Project Hanse

Konstantin Strümpf

March 10, 2021

Project Hanse

The goal of this project is to build a platform that allows non-technical experts to gain insights from data. The software tool that will be developed should make professional data science tools like Pandas, NumPy and potentially others available to them without the need to learn how to code.

Concept

In order to accomplish this goal a node based tool (similar to shader nodes in Blender) will be used to model python scripts as directed acyclic graphs (DAG). Each node in this graph will represent a transformation of one or more dataframes to a new dataframe. By combining multiple nodes using edges (each node can have 0..n input and output edges) so called “pipelines” can be created, that model the steps required to transform raw data to a visualization (or any other relevant insight).

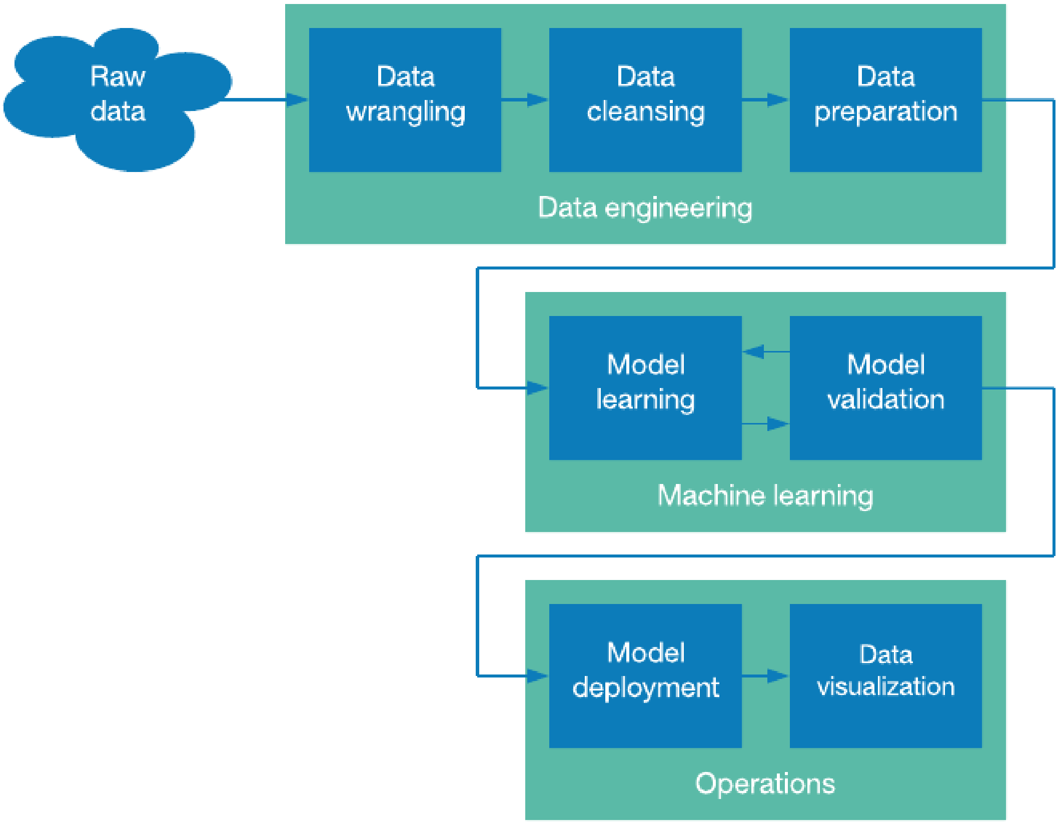

Figure 1: Typical Data Science Pipeline as described here

To further support non-technical experts an artificial neural network (built using Tensorflow) will learn from previously created pipelines and suggest users potential next nodes while they are building a new pipeline. In a first attempt this suggestion will be based on the operation (e.g. `.dropna()` operation from pandas) of the previous node and metadata (e.g. dimensions, datatypes) about the dataframe that will be transformed.

A web-based pipeline editor will make those features available. As a first step a user will be able to upload their own datasets (e.g. csv-files), in the future, however, it should also be possible make this tool available as a platform that allows trading and sharing of datasets.

Technical Architecture

Overview

This is a general overview of the components involved in a somewhat functional system. The first proof of concept will primarily focus on implementing the communication between the `Pipeline Service`, `Python Workers` and the `Dataset Store`. All data will be held in-memory and will not be persisted to disk in order to reduce the complexity of this step. In a second iteration it is planned to also include the `Suggegstion Service` that will allow prediction of potential next nodes based on previously created pipelines. In a third step a basic frontend (`Pipeline Editor`) will be implemented.

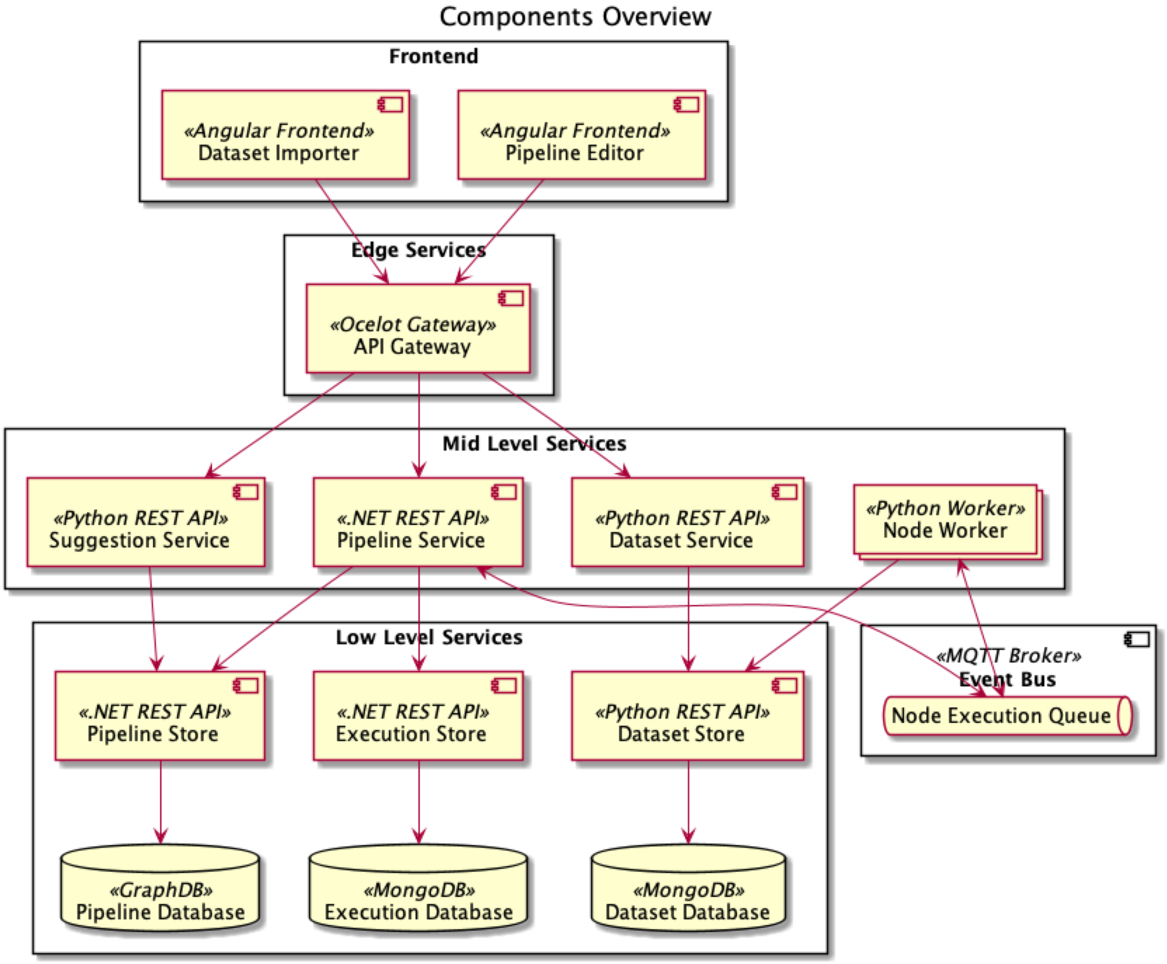

Figure 2: Architecture Diagram Overview

The following sections will be adapted and extended as the project progresses.

Sequence Diagram Iteration 1

The first iteration, implementing the execution of a pipeline, follows this diagram:

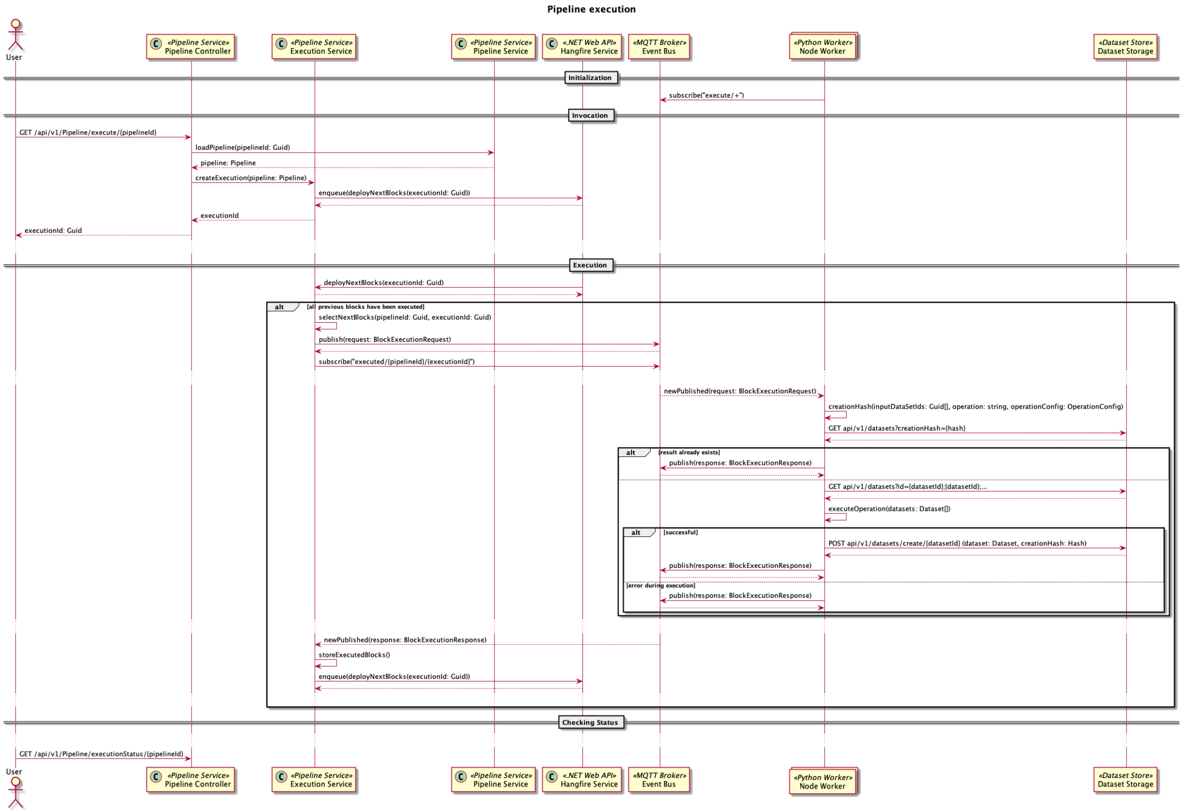
Pipeline execution
User
«Pipeline Service» Pipeline Controller
«Pipeline Service» Execution Service
«Pipeline Service» Pipeline Service
«.NET Web API» Hangfire Service
«MQTT Broker» Event Bus
«Python Worker» Node Worker
«Dataset Store» Dataset Storage
Initialization
subscribe("execute/+")
Invocation
GET /api/v1/Pipeline/execute/{pipelineId}
loadPipeline(pipelineId: Guid)
pipeline: Pipeline
createExecution(pipeline: Pipeline)
enqueue(deployNextBlocks(executionId: Guid))
executionId
executionId: Guid
Execution
deployNextBlocks(executionId: Guid)
alt
[all previous blocks have been executed]
selectNextBlocks(pipelineId: Guid, executionId: Guid)
publish(request: BlockExecutionRequest)
subscribe("executed/{pipelineId}/{executionId}")
newPublished(request: BlockExecutionRequest)
creationHash(inputDataSetIds: Guid[], operation: string, operationConfig: OperationConfig)
GET api/v1/datasets?creationHash={hash}
alt
[result already exists]
publish(response: BlockExecutionResponse)
GET api/v1/datasets?id={datasetId};{datasetId};...
executeOperation(datasets: Dataset[])
alt
[successful]
POST api/v1/datasets/create/{datasetId} (dataset: Dataset, creationHash: Hash)
publish(response: BlockExecutionResponse)
[error during execution]
publish(response: BlockExecutionResponse)
newPublished(response: BlockExecutionResponse)
storeExecutedBlocks()
enqueue(deployNextBlocks(executionId: Guid))
Checking Status
GET /api/v1/Pipeline/executionStatus/{pipelineId}
User
«Pipeline Service» Pipeline Controller
«Pipeline Service» Execution Service
«Pipeline Service» Pipeline Service
«.NET Web API» Hangfire Service
«MQTT Broker» Event Bus
«Python Worker» Node Worker
«Dataset Store» Dataset Storage

Figure 3: Sequence Diagram Iteration 1

Implementation

The code for the proof of concept is stored in this git repository https://github.com/project-hanse/prototype-a.

The technologies used for this project are primarily .NET 5 using C# for the services that are not directly interacting with datasets. Services handling datasets (e.g. `Dataset Service` when importing new datasets) or services that execute single nodes of pipelines (e.g. `Node Worker`) will be implemented using Python and appropriate libraries like Pandas and NumPy. For storing data in a persistent way databases like MongoDB and GraphDB could be used. For the `Event Bus` a MQTT message broker like Eclipse Mosquitto will be used. This can be replaced by a more scalable technology like RabbitMQ in the future. All services will be dockerized to allow for a simple deployment.

How to use

An installation of Docker and Docker Compose is required for running the prototype. Please checkout the installation guides for Docker Desktop for Mac or Windows.

For starting the prototype open a shell (cmd.exe on Windows, Terminal on Mac), navigate to the project's root directory and run: `docker-compose build && docker-compose up`.

You can then open a new browser window and navigate to http://localhost:5000/index.html. You can then test the Pipeline Service via the provided Swagger UI. To stop the prototype go back to your shell and press ctrl + C (this will send a `SIGINT` signal to the prototype telling it to shutdown).

Vision

The tool developed in this prototype could become the central `Node System` of a larger platform that allows data providers and data consumers to exchange, transport and trade data.

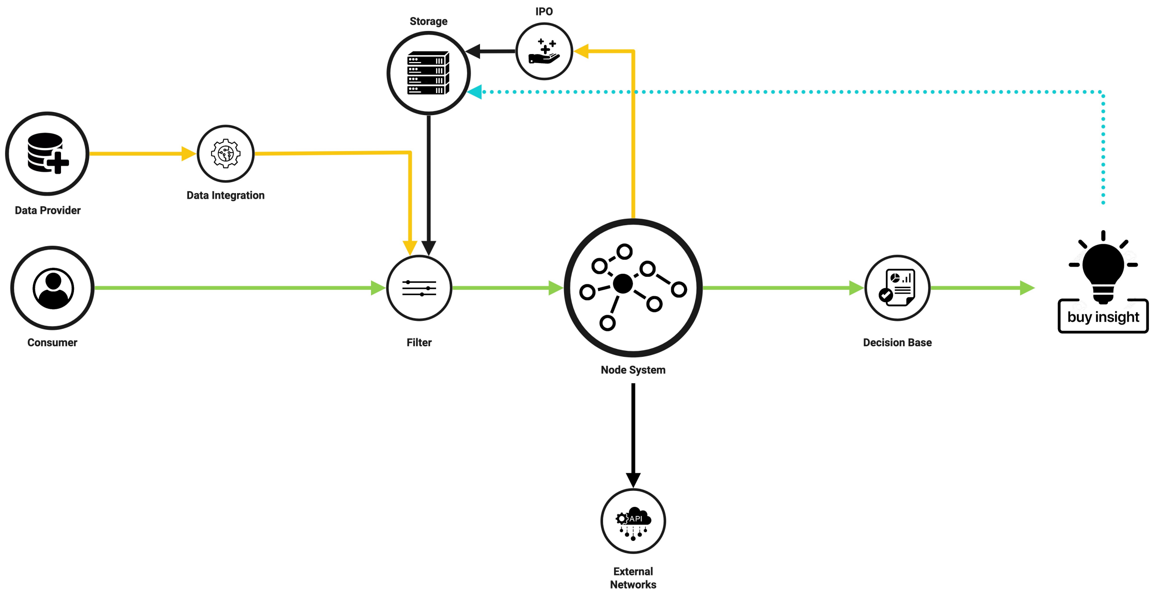

Figure 4: Platform Vision