**Motivation and Architecture for Data Science Workbench**

**Audience:** This document is for data scientists and ML engineers who want to adapt their CI/CD practices to move ML solutions to production, and who want to help ensure the quality, maintainability, and adaptability of their ML pipelines.

**Various Customer Data Interaction Scenarios:**

Customers are very sensitive about their data. In any ML processing scenario, there are three roles possible: the input party (data owners or contributors), the computation party, and the results party.

In one such scenario, the data owner(s) send their data to the computation party that performs the required ML task and delivers the output to the results party. Such output could be an ML model that the results party can utilize for testing new samples.

In another scenario, the computation party might keep the ML model, and performs the task of testing new samples submitted by the results party, and returning the testing results to the result’s party.

Note: If all the three roles are assumed by the same entity, then the privacy is naturally preserved; however, when these roles are distributed across two or more entities (for ex- epsilon and the client) then we need to be careful with the privacy dimension and customer sensitivity to it.

To facilitate data scientists, data science engineers, data engineers, MLops (let’s call them users!) to interact with the data in above mentioned scenarios, we bring in a concept of Data Science Workbench (let’s call it DSWB). Using DSWB, users will be able to interact with customer data anywhere (on-cloud or on-prem). Assumption being that customer can be present on any cloud or might be having their own on-prem data center. This constraint requires us to think about building the DSWB using technologies which are cloud compatible and cloud agnostic simultaneously. ML solutions built using DSWB should be deployable anywhere (on-cloud or on-prem)

Using DSWB users will be able to perform data exploration, experiments, building pipeline, deployment of the pipeline and monitor the deployed pipeline (both system and algorithmic).

Uniformity in interacting technology (DSWB) will help data scientists, data science engineers and data engineers to quickly iterate with development and bring out solution quickly. And at the same time reuse the components from other solutions if need be.

DSWB will interact with multiple technologies and help in building ML Solution using Continuous Integration, Continuous Deployment and Continuous Training (CI/CD/CT) flow explained below.

**Various Phases of ML Solution building post customer on-boarding:**

1. **Requirement Phase:** This phase is business and data science heavy. We interact with the customer to understand the solution requirement from all the dimensions including technical constraints (on-cloud or on-prem etc) and protocol for data access.
2. **Infra-Setup Phase:** This is engineering heavy phase. In this phase DSWB is setup by the data science engineering and iron out all the technical constraints needed for seamless interaction with data.
3. **Data Exploration Phase:** This is data science heavy phase. EDA is performed in this stage by the data scientists, mathematical problem formulation is done and if need be recalibrating the requirements defined in requirement phase.
4. **Data Science Experimentation Phase:** This is data science and data science engineering heavy phase. Experimentation, Model Building and Prototyping is done in this phase.

This is most important phase success. It lays the foundation of the complete solutioning.

1. **Operationalizing ML Solution Phase:** This is data science engineering heavy phase. Pipeline Implementation using the prototyping done in data science experimentation phase is done in this phase. Pipeline development/testing/deployment is done in this phase with due engineering diligence.
2. **Monitoring ML Solution (Systemic/Algorithmic) Phase:** This will be an ongoing monitoring phase where both the system (by data engineers/data science engineers) and model (by data scientists) will be monitored.

**Operationalizing Machine Learning:**

To integrate an ML system in a production environment, you need to orchestrate the steps in your ML pipeline. In addition, you need to automate the execution of the pipeline for the continuous training of your models. To experiment with new ideas and features, you need to adopt CI/CD practices in the new implementations of the pipelines.

**ML Pipeline Automation (CI/CD/CT):**

In some use cases, the manual process of training, validating, and deploying ML models can be sufficient. This manual approach works if your team manages only a few ML models that aren't retrained or aren't changed frequently. In practice, however, models often break down when deployed in the real world because they fail to adapt to changes in the dynamics of environments, or the data that describes such dynamics.

For your ML system to adapt to such changes, you need to apply the following MLOps techniques:

* Automate the execution of the ML pipeline to retrain new models on new data to capture any emerging patterns. This is known as Continuous Training (CT).
* Set up a continuous delivery system to frequently deploy new implementations of the *entire* ML pipeline. This is known as Continuous Integration/Deployment (CI/CD).

You can automate the ML production pipelines to retrain your models with new data. You can trigger your pipeline on demand, on a schedule, on the availability of new data, on model performance degradation, on significant changes in the statistical properties of the data, or based on other conditions.

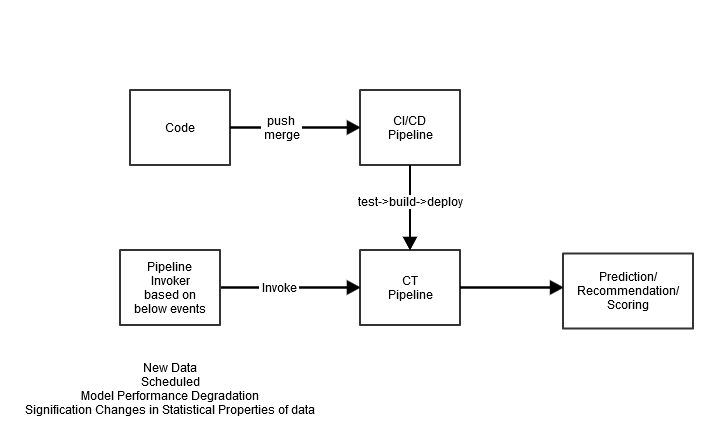
**CI/CD pipeline compared to CT pipeline:**

The availability of new data is one trigger to retrain the ML model. The availability of a new implementation of the ML pipeline (including new model architecture, feature engineering, and hyperparameters) is another important trigger to re-execute the ML pipeline. This new implementation of the ML pipeline serves as a new version of the model prediction service, for example, a microservice with a REST API for online serving or for batch scoring. The difference between the two cases is as follows:

* To train a new ML model with new data, the previously deployed CT pipeline is executed. No new pipelines or components are deployed; only a new prediction service or newly trained model is served at the end of the pipeline.
* To train a new ML model with a new implementation, a new pipeline is deployed through a CI/CD pipeline.

To deploy new ML pipelines quickly, you need to set up a CI/CD pipeline. This pipeline is responsible for automatically deploying new ML pipelines and components when new implementations are available and approved for various environments (such as development, test, staging, pre-production, canary, and production).

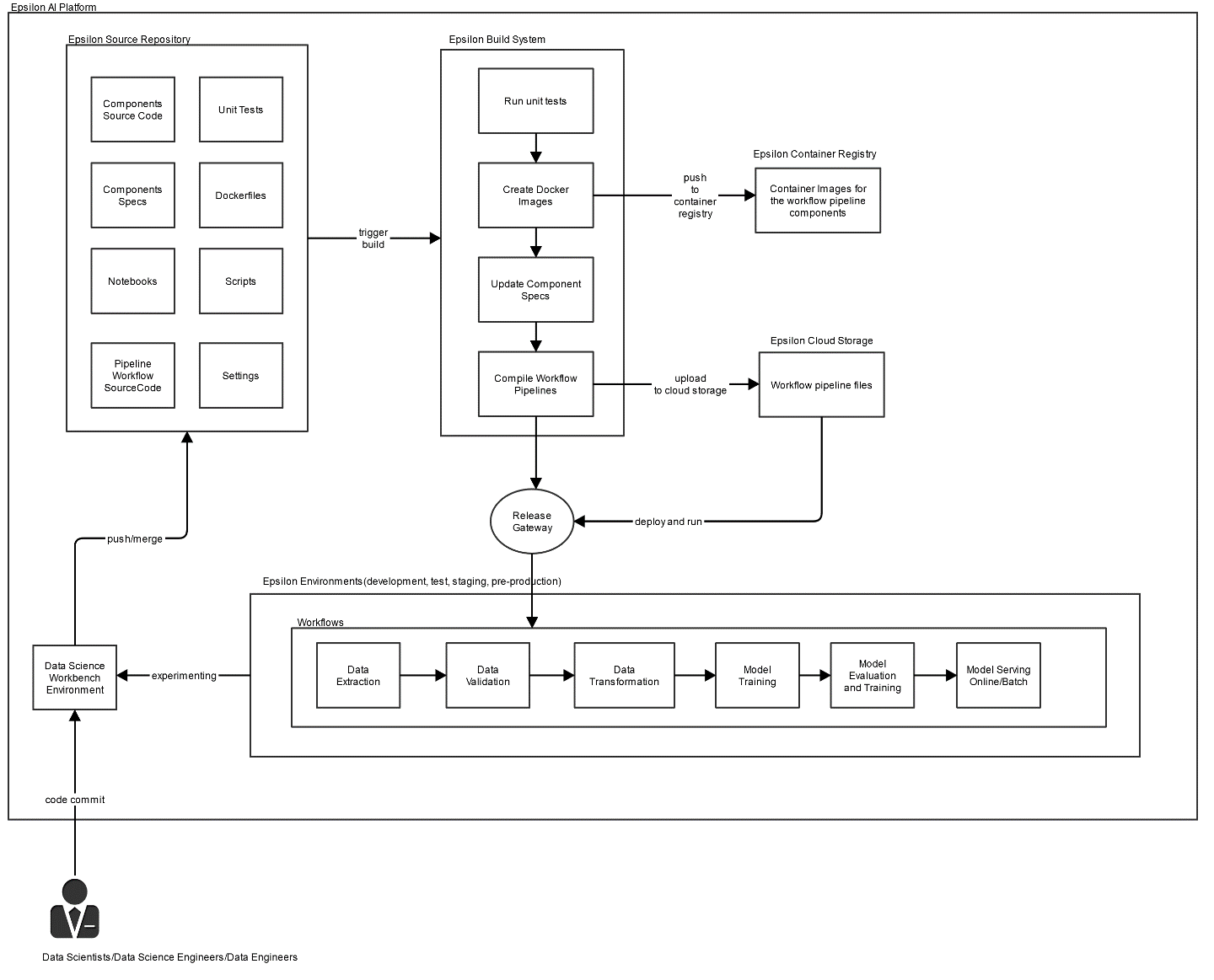
The following diagram shows the relationship between the CI/CD pipeline and the ML CT pipeline.



The output for these pipelines is as follows:

* If given new implementation, a successful CI/CD pipeline deploys a new ML CT pipeline.
* If given new data, a successful CT pipeline serves a new model prediction service.

**ML CI/CD System Overview:**



As we can see in above diagram, the DSWB interacts with multiple other subsystems and it needs them to be fully operational:

1. Epsilon Source Repository: Something like Git, GitLab, Bitbucket
2. Epsilon Build System: Something like Jenkins
3. Epsilon Container Repository: Something like Docker Registry
4. Epsilon Cloud Storage: Something like S3
5. Epsilon Workflow Platform: The Workflow platform consists of the following:

* A user interface for managing and tracking experiments, jobs, and runs.
* A workflow engine for scheduling multistep ML workflows.
* Interfaces for defining and manipulating pipelines and components.
* Notebooks for interacting with the system using the Python SDK.
* An ML Metadata store to save information about executions, models, datasets, and other artifacts.

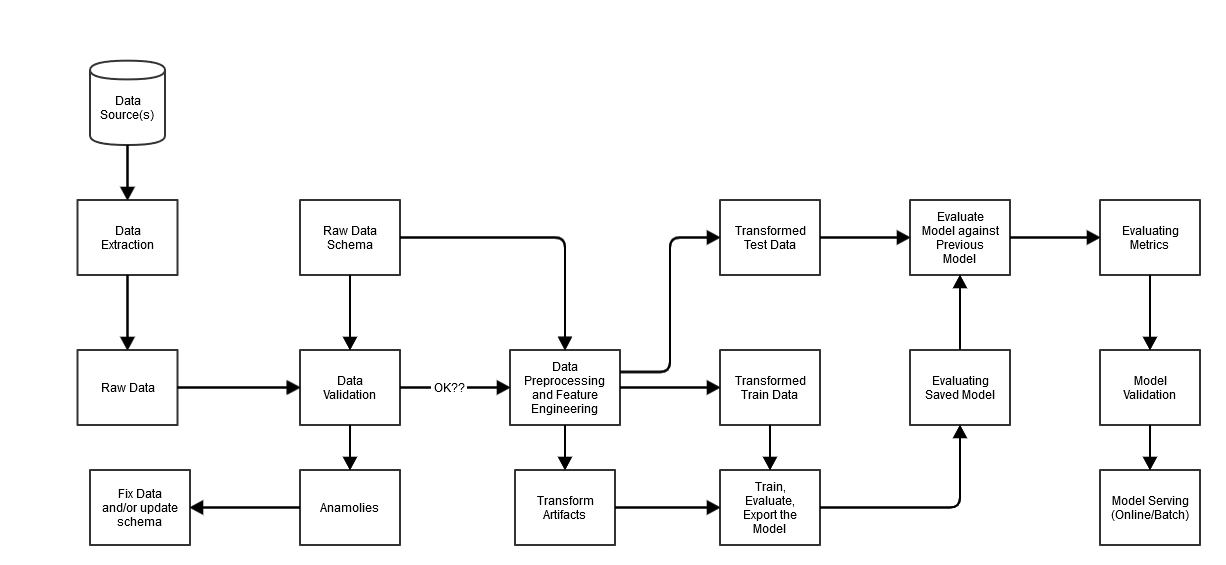
The following constitutes a Workflow pipeline:

* A set of containerized ML tasks, or components. A pipeline component is self-contained code that is packaged as a docker image. A component takes input arguments, produces output files, and performs one step in the pipeline.
* A specification of the sequence of the ML tasks, defined through topology. The topology of the workflow is implicitly defined by connecting the outputs of an upstream step to the inputs of a downstream step. A step in the pipeline definition invokes a component in the pipeline. In a complex pipeline, components can execute multiple times in loops, or they can be executed conditionally.
* A set of pipeline input parameters, whose values are passed to the components of the pipeline, including the criteria for filtering data and where to store the artifacts that the pipeline produces.

Containerizing tasks in Workflow Pipelines has the following advantages:

* Decouples the execution environment from your code runtime.
* Provides reproducibility of the code between the development and production environment, because the things you test are the same in production.
* Isolates each component in the pipeline; each can have its own version of the runtime, different languages, and different libraries.
* Helps with composition of complex pipelines.
* Integrates with ML metadata store for traceability and reproducibility.

**ML CT System Overview:**



The following steps can be completed manually or by an automated pipeline:

1. **Data extraction:** The first step is to extract the new training data from its data sources. The outputs of this step are data files that are used for training and evaluating the model.
2. **Data validation:** This step validates the data against the expected (raw) data schema. The data schema is created and fixed during the development phase, before system deployment. The data validation steps detect anomalies related to both data distribution and schema skews. The outputs of this step are the anomalies (if any) and a decision on whether to execute downstream steps or not.
3. **Data transformation:** After the data is validated, the data is split and prepared for the ML task by performing data transformations and feature engineering operations. The outputs of this step are data files to train and evaluate the model. In addition, the transformation artifacts that are produced help with constructing the model inputs and with exporting the saved model after training.
4. **Model training and tuning:** This step implements and trains the ML model using transformed data produced by the previous step. The output of this step is a saved model that is used for evaluation, and another saved model that is used for online serving of the model for prediction.
5. **Model evaluation and validation:** When the model is exported after the training step, it's evaluated on a test dataset to assess the model quality. The output of this step is a set of performance metrics and a decision on whether to promote the model to production
6. **Model serving for prediction:** After the newly trained model is validated, it's deployed as a microservice to serve online predictions or batch predictions. The output of this step is a deployed prediction service of the trained ML model. You can replace this step by storing the trained model in a model registry. Subsequently a separate model serving CI/CD process is launched.