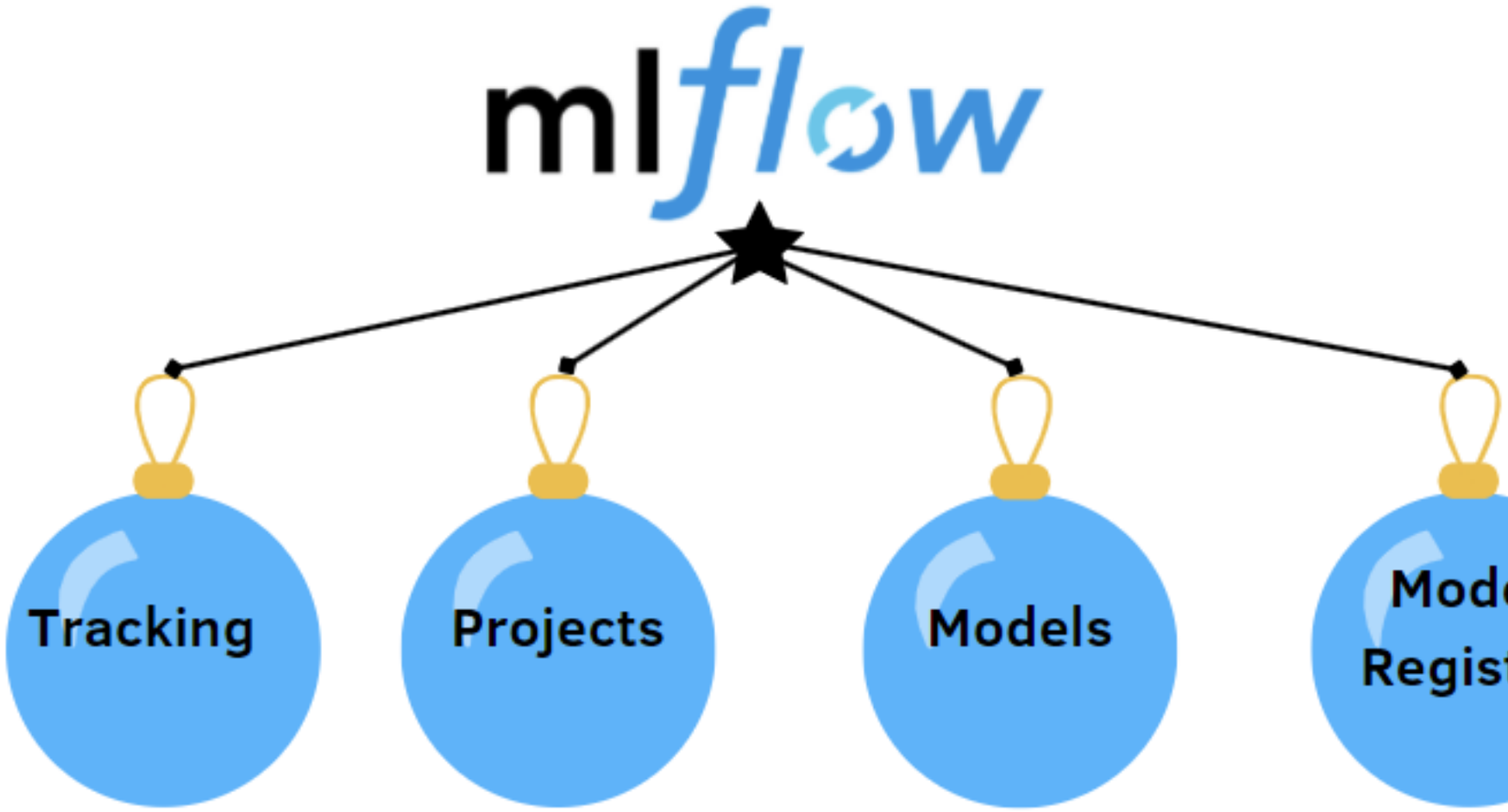


Source: https://towardsdatascience.com/bridging-dataops-mlops-301f010caf30

MLOps Orchestration Requirements

- In additional to workflow control, orchestrating MLOps requires some unique requirements
  - Code + Data = combination of development and data pipeline
  - Model version control
  - Model performance tracking during training (and serving)

MLflow Components



Track and record experiments

Package code in a reusable and reproducible way

Persist ML model for deployment in several environments

Manage models and their lifecycle centrally

Key Concepts in MLflow

- A run is a collection of parameters, metrics, labels, and artifacts related to the training process of a machine learning model

- An experiment is the basic unit of MLflow organization ; all MLflow runs belong to an experiment

- For each experiment, you can analyze and compare the results of different runs, and easily retrieve metadata artifacts for analysis using downstream tools

How to run MLflow

- Install the MLflow

```
pip install mlflow
```

- Install the tutorial code

```
git clone https://github.com/mlflow/mlflow
```

mlflow/examples/sklearn_elasticnet_wine/train.py

Harutaka Kawamura, 2 years ago | 4 authors (Matei Zaharia and others)

```
# The data set used in this example is from http://archive.ics.uci.edu/ml/datasets/Wine+Quality      Matei Zaharia, 4 years
# P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
# Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547

import os
import warnings
import sys

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
from urllib.parse import urlparse
import mlflow
import mlflow.sklearn

import logging

logging.basicConfig(level=logging.WARN)
logger = logging.getLogger(__name__)


def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    r2 = r2_score(actual, pred)
    return rmse, mae, r2


if __name__ == "__main__":
```

mlflow/examples/sklearn_elasticnet_wine/train.py

```
58  with mlflow.start_run():
59      lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
60      lr.fit(train_x, train_y)
61
62      predicted_qualities = lr.predict(test_x)
63
64      (rmse, mae, r2) = eval_metrics(test_y, predicted_qualities)
65
66      print("Elasticnet model (alpha=%f, l1_ratio=%f):" % (alpha, l1_ratio))
67      print("  RMSE: %s" % rmse)
68      print("  MAE: %s" % mae)
69      print("  R2: %s" % r2)
70
71      mlflow.log_param("alpha", alpha)
72      mlflow.log_param("l1_ratio", l1_ratio)
73      mlflow.log_metric("rmse", rmse)
74      mlflow.log_metric("r2", r2)
75      mlflow.log_metric("mae", mae)
76
77      tracking_url_type_store = urlparse(mlflow.get_tracking_uri()).scheme
78
79      # Model registry does not work with file store
80      if tracking_url_type_store != "file":
81
82          # Register the model
83          # There are other ways to use the Model Registry, which depends on the use case,
84          # please refer to the doc for more information:
85          # https://mlflow.org/docs/latest/model-registry.html#api-workflow
86          mlflow.sklearn.log_model(lr, "model", registered_model_name="ElasticnetWineModel")
87      else:
88          mlflow.sklearn.log_model(lr, "model")
```

Running Model and View Results

- At mlflow/examples

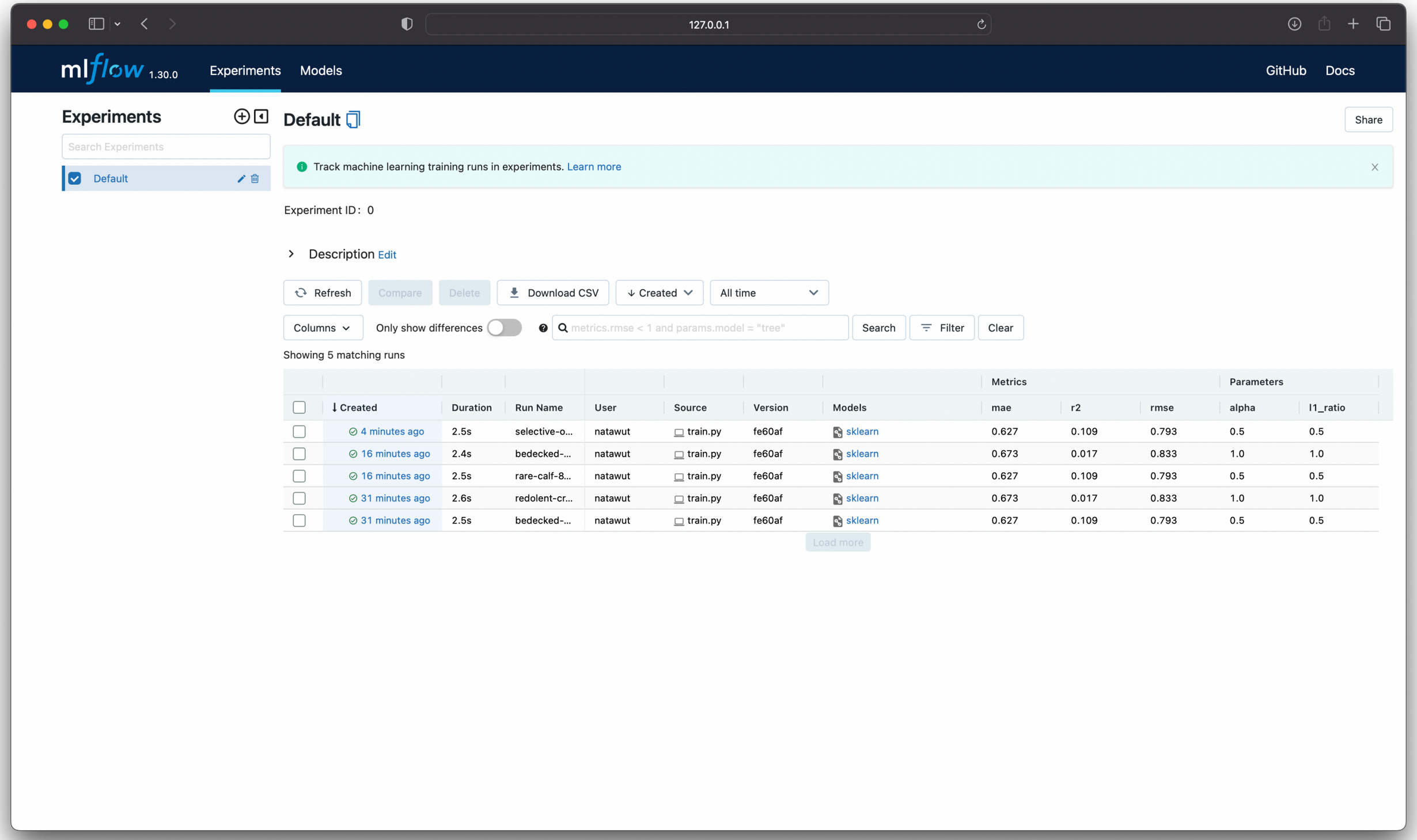
```
python sklearn_elasticnet_wine/train.py
python sklearn_elasticnet_wine/train.py [alpha] [l1]
```

- View results (run at mlflow/examples) and use your browser to http://localhost:5000

```
mlflow ui
```

- You can change to other port with -p

```
mlflow ui -p 8000
```


127.0.0.1
mlflow 1.30.0
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
Default
Share
Track machine learning training runs in experiments. Learn more
Experiment ID: 0
Description Edit
Refresh
Compare
Delete
Download CSV
Created
All time
Columns
Only show differences
metrics.rmse < 1 and params.model = "tree"
Search
Filter
Clear
Showing 5 matching runs
Metrics
Parameters
Created
Duration
Run Name
User
Source
Version
Models
mae
r2
rmse
alpha
l1_ratio
4 minutes ago 2.5s selective-o... natawut train.py fe60af sklearn 0.627 0.109 0.793 0.5 0.5
16 minutes ago 2.4s bedecked-... natawut train.py fe60af sklearn 0.673 0.017 0.833 1.0 1.0
16 minutes ago 2.5s rare-calf-8... natawut train.py fe60af sklearn 0.627 0.109 0.793 0.5 0.5
31 minutes ago 2.6s redolent-cr... natawut train.py fe60af sklearn 0.673 0.017 0.833 1.0 1.0
31 minutes ago 2.5s bedecked-... natawut train.py fe60af sklearn 0.627 0.109 0.793 0.5 0.5
Load more

Packaging and Serving Model

- Model can be packaged for running with its own python environment (for other to test or run on cloud)

```
mlflow run sklearn_elasticnet_wine -P alpha=0.42
```

- Model can be deployed as a standalone service by selecting the run that we want to use

```
mlflow models serve -m ./mlruns/0/12334a9519a94de99a3363341b69c97c/
artifacts/model -p 1234
```

```
curl -X POST -H "Content-Type:application/json; format=pandas-split" --
data '{"columns":["alcohol", "chlorides", "citric acid", "density",
"fixed acidity", "free sulfur dioxide", "pH", "residual sugar",
"sulphates", "total sulfur dioxide", "volatile acidity"],"data":[[12.8,
0.029, 0.48, 0.98, 6.2, 29, 3.33, 1.2, 0.39, 75, 0.66]]}' http://
127.0.0.1:1234/invocations
```

127.0.0.1

mlflow 1.30.0 Experiments Models GitHub Docs

Default › selective-owl-216

selective-owl-216

Run ID: 12334a9519a94de99a3363341b69c97c

Date: 2022-10-31 20:48:19

Source: train.py

Git Commit: fe60afd98c8a783ed3ced24150f2784f980ce790

User: natawut

Duration: 2.5s

Status: FINISHED

Lifecycle Stage: active

› Description Edit

› Parameters (2)

› Metrics (3)

› Tags

⌄ Artifacts

- model
  - MLmodel
  - conda.yaml
  - model.pkl
  - python_env.yaml
  - requirements.txt

Full Path:./mlruns/0/12334a9519a94de99a3363341b69c97c/artifacts/model

Register Model

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control

Model schema

Input and output schema for your model. Learn more

Name	Type

No schema. See MLflow docs for how to include input and output schema with your model.

Make Predictions

Predict on a Spark DataFrame:

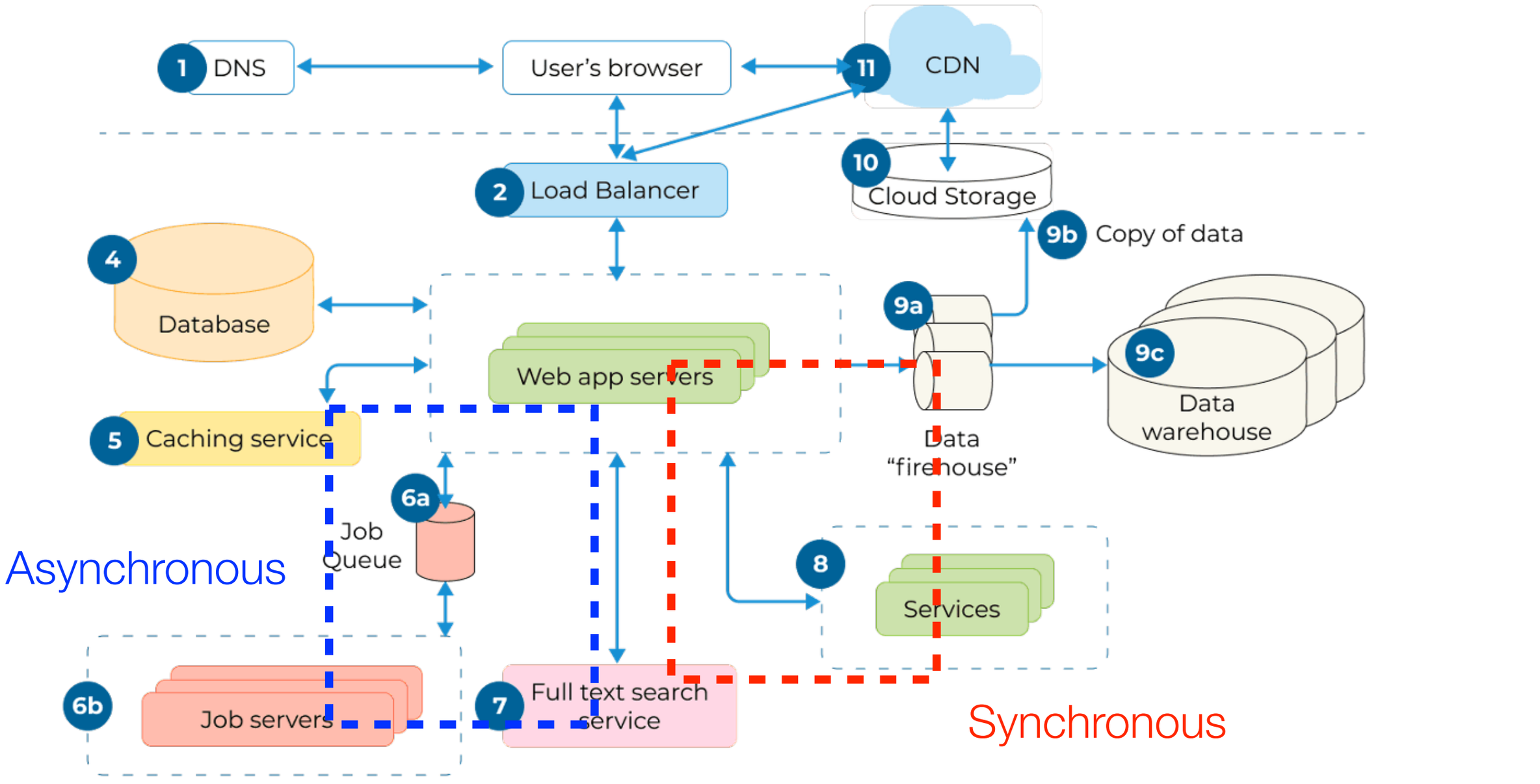
```
import mlflow
from pyspark.sql.functionsimport struct, col
logged_model = 'runs:/12334a9519a94de99a3363341b69c97c/model'

# Load model as a Spark UDF. Override result_type if the model does not return double values.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model, result_type='double')

# Predict on a Spark DataFrame.
df.withColumn('predictions', loaded_model(struct(*map(col, df.columns))))
```

Predict on a Pandas DataFrame:

Provisioning ML Model

Ops Stars

Real-Life Web Application Architecture

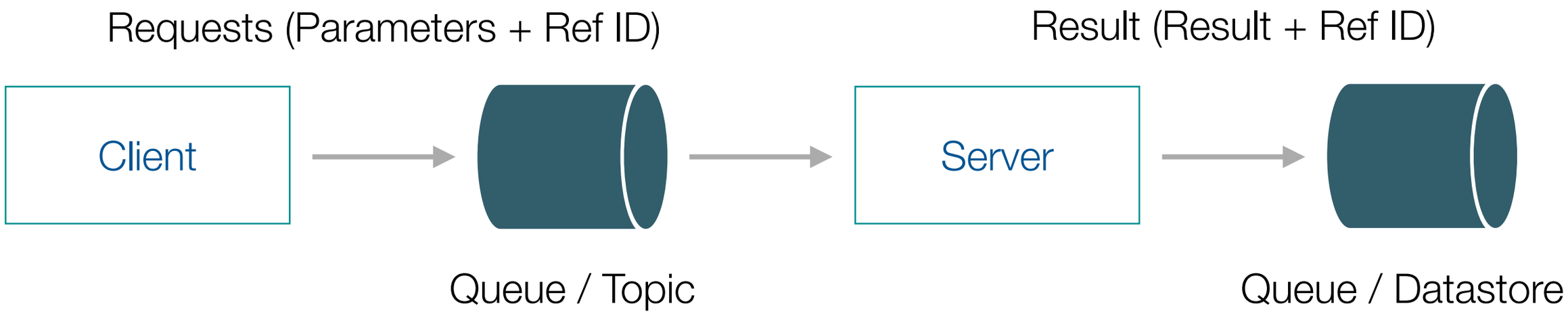


Source: https://litslink.com/blog/web-application-architecture

3 Modes of ML Model Integration

- Batch (e.g. using Airflow) ยาจจะรู้จนวันสะคร้ง
  - Focus on throughput
  - Highly effective resource utilization
- Asynchronous
  - Use queue to control inferencing execution
  - Balance between resource utilization and response time
- Synchronous
  - Provide API to response in real-time
  - High-resource consumptions

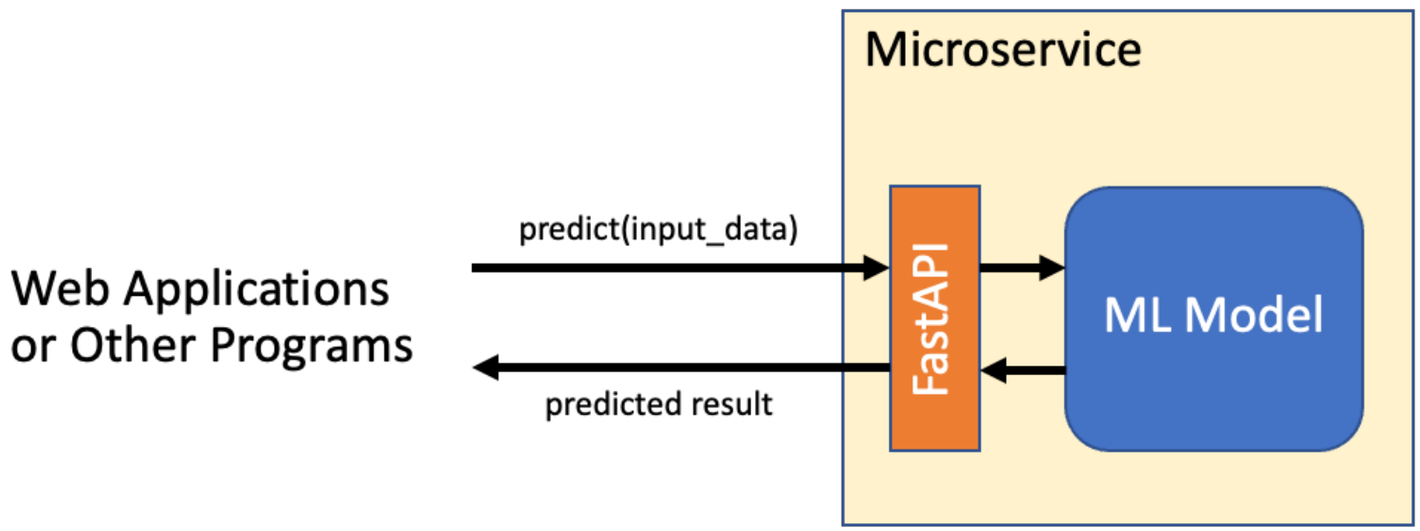
Provisioning ML Model : Asynchronous



- Using MOM (message oriented middleware) e.g. Rabbit MQ, Kafka
- Clients send requests (with parameters and reference ID) into inference queue (or topic)
- Inference server grabs a request from the queue (one at a time), performs inferencing, stores the result in datastore or sends it to another service via another queue

Provisioning ML Model : Synchronous

- Encapsulating your model as a microservice is a popular approach to deploy machine learning model
- The model becomes a black box and can be invoked using REST protocol
- Using a microframework such as FastAPI can simplify microservice development



Microframework

- Minimalistic web application frameworks designed for Microservices and API development

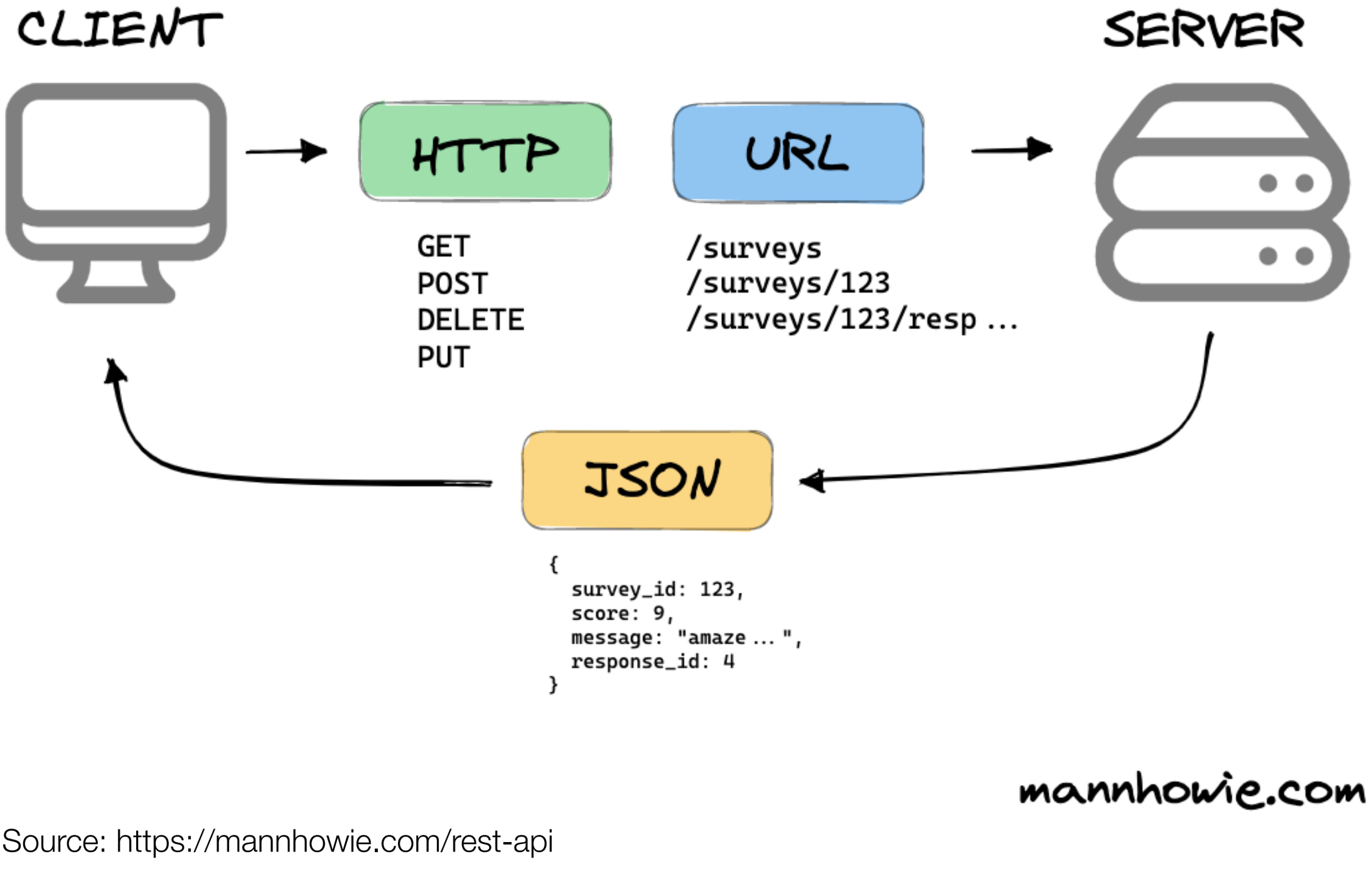
- Facilitate receiving an HTTP request, routing the HTTP request to the appropriate controller, dispatching the controller, and returning an HTTP response

- Designed for building the APIs for another service or application; therefore, lacking most of the functionality expected in full-stack framework

REST - Heart and Soul of Microservices

- REpresentational State Transfer
- Lightweight and simple
  - Only simple XML or JSON on HTTP
  - Utilize HTTP methods instead of using “envelope” style
  - URI as method identification
  - Emulate CRUD operations

WHAT IS A REST API?



Source: https://mannhowie.com/rest-api

REST over HTTP - GET Method

Request

```
GET /music/artists/beatles/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Method

Resource

Response

State transfer

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

Representation

FastAPI

- Web framework for developing RESTful APIs in Python based on Pydantic and starlette
- Provide several facilities for REST development
  - Data validation / serialization / deserialization
  - Auto-generate OpenAPI documents
  - Asynchronous programming
  - Dependency Injection
  - Support GraphQL for complex data model
  - Run with Uvicorn and Unicorn

Install Fastapi

- Very simple installation process with pip
- Make sure that you install both fastapi and unicorn

```
pip install fastapi uvicorn
```

Example: Simple Service

```python
# simple fastapi example

import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "My first FastAPI service"}

if __name__ == "__main__":
    uvicorn.run('simple:app', host="0.0.0.0", port=8000, reload=True)
```

Testing Our First Service

- Test the service with your browser at http://localhost:8000

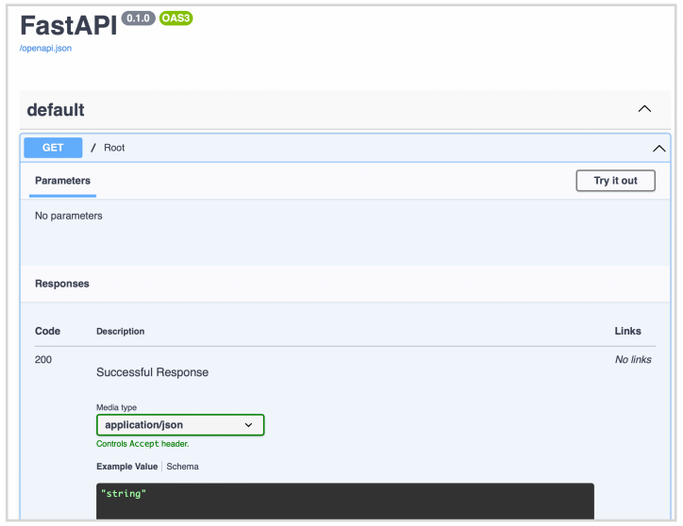
- More information at http://localhost:8000/docs

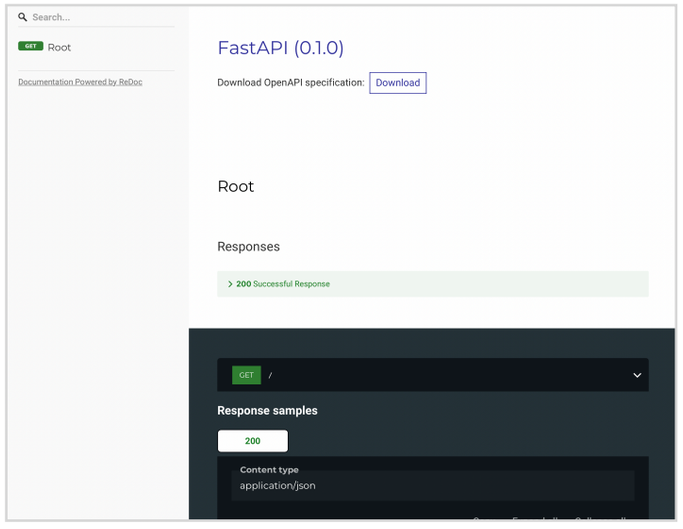
- You can also try http://localhost:8000/redoc

- Beware of browser cache

- You can also use other tools: curl, postman, etc.

{"message":"My first FastAPI service"}





Example: Path Parameter with Type Validation

```python
# path parameter example

import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get('/items/{item_id}')
async def read_item(item_id: int):
    return { 'item_id': item_id }

if __name__ == "__main__":
    uvicorn.run('path_param:app', host="0.0.0.0", port=8000, reload=True)
```

Test this service at: http://localhost:8000/items/1 and api document at: http://localhost:8000/docs

REST CRUD Operations

HTTP	CRUD	Decorator
GET	Read	`@app.get( )`
POST	Create, Update, Delete	`@app.post( )`
PUT	Create, Overwrite / Replace	`@app.put( )`
DELETE	Delete	`@app.delete( )`

Note

- FastAPI doesn’t enforce any specific meaning
- Other operations (options, head, patch, trace) are also supported

POST Operation

- POST operation allows passing complex input data in JSON via request body
- Declaring data type with pydantic, FastAPI will
  - Read the body of the request as JSON
  - Convert the corresponding types if needed
  - Validate the data and return a clear error if it is invalid
  - Put the received data in the input parameter of the function

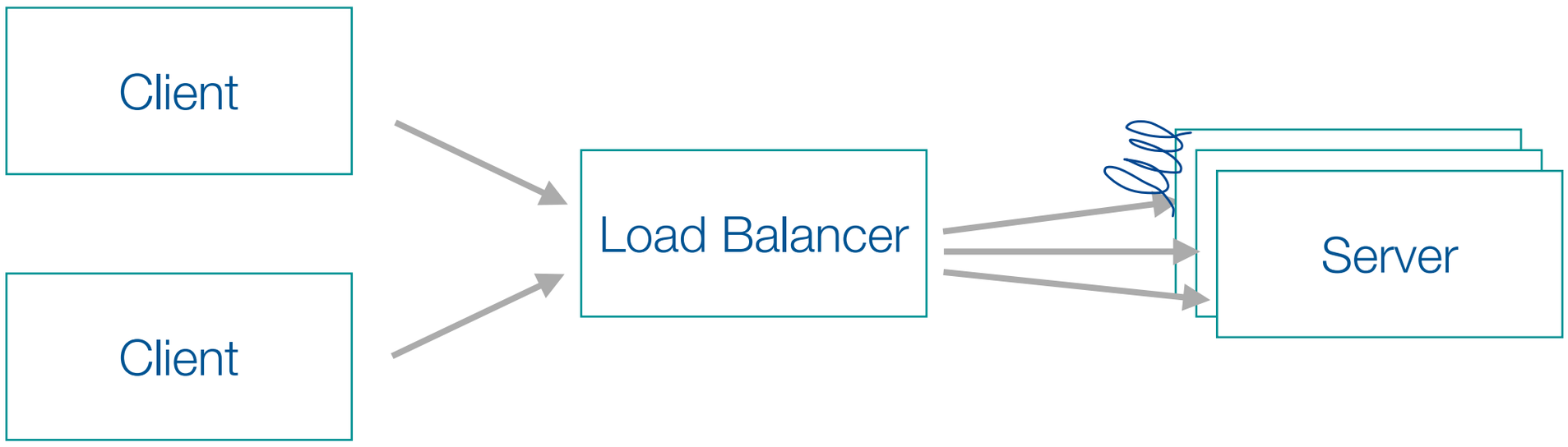
Example: POST for Complex Input Parameter

```
# post example

import uvicorn
from fastapi import FastAPI

from typing import Optional
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

app = FastAPI()

@app.post("/items/")
async def create_item(item: Item):
    return item

@app.get('/items/{item_id}')
async def read_item(item_id: int):
    return { 'item_id': item_id }

if __name__ == "__main__":
    uvicorn.run('post:app', host="0.0.0.0", port=8000, reload=True)
```

Example: Using Path Parameter and Request Body

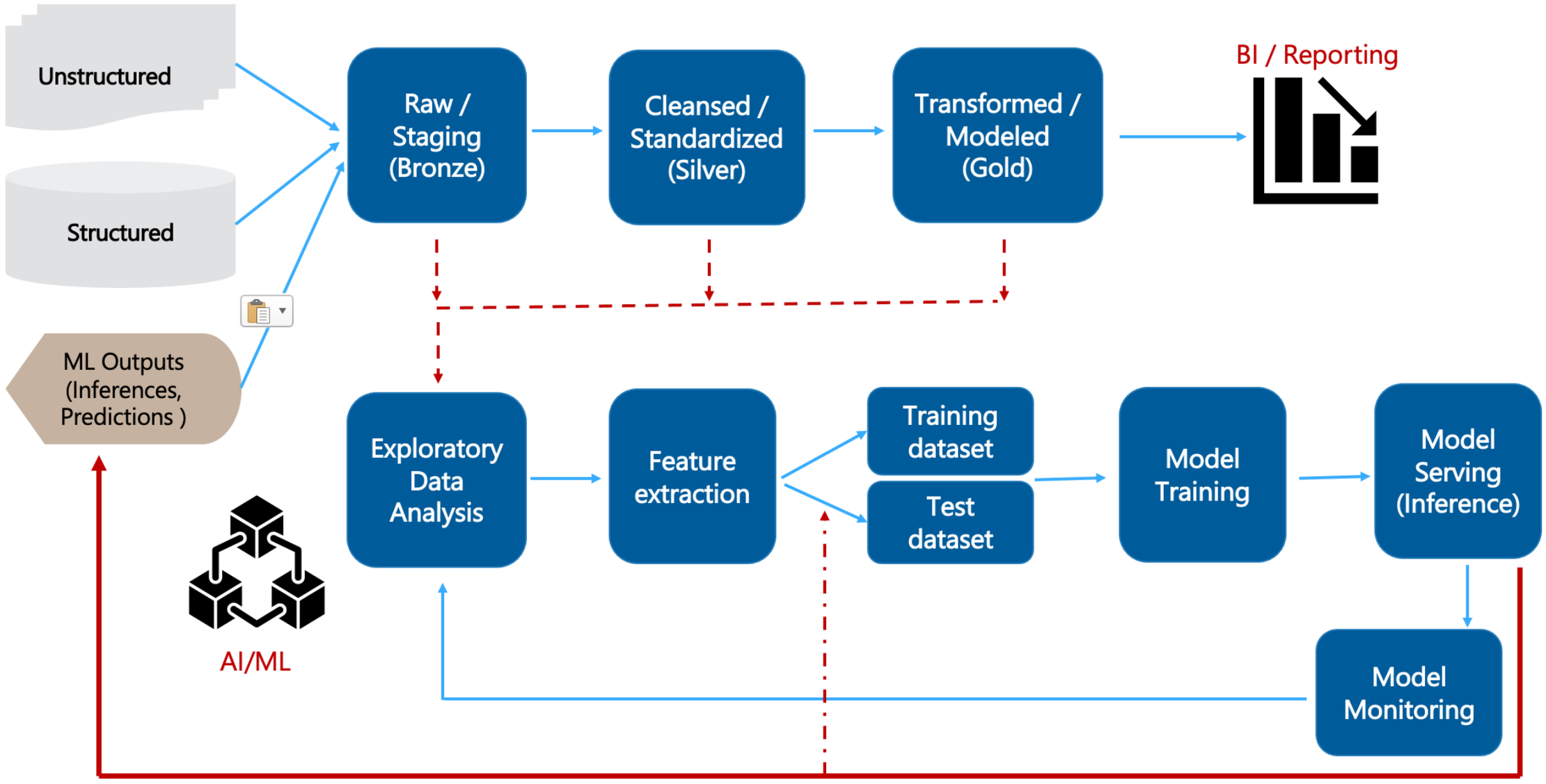
```python
# put example

import uvicorn
from fastapi import FastAPI

from typing import Optional
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

app = FastAPI()

@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item):
    return {"item_id": item_id, **item.dict()}

if __name__ == "__main__":
    uvicorn.run('put:app', host="0.0.0.0", port=8000, reload=True)
```

How to Scale Inferencing Services



- Increase number of inferencing services
  - inferencing is “stateless”, multiple copies of services can be run without problem
  - Load balancer (e.g. nginx, HAproxy) is needed
- Pre-load ML model with “lifespan” method

DataOps and MLOps Full Loop



Source: https://towardsdatascience.com/bridging-dataops-mlops-301f010caf30

Conclusion

- To embrace agile development, Data Science process becomes more complicated with DataOps, MLOps, and DevOps

- Data pipeline can be very complex as it requires task orchestration across multiple platforms

- Airflow provides sophisticated data pipeline orchestration system with configuration as code and other comprehensive facilities

- MLflow provides comprehensive framework for MLOps including performance tracking, model packaging, model serving

Conclusion

- Microframework is one of simplest approach to create a microservice

- FastAPI becomes very popular to ML service deployment as it can integrate with other python packages with ease

- FastAPI facilitates input data validation / deserialization, return data serialization, and automatic API document generation

- Combine with docker, ML microservice will be scalable and simple to deploy

References

- P.S. Tan, “Differences: DevOps, ITOps, MLOps, DataOps, ModelOps, AIOps, SecOps, DevSecOps”, https://medium.com/vitrox-publication/differences-devops-itops-mlops-dataops-modelops-aiops-secops-devsecops-part-1-3-8b238cf72942

- D. Biswas, “Bridging DataOps and MLOps: ML model inferences as a new Data Source”, https://towardsdatascience.com/bridging-dataops-mlops-301f010caf30

- run.ai, “MLflow: the Basics and a Quick Tutorial”, https://www.run.ai/guides/machine-learning-operations/mlflow

- T. Danka, “You Should Start Using FastAPI Now”, https://towardsdatascience.com/you-should-start-using-fastapi-now-7efb280fec02

- T. Danka, “How to properly ship and deploy your machine learning model”, https://towardsdatascience.com/how-to-properly-ship-and-deploy-your-machine-learning-model-8a8664b763c4