# Distributed Data and Model Hosting System Design

# Ryan Pacheco

## Scenario

A consortium of enterprise partners requires a unified service to search their proprietary datasets. Each partner's data must remain within their private, on-premise network.

## Core Requirements

- Onboard distributed datasets securely
    - Have a central process where partners can share access tokens or secret keys to register with a data set in their on premise data stores.
- Secure data scharing
    - Query across multiple independent networks with not centralized data store
        - Multiple organizations have data and we need to be able to access them all
- Algorithmic flexibility
    - Create system to use different ML models for different use cases
        - Be able to switch models on demand or route traffic to the correct model

## Design Points

### 1. Data Onboarding & Secure Sharing Framework

**Problem**

Partners need a secure, low-friction process to make their datasets discoverable without moving the data itself.

**Requirements**

1. Partner needs to be able to register and index a dataset
    - Data is stored in an on-premise system
2. Only metadata and vector embeddings can be transmitted to central service
    - Only essential data can be sent to central service
3. Central service must enforce strict authorization policies
    - Manage user access to querying partner datasets
4. Central service must securely broadcast a query request to on-premise agents and receive back a list of candidate IDs or anonymized results.
    - Central service must authorize a query request with on-premise agent and process a list of anonymized results or candidate IDs to fulfill query request in a user readable way

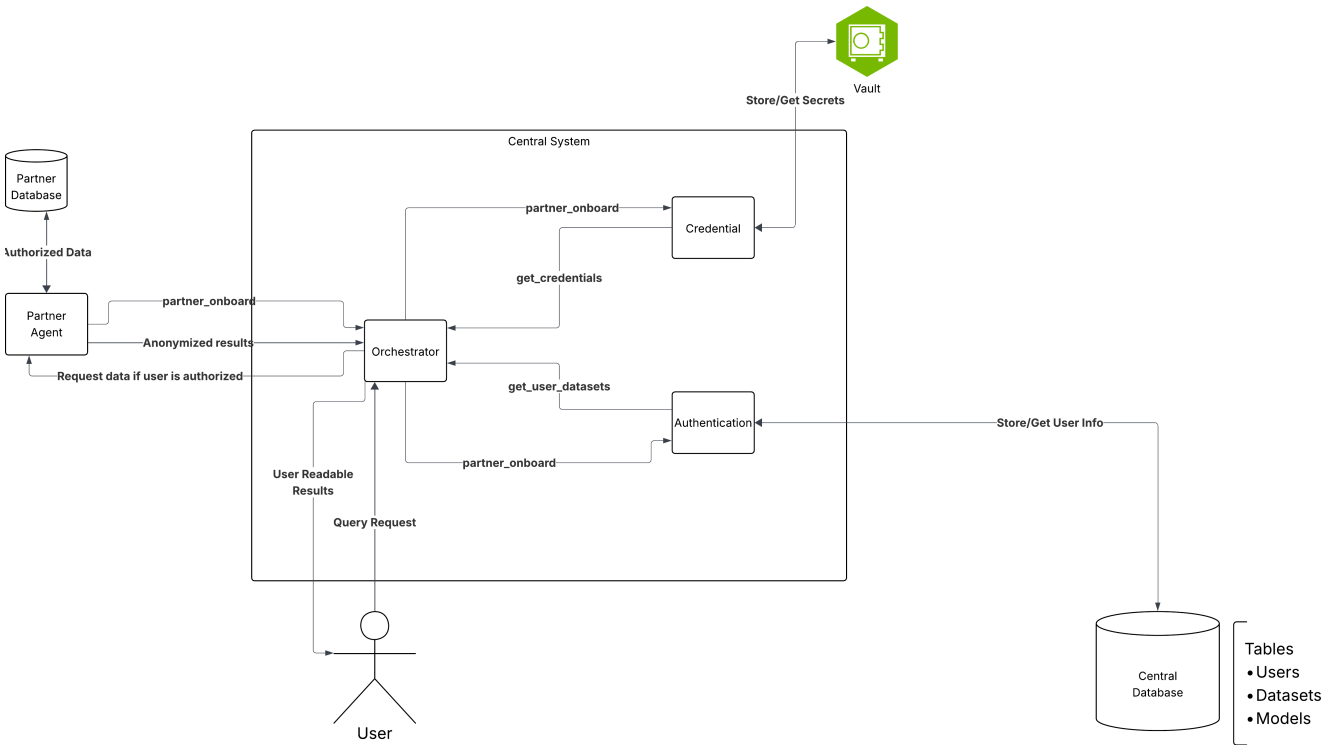
**Proposed Approach**

- Central System

- Handle all the authorization and query orchestration for data requests
- Central Orchestrator
  - Microservice containing the following endpoints
    - partner_onboard
      - POST
      - Allow a partner to store access tokens and specify which users are allowed access to query partner data
      - Will interface with Authorization Service and Credential Service
      - Parameters
        - `partner_id` - UUID
        - `dataset_id` - UUID
        - `authorized_users` = [string]
      - Returns
        - None
    - query_partner
      - POST
      - Authenticate a user and properly format a query for a partners on-premise agent and receive anonymized results and decode for user readability
      - Parameters
        - `user_id` - UUID
        - `dataset_id` - UUID
        - `query` - string
      - Returns
        - `query_results` - JSON
    - get_user_credentials
      - GET
      - Returns which datasets a user has access to query
      - Audit endpoint for admins to see which datasets a user has access to query
      - Parameters
        - `user_id` - UUID
      - Returns
        - JSON List of `dataset_ids`
- Authentication Service
  - get_user_datasets
    - Queries Postgres or other database that contains user permissions
    - Parameters
      - `user_id` - UUID
    - Returns
      - `authorized_data_sets` - List of `dataset_ids`
  - onboard_dataset
    - Allows a partner to set user access to the dataset
    - Parameters
      - `dataset_id` - UUID
      - `users` - List of user_ids that are allowed access to query provided dataset
    - Returns
      - `datasets` - List of datasets a user has access to, can be an empty list

    - Credential Service
        - store_credential
            - Allows a partner to store access tokens or secret keys for specific datasets in a hashicorp vault
            - Parameters
                - `dataset_id` - UUID
                - `secret` - string
            - Returns
                - None
        - get_credentials
            - Retrieves the secret from the vault
            - Parameters
                - `dataset_id` - UUID
            - Returns
                - `secret` - string
                - None
- Partner Agents
    - On-premise service to assist partner integration with central service
    - Onboard partner into central system
        - Send dataset and credentials to central system
    - Handles central service query requests
        - Receives a query from central system and returns metadata back to service

**Architecture Diagram**




## 2. Framework for Swappable Models & Algorithms

**Problem**

The system cannot be tied to a single algorithm. It must support the dynamic use of different models for embedding, retrieval, and ranking based on the query, data type, or user.

**Requirements**

1. Machine learning models need to be pluggable compenents in the infrastructure
    - Container based
2. Central orchestrator to dynamically select and invoke specific models for specific use cases
    - Optional - Link models in a child/parent relationship for more nuanced results
        - Child model takes result of parent model as input
3. On-premise agent delivers only non-sensitive feature vector
    - Based on instructions from selected model

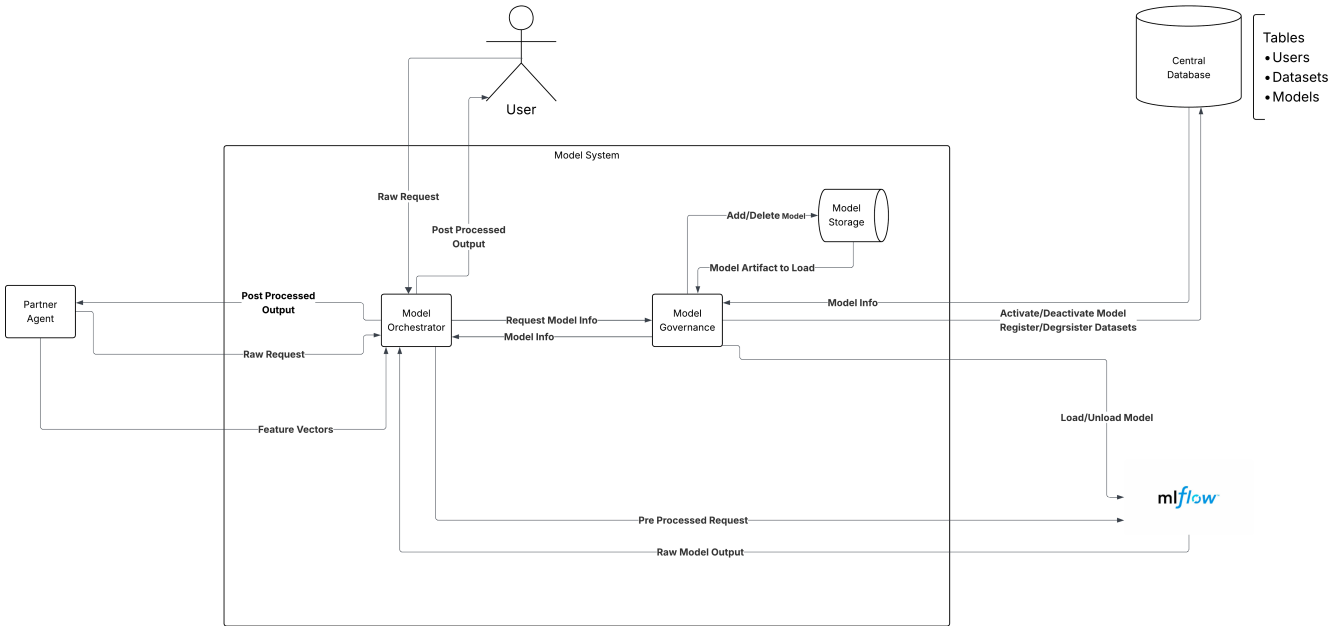
**Proposed Approach**

- Model Registry
    - MlFlow
        - Host and store model artifacts
- Serve models in Docker containers on Kubernetes clusters
- Model System
    - A central source of knowledge for all things machine learning model related
    - Model table present in Postgres or other database chosen in stage 1
        - Keeps track of models, their status, what datasets they are linked to
    - Model storage
        - A location for storing raw model artifacts
        - Could be local file store or cloud based like S3
    - Model Governance
        - add_model
            - POST
            - Adds a model model storage
            - Registers model in Model table with any provided metadata, uploads model artifact, and loads model into MlFlow
                - Default to inactive
            - Parameters
                - `model_id` - UUID
                - `model` - serialized model file (.zip, .joblib, .pkl)
                - `status` - string - **OPTIONAL**
                - `datasets` - list of `datasets_ids` the model is authorized to interact with for feature vectors - **OPTIONAL**
            - Returns
                - Success or failure message
        - delete_model
            - DELETE
            - Deletes a model from Model table, and unloads it from MlFlow, and deletes source artifact
            - Parameters
                - `model_id` - UUID

        - Returns
            - None
    - load_model
        - PATCH
        - Loads an uploaded model into MlFlow and sets status to active in model table
        - To be used when a model artifact is already uploaded but not yet hosted in MlFlow
        - Parameters
            - `model_id` - UUID
            - `datasets` - list of `datasets_ids` the model is authorized to interact with for feature vectors - **OPTIONAL**
        - Returns
            - Success or failure message
    - unload_model
        - PATCH
        - Unloads model from MlFlow and sets status to 'inactive' in Model table but does not delete model artifact and the model can be re loaded at a later time
        - Parameters
            - `model_id` - UUID
        - Returns
            - Success or failure message
    - activate_model
        - PATCH
        - Sets a models status to 'active' in the Model table
        - **ONLY TO BE USED TO REACTIVATE A PREVIOUSLY DEACTIVATED MODEL**
        - Needs a check to ensure model is currently loaded in MlFlow
        - Parameters
            - `model_id` - UUID
        - Returns
            - Success or failure message
    - deactivate_model
        - PATCH
        - Sets a models status to 'inactive' in the Model table while not unloading the model or deleting model artifiact
        - Parameters
            - `model_id` - UUID
        - Returns
            - Success or failure message
    - register_datasets
        - PATCH
        - Registers datasets the model can get feature vectors from
        - Parameters
            - `model_id` - UUID
            - `datasets` - list of `datasets_ids` the model is authorized to interact with for feature vectors
        - Returns
            - Success or failure message

        - revoke_datasets
            - PATCH
            - Removes dataset or sets from list of `dataset_ids` a model can interact with for feature vectors
            - Parameters
                - `model_id` - UUID
                - `datasets` - list of `datasets_ids` the model is authorized to interact with for feature vectors
            - Returns
                - Success or failure message
        - get_all_models
            - GET
            - Returns a list of all models in the Models table
            - Parameters
                - `Metadata` - bool - **OPTIONAL**
            - Returns
                - `Metadata` = True
                    - JSON List of all models and all metadata
                - `Metadata` = False
                    - JSON List of all models
        - get_model_metadata
            - GET
            - Returns model metadata for a specific requested model
            - Parameters
                - `model_id` - UUID
            - Returns
                - JSON List of all model metadata
        - get_feature_vectors
            - GET
            - Returns a set of feature vectors for a specified model
            - Interacts with the on-premise agents to link a model with feature vectors
            - Parameters
                - `model_id` - UUID
            - Returns
                - JSON object of feature_vectors for specified model
    - Model Orchestrator
        - Service that routes requests to the correct model
        - Interact with model governance endpoints to select the correct model and gather any metadata (feature vectors) that model needs
        - Route traffic to specified model
        - Preform any pre or post processing necessary for model to be effective.

**Architecture Diagram**




## 3. Supporting Infrastructure

### Problem

The entire system must be secure and observable, despite its distributed nature.

### Requirements

1. Use mutual TLS (mTLS) and private certificate authority to ensure only authenticated services can communicate
    - Central service/partner handshake
    - Secrets stored in vault
2. Monitor end to end query latency and diagnose bottlenecks inside partners network
    - Push-based monitoring
    - Set up alerts to monitor latency from partners

### Proposed Approach

- Ensure that all aspects of the system have a secure handshake between services
    - Central Service
    - Partner Agents
    - Model Governance
    - Ensures security and stability
    - Force strong authentication via TLS handshake with shared secrets stored in vault
- Use OpenTelemetry to monitor latency steps for each query. Log to metrics platform such as data dog to monitor the latency of queries between each client.
    - Asynchronous
        - Don't have the logging take up more time
        - Kafka
    - Alerting
        - Have data dog set up alerts and performance thresholds to alert admins to a performance bottleneck. Specifically make note of what partner network is the

bottleneck so communication can be established with partner admins to fix the bottleneck or determine how to better optimize queries to that partner.

### Architecture Diagram

Present in complete end-to-end system architecture diagram

## Complete System Architecture Diagram