Framework and Patterns for Machine Learning as Microservices

***Abstract*: This study explores implementing and using machine learning in an easy, scalable, and sustainable way to solve common everyday problems. There are multiple challenges with new technology adoption and constraints around accessing and using data, running computational workload, especially when using multiple cloud vendors and proprietary technologies. The paper proposes adopting the microservices architecture style to implement and practice machine learning, applicable for a broad set of machine learning frameworks and algorithms.**

***Keywords:* machine learning, deep learning, data analytics, microservices, containers, cloud native, open source.**

I. INTRODUCTION

Machine Learning (ML) is an application of artificial intelligence (AI) that automatically learns from data and is able to make guesses and predictions that are statistically verifiable as accuracy and subsequently prove good enough to be applicable and useful. ML represents data analytics practices and patterns that teach computers to do what humans and animal perform naturally, such as learn from prior experience, use evidence to inference and abstract. ML models are established by learning directly from the data and in some cases taking input from humans of other machines (ex: supervised learning). Machine Learning uses algorithms to learn from data, perform statistical processing using the data and performs subsequent predictions. Modern implementations of ML develop and harvest ML Models and produce programs that can be implemented and executed as jobs. Jobs will consume data and are implemented on various platforms, such as in-house ("on-premise") computers and/or on public cloud platforms.

The proposed ML Framework and Patterns are covering key practical elements of applied ML, with focus on simplification, streamlining and making it accessible for everyday practitioner to implement them end-to-end (E2E) and produce practical results. Pattern Recognition (PR) is the process of recognizing patterns by using Machine Learning Algorithms. PR helps detect characteristics of data, including features, classes, clusters and other properties that yield information about a given system, service or studied entity. Microservices represent engineering patterns to developing applications as a suite of small, independently deployable services built in alignment with key business functions of a solution. Microservices enable simplification and streamlining, while open-source and cloud-native technologies will provide accessibility and ability to implement on-premise and/or public-cloud agnostic. The proposed ML framework includes practices for developing, training and deploying deep learning models as we microservices.

II. CHALLENGES WITH PRACTICAL MACHINE LEARNING

The multitude of today's machine learning frameworks, together with the broad spectrum of computing platforms and associated technologies for virtualization, computation acceleration and real-time streaming make ML projects implementation very complex and hard to accomplish at scale and as a service. This is made worst by presence of multiple public cloud services providers that use different and many times proprietary technologies that can easily lock in customers and make them cloud and platform dependent. For example, when using Amazon Web Services (AWS) the offered ML solution is based on SageMaker Framework, while within Azure the preferred technology is based on MXNet Framework. Often proprietary technology will lock-in developers and data scientists to sub-optimal and expensive solutions, thus negatively impacting their research and ML implementation efforts.

The challenges this paper addresses are defining simple enough framework with patterns and practices that can be applied for a broad set of problems, leverage multiple open source machine learning technologies, be easy to implement and use anywhere, including developer workstations, on-premise computers and public cloud environment. To achieve this, the patterns and practices must be computation, operating system and cloud-platform agnostic

III. SOLUTION STRATEGY

The solution strategy entails researching and developing (R&D) a framework and underpinning patterns that leverage the following elements:

- REST-based microservices architectures and development patterns
- Light-weight virtualization components, that implement simplified Software Stack for various ML frameworks, such as Nvidia CUDA Stack for TensorFlow.
- Cloud agnostic big data components that enable seamless parallel and distributed functionality and scalability
- Open Source technologies including Big Data, Virtualization and ML Frameworks
- Cloud Native tools and technologies that enable cloud provider agnostic solutions

- Opensource workflow type solutions that implement multiple practices end-to-end across ML projects and solutions. Such workflows enable the training, improvement, test, validation and use of ML Models and associated ML Features.

IV. TECHNOLOGY LANDSCAPE

Let's review the key cloud computing concepts definitions.

Virtualization is one of the key technologies for cloud computing, and virtualization in the cloud covers multiple areas:

- Compute: includes virtual machines and other components that use CPUs or GPUs
- Compute acceleration with GPUs; some cloud providers enable GPU virtualization as well.
- Networking, that is required components to communicate. Networking has multiple virtualization implementations, such as VLANs, Overlay Networks, VPCs (virtual private clouds) or VDCs (virtual data centers).
- Virtualized storage attached to virtual machines / compute and unattached storage exposed as API etc.
- Data virtualization and metadata management

Cloud Computing is type of shared multi-tenant computing which usually provides remote pool of computing resources and compute services via the internet. The representative cloud compute services are the following:

- Infrastructure as a Service (IaaS), which provides virtualized compute (virtual machines with dedicated operating system), virtualized storage and virtualized network (virtual LANs; VLANs).
- Container as a service (CaaS) also provides virtualized compute which is abstracted away from operating systems using container packaged runtime environment including libraries and applications, Containers don't have dedicated operating system. Containers communicate with the operating systems kernel, as such hey inherit a series of process management and security functions
- Function as a Service (FaaS) , which is a lightweight, event-based, asynchronous compute solution that allows creating and running small single-purpose functions that can be instantiated within the cloud without the need to instantiate a server or a runtime environment.
- Jobs (cloud runs) that are stateless containers invoked via HTTPS requests. Jobs can run on top of CaaS or simply abstracted within the provider's cloud environment, such as Google Cloud Run. Sometimes these jobs or runs are called Jobs as a Service (JaaS).

The "**Cloud-Native**" concept emerged within the last 4 years and is characteristic to service or applications that can leverage properties and interfaces offered by cloud computing providers or frameworks, but their code packaging libraries are independent from the cloud provider platforms. Cloud native applications leverage elements from the "The Twelve-Factor App" definitions (https://12factor.net) which promotes concepts for optimal design and deployment in the cloud environment such as declarative formats for setup automation, clean contract with operating system / cloud provider, continuous build and deployment and minimum difference between development and production environments. While many of the concepts of cloud computing implementations are similar, cloud service providers implemented in their cloud environment many proprietary technologies and solution. These proprietary technologies impacted Big Data, Machine Learning, AI, DevOps and other cloud services areas, making very difficult to migrate machine learning applications between cloud providers, as well as between on-premise solutions and cloud providers. The **Cloud Native Computing Foundation** (CNCF) was established as part of the nonprofit Linux Foundation to promote open source software that is cloud agnostic and can be practically deployed to any public cloud of choice without major modifications and re-engineering effort. "CNCF serves as the vendor-neutral home for many of the fastest-growing open source projects, including Kubernetes, Prometheus and Containerd" (cncf.io main page)

Microservices, also referred to as Microservice Architecture, represent an architectural style that provides framing for applications as a structured collection of REST-based services, with the following main properties:

- Easy maintainable and testable
- Independently deployable
- Loosely coupled
- Organized around business capabilities and functions
- Owned by small teams

In dept description and references for microservices are available at https://microservices.io

V. DESIGN PATTERNS AND FRAMEWORK

Design patterns present good practice solutions to frequently occurring problems in Information technology (IT) and Computer Science (CS) solutions design, such as for example object-oriented software patterns, machine learning patterns and others. Their correct application in a solutions design may significantly improve its quality and usability attributes such as ability to reproduce, maintain, scale, accelerate as computation and other attributes. According to Steven Bradley in [1], design patterns, design components, and design frameworks are concepts related to each other. Within this chapter, I will refer to them simply just as

patterns, components and frameworks. These three entities help software developers and software engineers developing and implementing modular design, reuse code and machine learning artifacts, implement built-in flexibility and separation of functions or services. Patterns are abstract solutions that have been observed to work for specific tangible problems. They capture common characteristics of similar problems, describe solution elements used, including pros, cons and potential resulting implications. Collecting and organizing representative patterns to a common problem, can lead to a pattern library or pattern catalog. Frequently pattern language and//or component libraries (example stored in Github.com repos) can emerge as well.
Components represent the actual implementations of design patterns. Components provide artifacts, such as code we can reuse in future similar projects. Components should be flexible to enable adoption and adaptation, including patterns-based mechanisms for their modification. There can be multiple components for a single design pattern, as there can be multiple solutions to a problem.

Frameworks combine components, design patterns and package them together following some kind of services-based design, such as Services Oriented Architecture (SOA), Micro Services Architecture or others.

$$\text{Framework} = \sum \text{Components} + \sum \text{Design Patterns}$$

Frameworks support tools and application programming interfaces (APIs) that connect its instantiated components. By nature, frameworks tend to be more specialized.

In general, comparing patterns with frameworks, the following differences can be observed:

- Patterns are more abstract and less specialized than frameworks.
- Patterns represent smaller and more abstracted architectural elements than frameworks.

There are four (4) essential attributes for design patterns, as they emerged over the years:

- Name - defines how we will reference it
- Problem — defines the problem, including context and when the patterns is applicable. It also includes requirements and constraints
- Solution — specifies the elements of the pattern, including their relationships, function, and interactions.
- Implications (consequences) – these result from trade-offs in using this pattern. This is critical for evaluating design alternatives. It is good practice to add here pros and cons as well, which help better describe the implications.

VI. ACCELERATING COMPUTATION USING GPUS

Since the evolution of the Graphical Processing Units (GPU), there were an overwhelming number of studies published about their use to accelerate computation for machine learning and deep learning workloads. Kamil Aida-zade at al. within [3] compared running deep neural networks on GPU and CPU-based frameworks trained using the MNIST dataset. They leveraged CUDA (Compute Unified Device Architecture) parallel computing technology developed by introduced by NVIDIA Inc, which is dominant in the industry using GPUs for computation. The authors clearly demonstrated that GPU based deep learning frameworks are computationally multiple times more effective and faster compared to CPU technology. Youngrang Kim at al. in [4] investigated building Large-scale Deep Learning Framework based on Heterogeneous Multi-GPU Cluster, taking advantage of parallel GPU-based computing using multiple GPU worker nodes. The authors evaluated various distributed deep-learning scenarios, distributed within a Tensorflow based machine learning framework. The study demonstrated the efficiency of parallelizing workload across four different GPU systems, even when heterogeneous by nature. Eun-Ji Lim at al. in [5] proposed a Shared Memory based framework for Fast Deep Neural Network Training, where multiple deep learning workers shared training parameters (“weights”) using remote shared memory (RSM).

VII. CONTAINERIZED WORKLOADS FOR MACHINE LEARNING

Pengfei Xu at al. in [6] analyzed the feasibility of running deep learning workloads in Docker containers and performed a series of tests and evaluation regarding tools and technology performance and scalability. The authors concluded that deploying deep learning technology into docker containers is feasible and can benefit overall solution due to its flexibility, lightweight, and resource isolation abilities. Different deep learning software or different versions of the same software can coexist one the same systems yet enabling multi-tenant use and good performance.

Rupesh Raj Karn at al. in [7] has experimented with a Cloud DevOps type solution for ML using containerize infrastructure to enable easy parallelization, on-demand deployment of components and flexible scaling. Their study covered multiple use cases implemented on container scheduler:

- Matching a single optimized model to a given context in a dynamic environment.
- Creating and building multiple models and selecting the best for a given context.
- Closed loop, auto-selection mechanism in the cloud DevOps environment.

- Using unsupervised clustering to segment dataset ahead of supervised classification.
- End-to-end comparison with Ensemble Machine Learning (EML), where different data subsets are drawn from the training set and each training subset is used to train a different classifier.
- DL implementation and its hyperparameters tuning.

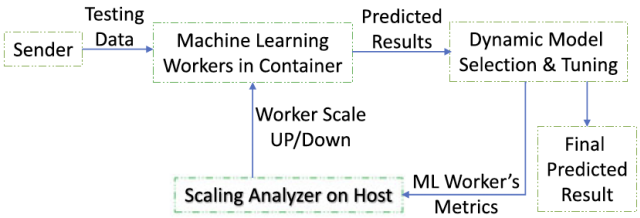

Figure 1. Containerized DevOps architecture for ML.

VIII. OVERVIEW OF MACHINE LEARNING PRACTICES

Machine Learning (ML) represents a set of practices and techniques that use algorithms and statistics to learn from data and make predictions that are good enough to be useful. ML enables computers to learn automatically without human intervention and generate outputs or actions based on algorithmic data analysis and inference performed.

Machine Learning algorithms can be categorized as follows:

- Supervised Learning, trains ML model using pre-labeled data, where the correct classes or outcome values and given. The predicted classes or values to be predicted are known and well defined from the beginning.
- Unsupervised machine learning algorithms work use data that is neither classified nor labeled. Unsupervised ML models infer a function to describe a hidden structure from unlabeled data.
- Reinforcement machine learning algorithms interacts with its environment to discover errors or rewards and take subsequent actions. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.
- Ensemble learning combines multiple ML models is the process by which multiple algorithms and resulting models, such as classification, prediction, function approximation and others, to solve a particular machine intelligence problem

IX. PATTERN RECOGNITION IN CONTEXT OF ML

Patterns are using Machine Learning algorithm to recognize patterns in the data. It focuses on classifying data based on knowledge already gained or on statistical information extracted from patterns. Pattern recognition discover data arrangements and hidden structures that yield information about a given system or data set.
Pattern recognition algorithms when used in prediction can identify statistically probable movements of time series or other type of data into the future. Their main characteristics are:

- Pattern recognition relies on data to derive outcomes and train models
- Pattern recognition systems must recognize patterns quickly and accurate, as well as classify unfamiliar entities with unknown applicable patterns quickly
- Identify patterns and entities using partially available and/or hidden data.

X. MACHINE LEARNING ARCHITECTURE AS MICROERVICES

Machine Learning solutions incorporate multiple functional building blocks, including data acquisition, data splitting into testing and training datasets, model building, model testing and finally model serving, s presented within the figure below.

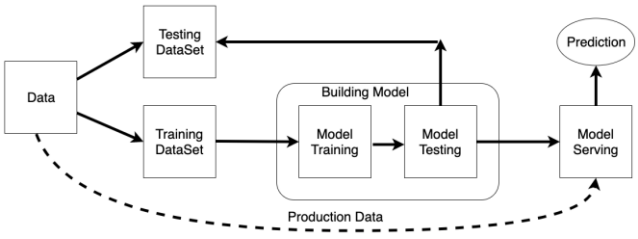

Figure 2. High-level architecture of a machine learning solution.

A Microservices style architecture places functional building blocks of a ML system into individual service components, which can be built, deployed, and scaled individually. Each component implements a standalone machine learning function and is implemented as a distinct virtual infrastructure compute component, such as a container of server-less function. Machine Learning Microservices (MLMs) are small, autonomous services that communicate between each other using REST APIs. The Figure below depicts a conceptual MLM design template for a Machine Learning Project.

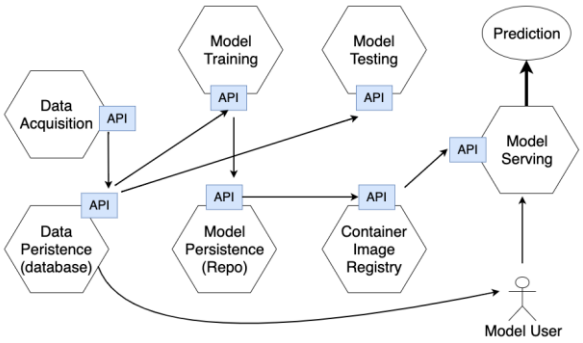

Figure 3. Conceptual MLM design for machine learning projects.

XI. PROPOSED FRAMEWORK FOR MLM PATTERNS AND PRACTICES

This paper is proposing the following high-level framework for MLM type solutions, which is broken up into three distinct categories of components:

- Data collection and data management components
- ML/DL models training and testing components
- Model serving components

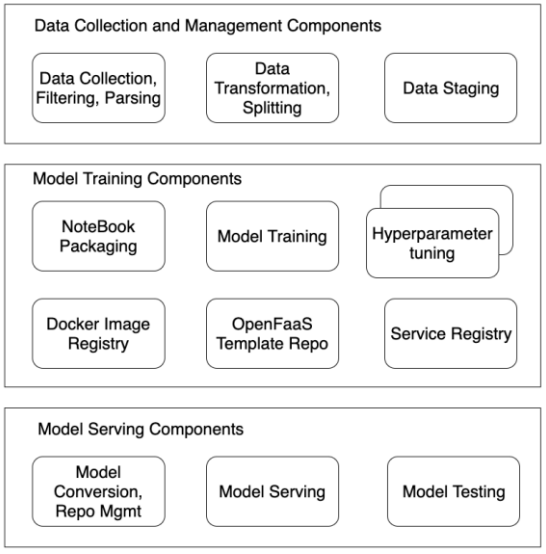

Figure 4. High-Level Architecture ML Components.

A generic architecture for data collection and management components is presented below:

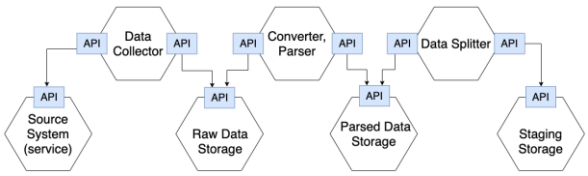

Figure 5. MLM architecture for data processing.

A generic MLM architecture for model training and testing leverages container images, deployable to Docker Swarm or Kubernetes platform. We packaged Talos enabled hyperparameter optimization as microservices, together with Keras, Tensorflow or Pytorch, as well as MongoDB and HDF5Lib containers for models and their weights persistence.

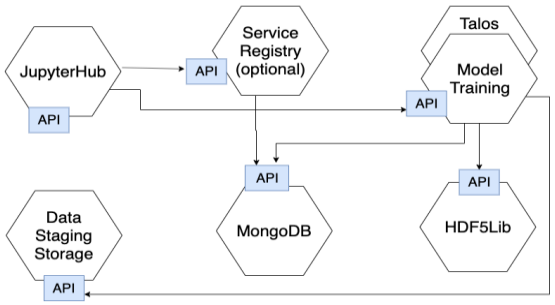

Figure 6. MLM architecture for models training and testing.

Model serving is frequently implemented as a microservice using container type deployment, as well as serverless deployment, publishing the microservice as function as a service (FaaS). We have experimented with serverless infrastructure using the OpenFaaS project (openfaas.com) and single node Kubernetes environment (minikube.sigs.k8s.io).

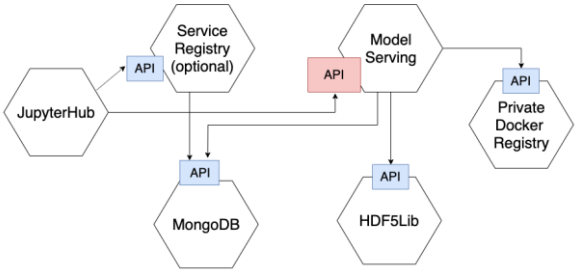

Figure: 7. MLM architecture for model serving.

XII. ALGORITHMS AND FRAMEWORKS WE USED MLM WITH

We evaluated the MLM concept, including selective implementation and testing for the following ML frameworks:

- SciKit-Learn (scikit-learn.org)
- Keras and Tensorflow (keras.io, tensorflow.org)
- Rapids-AI project (rapids.ai) and its machine learning library CuML (github.com/rapidsai/cuml)
- Machine Learning algorithms augmented with Quantum Computing (QML algorithms), such as for example:
    - QSVM from IBM Qiskit SDK (github.com/qiskit/aqua/algorithms).
    - QKMeans example from https://github.com/enniogit/Quantum_K-means

Below are sample supervised learning algorithms considered.

Scikit-Learn: Support Vector Machines

- Classification: from sklearn import svm; model = svm.SVC()
- Regression: from sklearn import svm; model = svm.SVR()

Scikit-Learn: Stochastic Gradient Descent

- Classification: from sklearn.linear_model import SGDClassifier
- Regression: from sklearn.linear_model import SGDRegressor

Scikit-Learn: Nearest Neighbors

- Classification: from sklearn import neighbors; model = neighbors.KNeighborsClassifier()
- Regression: from sklearn import neighbors; model =neighbors.KNeighborsRegressor()

Keras

- Classification: from keras.wrappers.scikit_learn import KerasRegressor

- Regression: from keras.wrappers.scikit_learn import KerasRegressor

Nvidia: CuML

- Classification:
  - from cuml import svm; model = svm.SVC()
- Regression:
  - from cuml import LinearRegression
  - from cuml import LogisticRegression

QML:

- Classification: from qiskit.aqua.algorithms import QSVM

XIII. SAMPLE PATTERN DEFNITION: MLM MODEL TRAINING

- **Pattern Name: "**Machine Learning Model Training"
- **Pattern ID:** MOD-TRAIN-001
- **Also Known As:** "Model training using various frameworks and algorithms as Microservice"
- **Problem** – Implementing machine learning environment SW stack, that integrates with Python libraries, GPU drivers, scales as single or multiple nodes is not a simple task to accomplish. Having all this enabled as a Microservices, that is easy deployable as container or callable as a function makes developers and students life much easier.
- **Purpose (Intent):** Implement a Microservice that takes a generic Python script or Jupyter Notebook and creates a function for it. The function will require as input pointer to training and test data, the loss function and the model name that will be saved and stored. For deep learning models training, may have to specify parameters for defining the neural network.
- **Motivation (Forces):** We need an easy, API based way to train model, available to developers or students with limited knowledge about machine learning and associated tooling.
- **Applicability:** This pattern is applicable for various deep learning networks. Requirements are ability to use various python libraries, input code based on Jupyter notebook or python program, use CPU or GPU based computation. In case of Quantum Neural Networks, the backend would be a Quantum simulator, or an IBM Quantum computer accessed as a Microservices using Qiskit or other Open Source SDK.
- **Structure:** The proposed structure for the Microservice Pattern is based on an initial Jupyter Notebook, that is converted into a microservice. Input parameters must be sent to the notebook, as such as initial step need to parameterize the notebook.
-

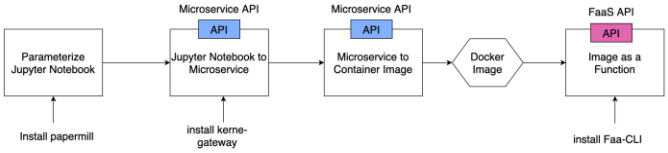

Figure 8. Generic Structure Model Training as a Service.

- **Participants:** The following entities participate within this pattern: Jupyter Notebook, Python Libraries, Papermill product, Github packages (Kernel-Gateway, OpenFaaS), Docker Registry Server: Private Registry or DockerHub.
- **Consequences:**
  - The user will have no direct access to Jupyter; rather via API and sending input parameters
  - Training and Test data must be prepared to Github Repo or some kind of Cloud Storage
- **Implementation (solution, may include diagram):**
  - Pattern deployed as Container; local system or cloud.
  - Pattern deployed as Function; Kubernetes in the cloud.
- **Known Uses:** This pattern can work with multiple machine learning frameworks, Scikit-Learn (extended via MlXtend), Apache Spark, Keras (uses Tensorflow or Pytorch), Nvidia CuML, experimental QML prototypes.

XIV. SAMPLE PATTERN DEFNITION: MLM DATA ACQUISITION

- **Pattern Name: "**Data Acquisition for Machine Learning"
- **Pattern ID:** DATA-AC-001
- **Also Known As:** "Data Acquisition using Open Source Collectors as Microservices"
- **Problem** – There are many different data sources, accessible via different protocols and using multiple authentication mechanisms. Diversified set of data collector capabilities and practices are required to work with these sources of data. Once data is acquired, additional data augmentation is manipulation is required, which should be performed at acquisition stage.

- **Purpose (Intent):** Define simple, easy to deploy, use and scale pattern that can handle most of data collection and processing needs. Data acquisition will have its own mini-framework of connectors (input, out-put, fan-in, fan-out, conditional, etc.) and processors.
- **Motivation (Forces):** Common data Acquisition components and practices are needed to implement this pattern to address broad scale and diversified data acquisition requirements.
- **Applicability:** Applicable to practically all Big Data and traditional relational databases type data sources. It is also applicable to large number of public cloud middleware, such as Google Big Table Aws S3, others.
- **Structure:** The proposed structure for the Data Collector is using one or multiple Microservices. In general, the data collector is deployed as one single microservice (typically as a container), with multiple staging libraries: one staging library for each of the modules within the figure, such as for input, output, processor and others.

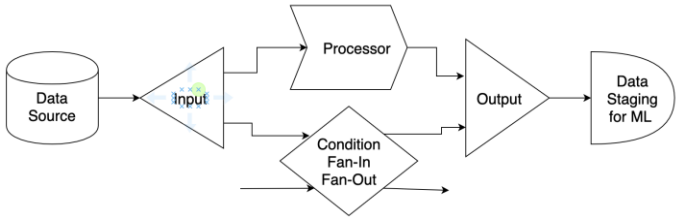

Figure 9. Generic Structure for a Data Collector.

- **Participants:** The following entities participate within this pattern:
  - Data source: Big Data Middleware, remote API, Database, web page, etc.
  - Target staging area, such as for example: Github, filesystem, cloud storage
  - Data collector itself, such as for example Apache NiFi or others
- **Consequences:**
  - Data is acquired and processed by this pattern
  - Proper authentication and authorization required to collect the data
  - Data during transfer will have to be encrypted (encryption in motion)
  - The microservice must be protected, including associated APIs
- **Implementation (solution, may include diagram):**
  - The recommended implementation is using Docker containers, with selectively added staging libraries to the Docker image.
  - The container can be deployed for continuous operation or as a job.
  - Deployment as a function is possible but recommended for simple data acquisition tasks, such as using APIs and with lightweight data processing.
- **Known Uses:** This pattern incorporates several data acquisition and integration use cases. Few examples:
  - Automated data acquisition, parsing, formatting and storage to MySQL.
    - Cloud solution with Google DataFlow.
    - On-premise solution using Apache Nifi.
  - Implementing foreign keys and subsequent joins between multiple data source (MySQL tables).
  - Enrich the data using formulas and new fields.

REFERENCES

[1] Design Patterns, Components, And Frameworks by Steven Bradley, https://vanseodesign.com/web-design/patterns-components-frameworks/

[2] Essential Framework Design Patterns, Wolfgang Pree

[3] http://choices.cs.illinois.edu/uChoices/Papers/fabio/2k/htdocs/Internal/dependence-chapter/related/EssDPs96.ps

[4] Comparison of Deep Learning in Neural Networks on CPU and GPU-based frameworks Kamil Aida-zade, E. Mustafayev, Samir Rustamov,

[5] Efficient Large-scale Deep Learning Framework for Heterogeneous Multi-GPU Cluster, Youngrang Kim, Hyeonseong Choi, Jaehwan Lee, Jik-Soo Kim, Hyunseung Jei, Hongchan Roh

[6] Distributed Deep Learning Framework based on Shared Memory for Fast Deep Neural Network Training, Eun-Ji Lim, Shin-Young Ahn, Yoo-Mi Park, Wan Choi

[7] Performance Evaluation of Deep Learning Tools in Docker Containers Pengfei Xu, Shaohuai Shi, Xiaowen Chu

[8] Dynamic Autoselection and Autotuning of Machine Learning Models for Cloud Network Analytics, Rupesh Raj Karn , Prabhakar Kudva , and Ibrahim (Abe) M. Elfadel

[9] Machine learning as a reusable microservice, Marc-Oliver Pahl ; Markus Loipfinger, NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium

[10] Packaging and Sharing Machine Learning Models via the Acumos AI Open Platform, Shuai Zhao ; Manoop Talasila ; Guy Jacobson ; Cristian Borcea ; Syed Anwar Aftab ; John F Murray

[11] A Container Scheduling Strategy Based on Machine Learning in, Microservice Architecture, Jingze Lv ; Mingchang Wei ; Yang Yu, 2019 IEEE International Conference on Services Computing

[12] Bodhisattva - Rapid Deployment of AI on Containers, Shreyas S Rao ; Pradyumna S ; Subramaniam Kalambur ; Dinkar Sitaram, 2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)