WP5 AI & Safe Autonomous Decision

G

WP5T52-04-01: Data version control

WP5T52-04-02: Metadata version control

Developed by: ZYLK

Component description

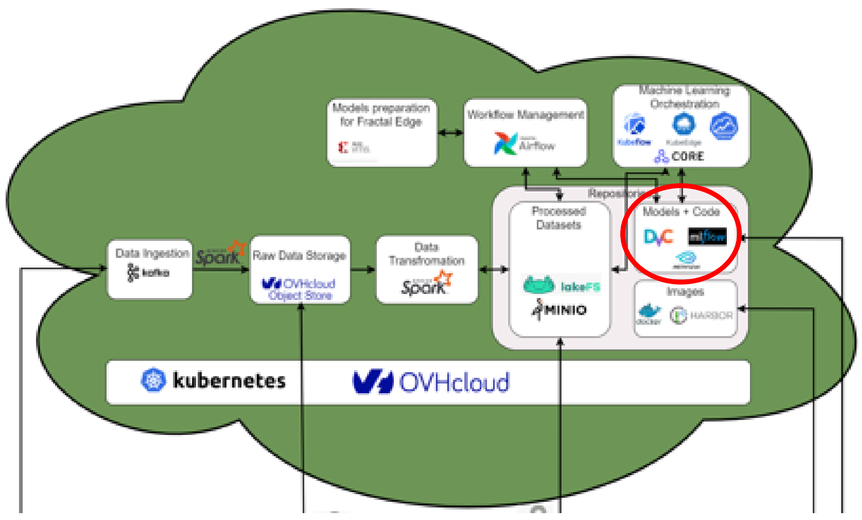
- **Objective of the component:**
  - Get a metadata and data version control for models, datasets, and large files which cannot be tracked by Git.
  - Have full control of the artifacts generated during the ML processes, data science, training, model creation and model validation.

- **Fractal Features associated:**
  - **ADAPTABILITY --> DATA ORCHESTRATION --> DATASET --> STORAGE, VERSION CONTROL**
  - **FRACTALITY --> ORCHESTRATION --> DATA**

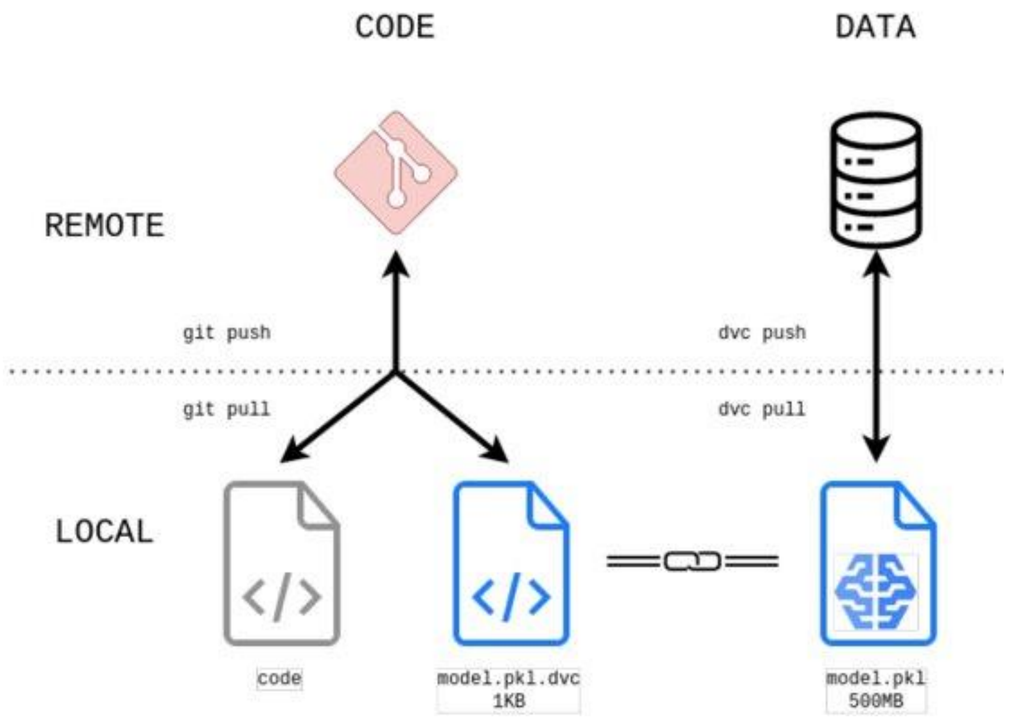
- **Inputs/Outputs:**
  - DVC: CLI commands/Data and large files being version controlled
  - Metaflow: Module usage in Python scripts / Data science projects management

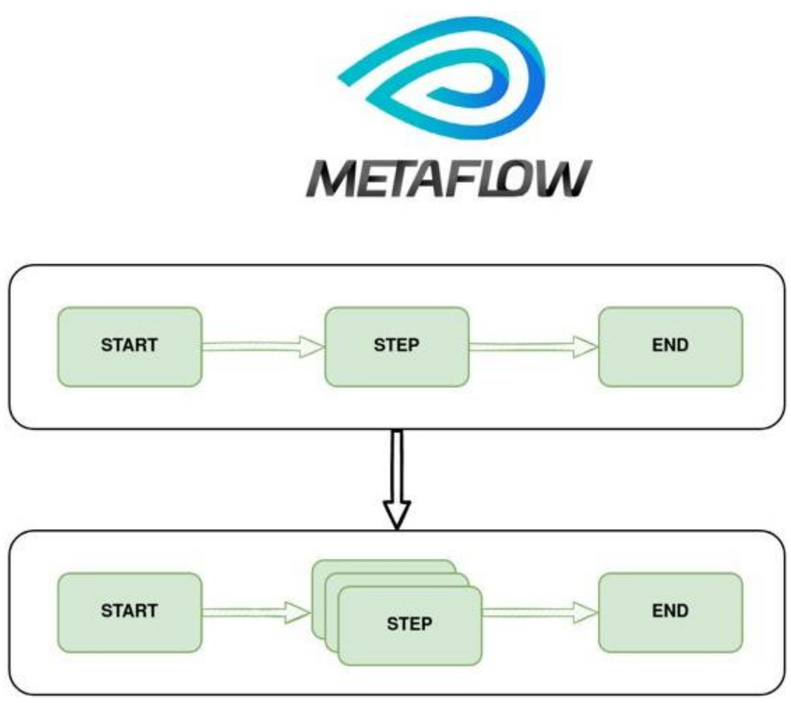
- **Integration:** With which components is or could be connected?
  - DVC: Minio integration for Cloud storage
  - Metaflow: Integrates with any ML Library (Tensorflow, SciKit Learn...) and can be managed from Kubernetes

Component location



Images/Diagrams to describe the component and its processes





Get started

Typical DVC workflow

- Initialize a DVC project in a Git repo with `dvc init` or `dvc exp init`.
- Copy data files or dataset directories for modeling into the project and use `dvc add` to tell DVC to cache and track them.
- Create a simple `dvc.yaml` file to codify a data processing pipeline. It uses your own source code and specifies further data outputs for DVC to control.
- Execute or restore any version of your pipeline using `dvc repro`, or experiment on it with `dvc exp` features.
- Sharing the repository will not include locally cached data. Use remote storage with `dvc push` and `dvc pull` to share data artifacts.

https://dvc.org/doc

```python
from metaflow import FlowSpec, step

class BranchFlow(FlowSpec):

    @step
    def start(self):
        self.next(self.a, self.b)

    @step
    def a(self):
        self.x = 1
        self.next(self.join)

    @step
    def b(self):
        self.x = 2
        self.next(self.join)

    @step
    def join(self, inputs):
        print('a is %s' % inputs.a.x)
        print('b is %s' % inputs.b.x)
        print('total is %d' % sum(input.x for input in inputs))
        self.next(self.end)

    @step
    def end(self):
        pass

if __name__ == '__main__':
    BranchFlow()
```


EU2020 Horizon **Project N.877056**

ECSEL Joint Undertaking
Electronic Components and Systems for European Leadership

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Austria, Germany, Finland, Switzerland.