Data Engineering and MLOps in Business

Refactoring & First Serverless App

Primoz Konda

AAUBS

March 12, 2024

pk@business.aau.dk

Plan for today

Why do we need to know MLOps?

- From experimentation to production
- Collaboration
- Continues ML Life-cycle
- Monitoring and Management

The role of APIs & Data bases

Your turn now...

What is Code Refactoring?

Definition

Code refactoring is the process of restructuring existing code to improve its internal structure, without changing its external behavior.

Goals

To make the code easier to read, understand, and maintain, as well as to improve its performance, scalability, and reliability

Code Smell Example: Long Function

```python
def calculate_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        if salary < 20000:
            bonus = 0.05 * salary
        elif salary < 50000:
            bonus = 0.1 * salary
        else:
            bonus = 0.15 * salary
        total_salary += salary + bonus
    tax = 0.2 * total_salary
    net_salary = total_salary - tax
    if net_salary < 15000:
        print("Warning: Net salary is too low!")
    return net_salary
```

Problems

- The function is too long and complex
- It performs multiple tasks at once (calculating salary, tax, and net salary)
- It mixes calculation and printing

Code Smell Example: Long Function

```python
def calculate_bonus(salary):
    if salary < 20000:
        return 0.05 * salary
    elif salary < 50000:
        return 0.1 * salary
    else:
        return 0.15 * salary

def calculate_total_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        total_salary += salary + calculate_bonus(salary)
    return total_salary

def calculate_net_salary(total_salary):
    tax = 0.2 * total_salary
    net_salary = total_salary - tax
    if net_salary < 15000:
        print("Warning: Net salary is too low!")
    return net_salary
```

Code Smell Example: Long Function

```python
def calculate_bonus(salary):
    if salary < BONUS_THRESHOLD_1:
        return BONUS_RATE_1 * salary
    elif salary < BONUS_THRESHOLD_2:
        return BONUS_RATE_2 * salary
    else:
        return BONUS_RATE_3 * salary

def calculate_total_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        total_salary += salary + calculate_bonus(salary)
    return total_salary

def calculate_net_salary(total_salary):
    tax = TAX_RATE * total_salary
    net_salary = total_salary - tax
    if net_salary < SALARY_WARNING:
        print("Warning: Net salary is too low!")
    return net_salary
```

Outside function:

```
BONUS_THRESHOLD_1 = 20000
BONUS_THRESHOLD_2 = 50000
BONUS_RATE_1 = 0.05
BONUS_RATE_2 = 0.1
BONUS_RATE_3 = 0.15
TAX_RATE = 0.2
SALARY_WARNING = 15000
```

We need to talk...

Do you think your code is well-written?

Code Smell types

Common types of code smell:

- Long functions
- Duplicate code
- Dead code
- Data Clumps
- Improper names

Code Smell types: Duplicate Code

```
x1 = 1
y1 = x1 * 2
z1 = y1 + 3

x2 = 2
y2 = x2 * 2
z2 = y2 + 3

x3 = 3
y3 = x3 * 2
z3 = y3 + 3
```

```
results = []
for i in range(1, 3):
    x = i
    y = x * 2
    z = y + 3
    results.append((x, y, z))
```

Code Smell types: Dead Code

It can be a function that is never called, a variable that is never used, or a conditional branch that is never taken.

```python
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b

result = add(2, 3)
```

```python
Age = int(input("Enter the age: "))
if Age >= 0 and Age <= 2:
    print("Person is an infant")
elif Age >= 3 and Age <= 18:
    print("Person is a child")
elif Age > 18:
    print("Person is an adult")
else:
    print("Person is less than 0 years old")
```

Code Smell types: Data Clumps

Data clumps occur when several data items are always found together.

```
def calculate_distance(x1, y1, x2, y2):
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def calculate_slope(x1, y1, x2, y2):
    return (y2 - y1) / (x2 - x1)

point1_x = 2
point1_y = 3
point2_x = 5
point2_y = 7

distance = calculate_distance(point1_x, point1_y, point2_x, point2_y)
slope = calculate_slope(point1_x, point1_y, point2_x, point2_y)
```

Code Smell types: Data Clumps

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])



def calculate_distance(point1, point2):
    return ((point2.x - point1.x) ** 2 + (point2.y - point1.y) ** 2) ** 0.5

def calculate_slope(point1, point2):
    return (point2.y - point1.y) / (point2.x - point1.x)

point1 = Point(2, 3)
point2 = Point(5, 7)

distance = calculate_distance(point1, point2)
slope = calculate_slope(point1, point2)
```

Code Smell types: Improper names

Improper naming of variables, classes, and functions can make the code harder to understand and maintain.

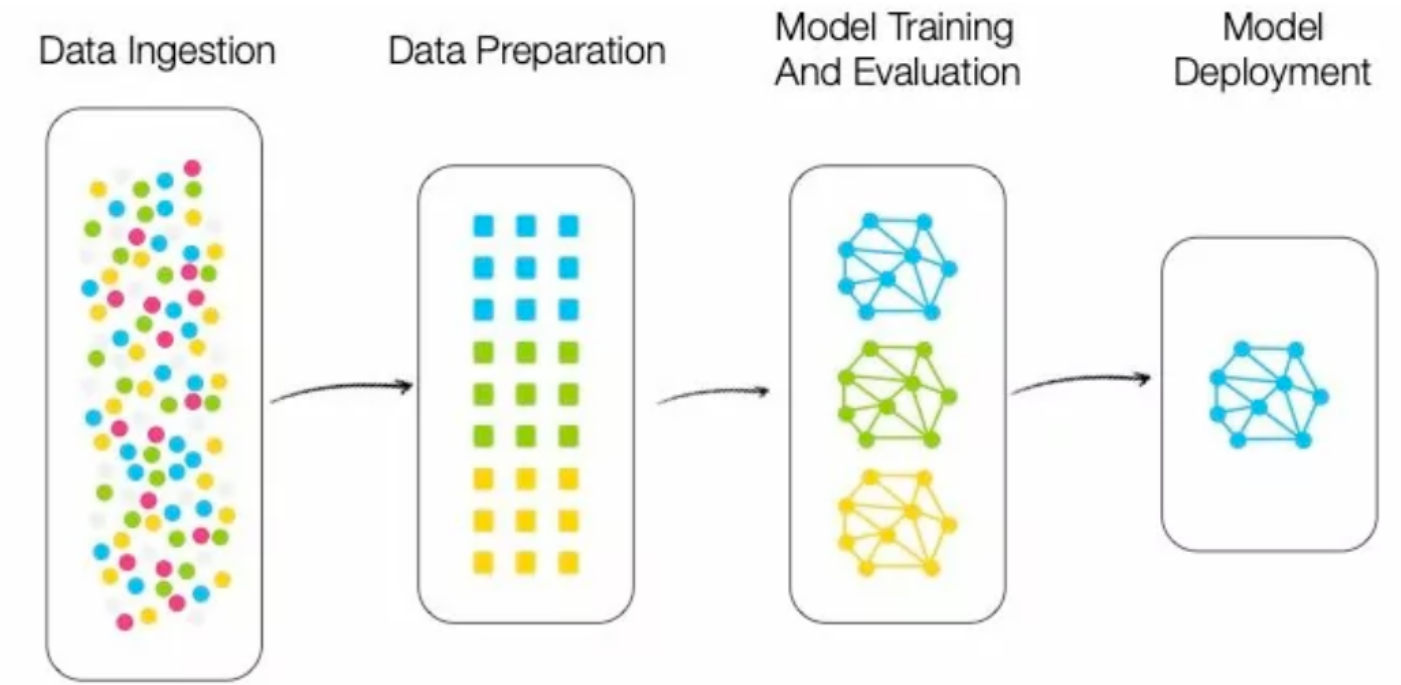
```python
def f(x):
    return x * 2

y = 5
z = f(y)
```

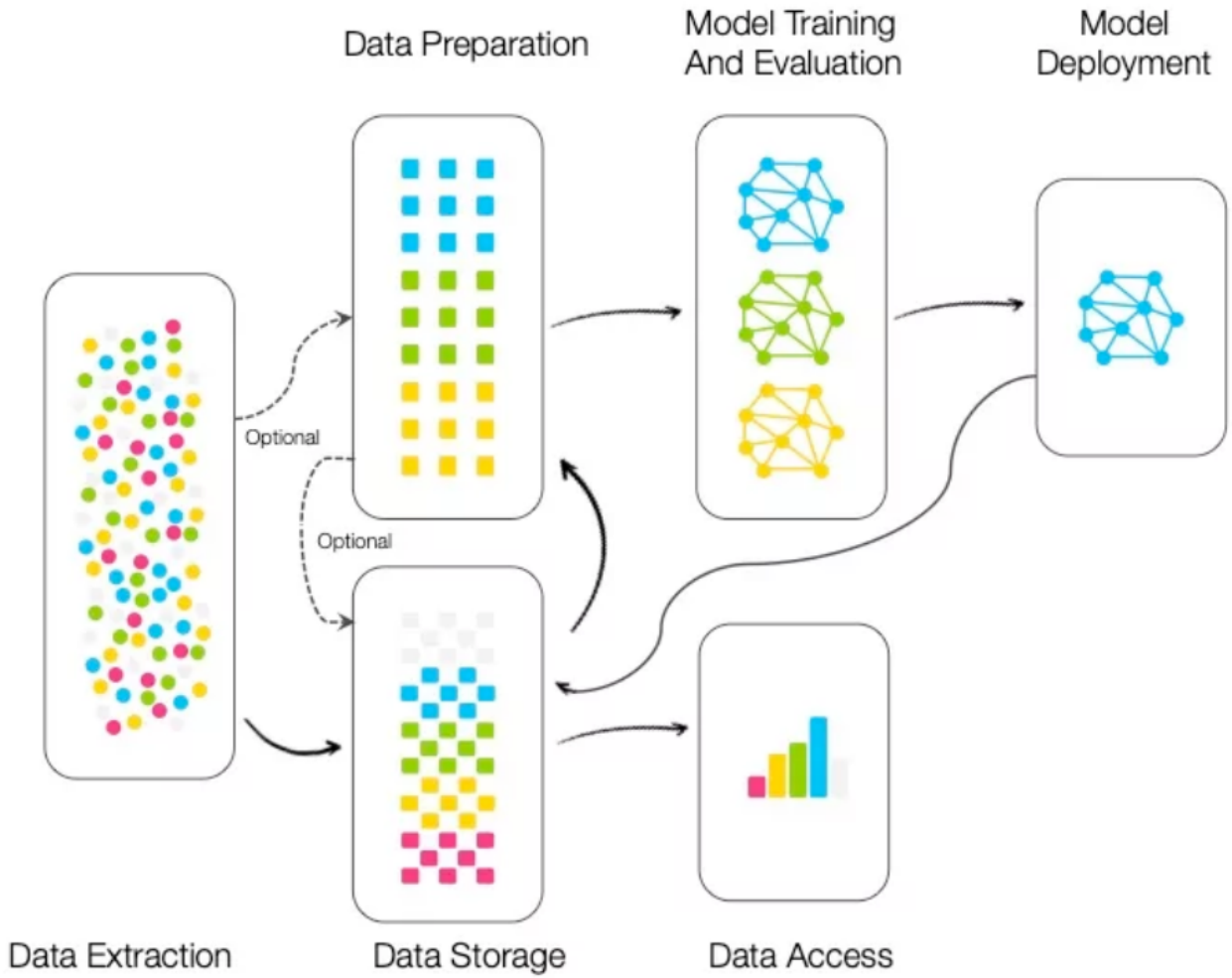
```python
def double(x):
    return x * 2

number = 5
result = double(number)
```

Basic ML Pipeline



ML & Data Pipeline



Modular Iteration in ML Pipelines

Why Refactor ML Pipelines into Separate Code Files?

- **Selective Iteration:** Enables focusing on and iterating over specific pipeline components—vital for deployed models under continuous evaluation.
- **Faster Adaptation:** Quickly update or extend parts of the pipeline based on new data or insights without overhauling the entire system.
- **Enhanced Collaboration:** Teams can work on different pipeline segments simultaneously, speeding up development and improvement cycles.
- **Continuous Improvement:** Facilitates the efficient use of new evaluation data to inform and train new models, ensuring the system evolves and improves over time.

Separate code files make ML pipelines more adaptable, maintainable, and capable of evolving in response to new information and requirements.

What is Serverless ML

Serverless ML is an approach to building and operating machine learning systems that leverages serverless computing architectures.

- cloud provider handles the operational aspects, such as infrastructure provisioning, scaling, and management
- data scientists to focus on the ML model development, training, and deployment processes
- write and deploy code without worrying about the underlying infrastructure (storage and computer resources)
- Cost Savings, Flexibility, Agility
- Challenges: Cold Starts, Vendor Lock-In, Monitoring and Debugging

Serverless ML Providers

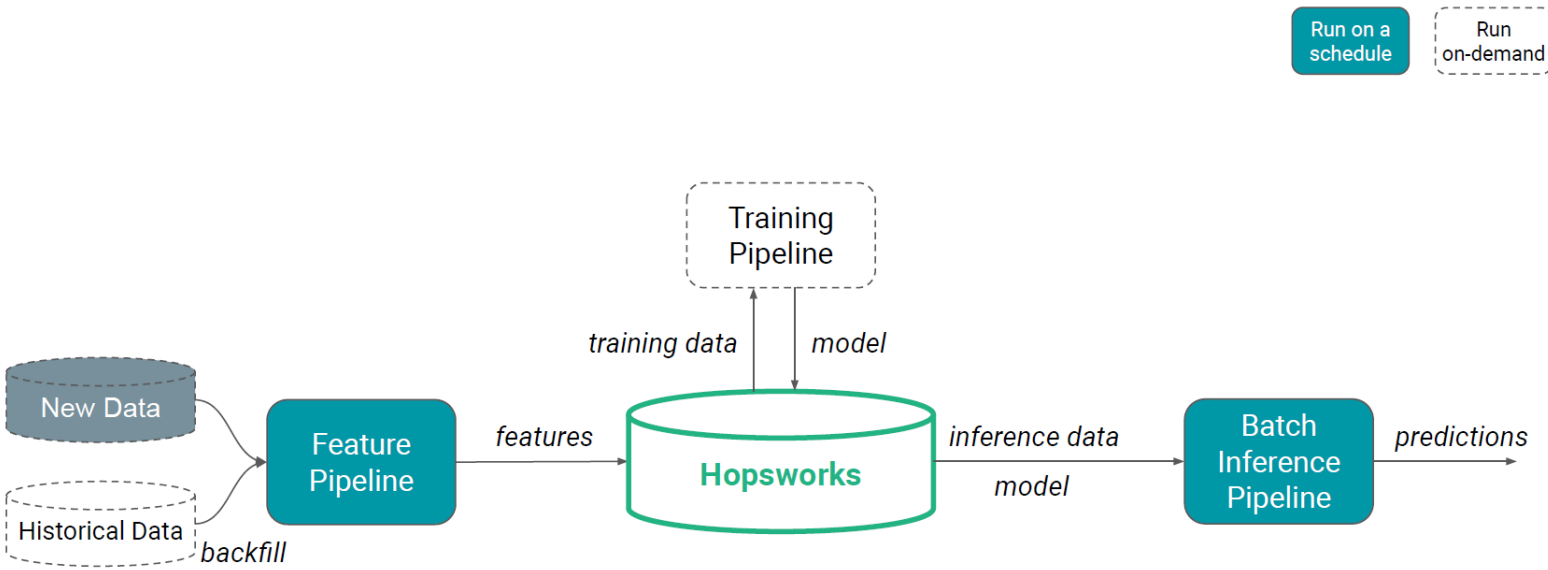
Big providers:

- Amazon Web Services (AWS) Lambda
- Microsoft Azure Functions
- Google Cloud Functions
- IBM Cloud Functions

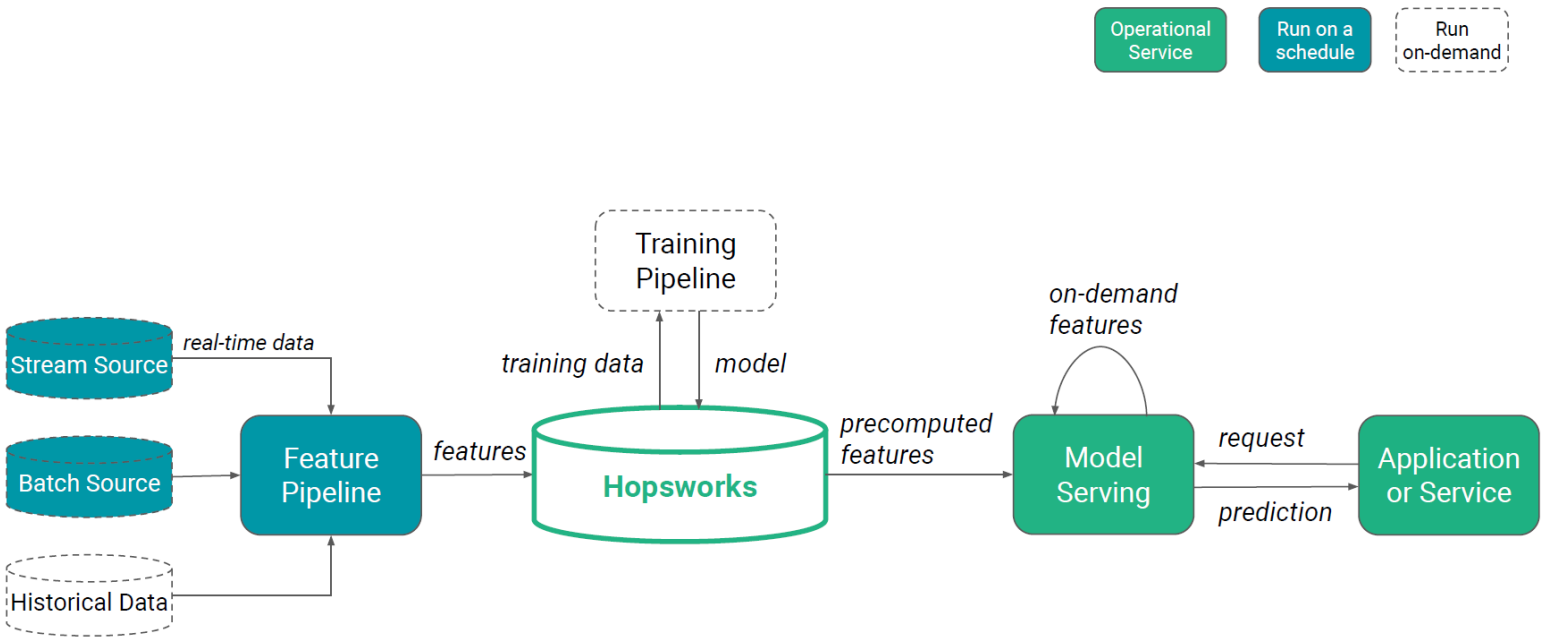
Hopsworks.ai

- Good learning support
- Free & relatively user friendly
- All functionalities

Serverlss pipelines Ex1



Serverlss pipelines Ex2



What are we trying to achieve

An online application that daily predicts on a new datapoint.

- Dataset: Iris
- Simple model
- Functions to generate new data points
- Use Github Actions and Github Pages UI

What are we trying to achieve