Autonomous Driving Training Platform - MLOps Architecture & Workflow Overview

This document provides a high-level summary of the machine learning architecture and workflows for an autonomous driving training platform. It is organized into three core subsystems: **Collection Management**, **Experiment Management**, and **Resource Management**.

Figure 1: Overview

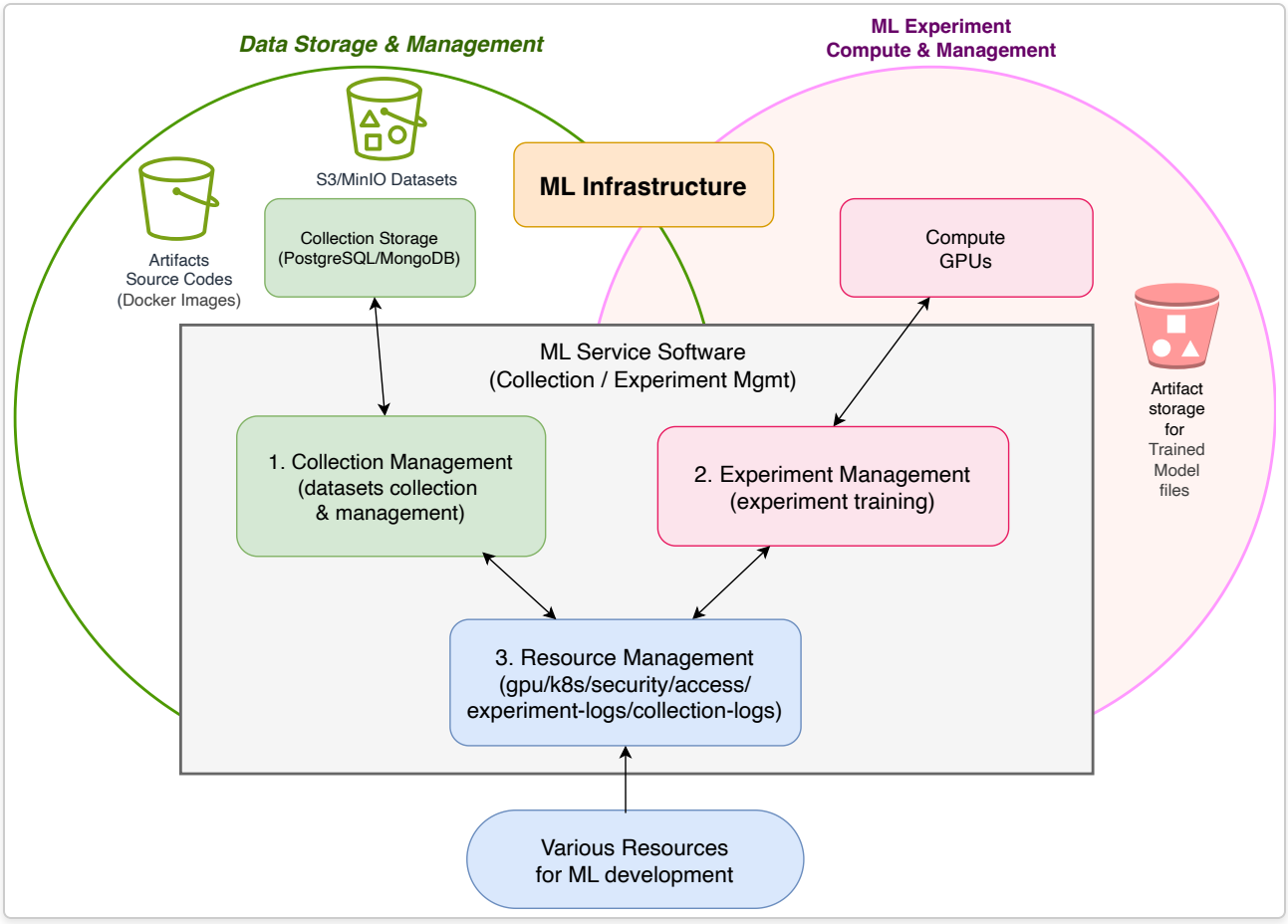


Figure 2: Architecture

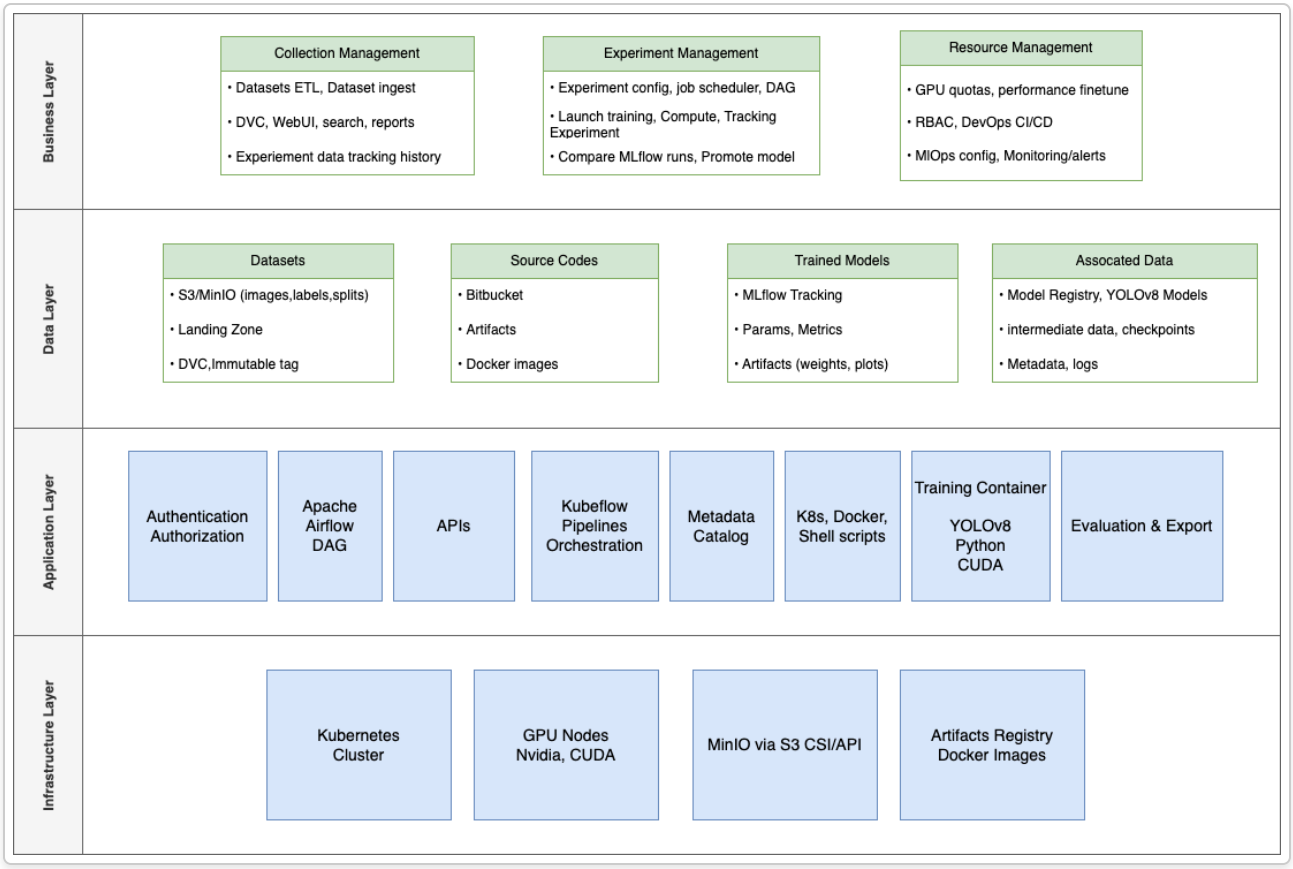
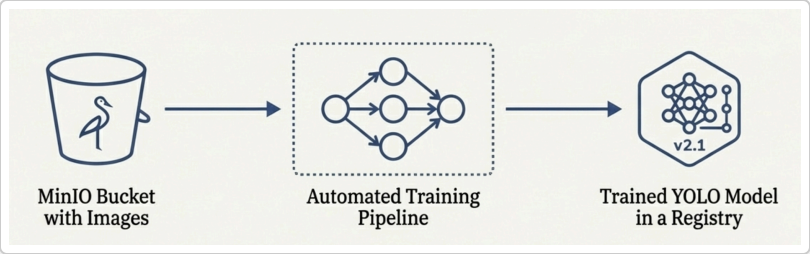


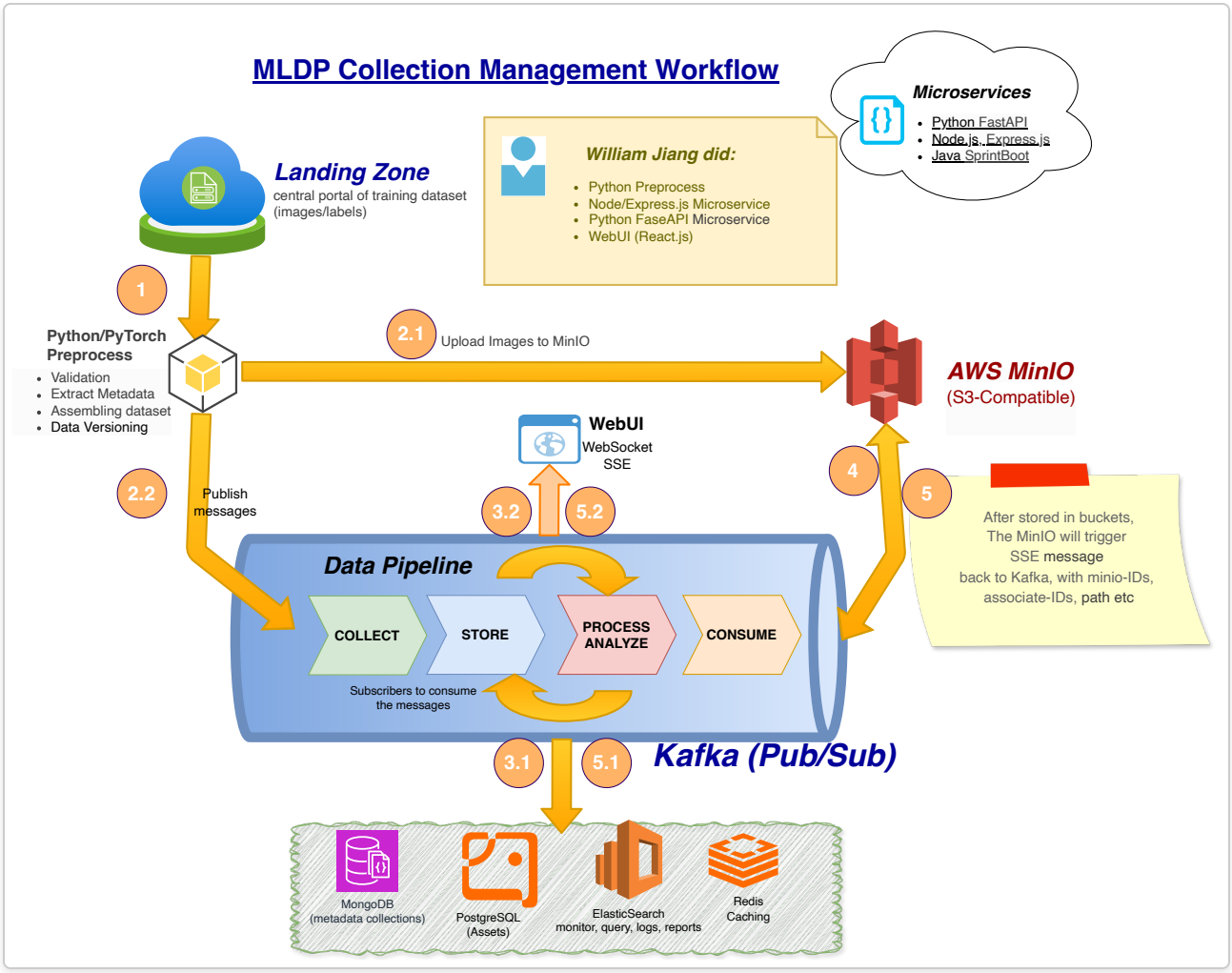
Figure 3: End-to-end overview - training data is collected and processed into datasets (left), fed into automated training pipelines (center), and results in a trained YOLO model registered for deployment (right).



1. Collection Management

This subsystem handles **data ingestion and dataset preparation**.

Figure 4: Collection Management Workflow



Data Ingestion & Preprocessing

- Raw driving data (**images and labels**) are uploaded to a **Landing Zone**, an internal portal for collecting new training data.
- A **Python ETL service** periodically scans the Landing Zone for new uploads and performs:
  - Image preprocessing and validation (e.g., using **PyTorch**)
  - Annotation parsing and metadata extraction
  - Data Versioning (DVC). **DVC (Data Version Control)** enforces dataset versioning and lineage tracking as new data is added.

Storage & Eventing

- Processed data images/labels are moved into **AWS S3/MinIO** object storage for long-term retention.
- Each successful upload triggers a **MinIO Server-Side Event (SSE)** notification.
- Events are published to **Apache Kafka** with dataset identifiers and storage locations to trigger downstream processing workflows (indexing, caching, and UI notifications).

Metadata & Indexing

Downstream **microservices** consume Kafka messages to update metadata and indexes

- Dataset metadata stored in **PostgreSQL (Assets database)**
- Collection and annotation data stored in **MongoDB**
- Frequently accessed metadata cached in **Redis**
- Records indexed in **Elasticsearch** for search and analytics

Collection Management UI

A WebUI (React + Node.js) provides:

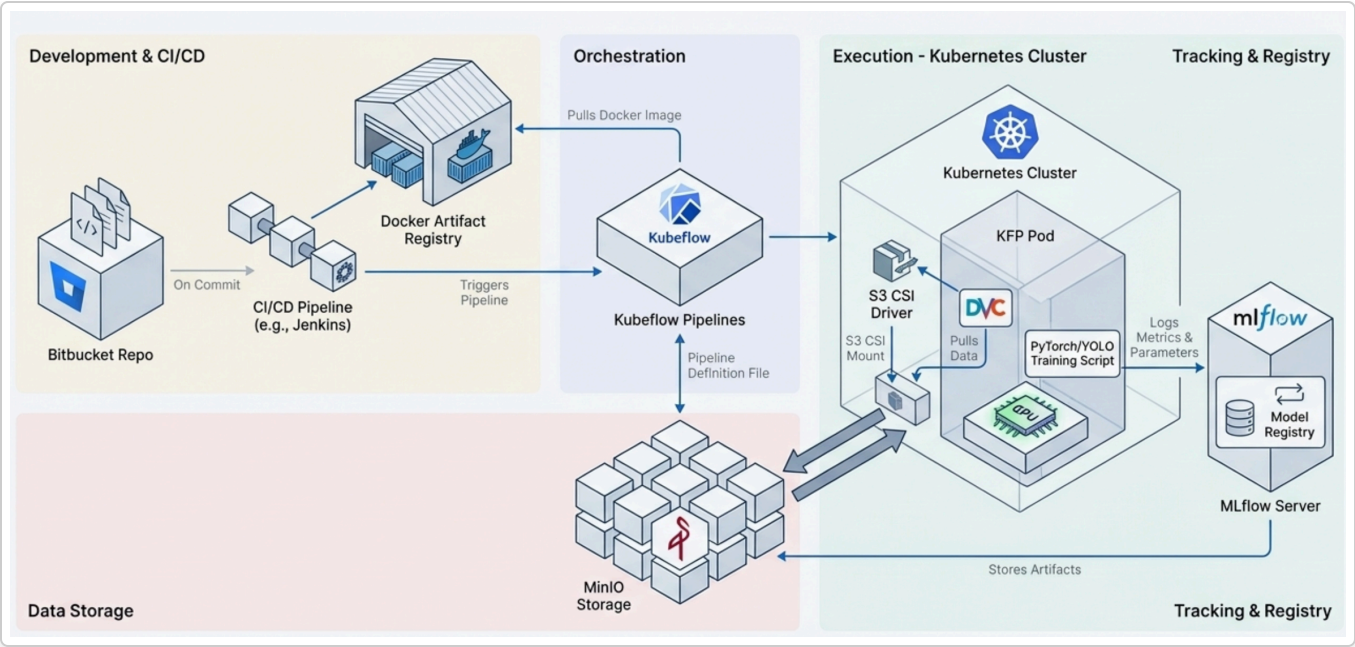
- Dataset collections (owners, categories, version tags, relationships with experiment data)
- Asset views (image previews, labels, metadata)
- Ingestion and job status monitoring
- RESTful APIs to upload supplementary data (e.g., edge cases, validation sets) directly to MinIO to augment existing collections
- Web forms for uploading additional metadata (e.g., sensor calibration, weather conditions) to enrich dataset context
- Real-time ingestion progress and system logs streamed via **WebSocket / SSE**, enabling immediate feedback for data engineers

The Collection Management subsystem ensures raw driving data is **efficiently collected, validated, stored, indexed, and versioned**, producing high-quality training datasets with **full traceability** for downstream ML workflows.

2. Experiment Management

Experiment Management covers the **model training workflows**, from orchestration to tracking and model governance. It uses **Kubeflow Pipelines** orchestrator over K8S.

Figure 5: Experiment Management Workflow



- The platform adopts **Kubeflow Pipelines (KFP)** to orchestrate end-to-end ML workflows.
- Kubeflow Pipelines are **Kubernetes-native** and better aligned with the full ML lifecycle.
- Pipelines are composed of custom components for each stage:
  - Data preparation
  - Training
  - Evaluation
  - Model export
- When a new experiment runs:
  - Datasets are pulled from **MinIO**
  - Training code is retrieved from pre-built **container images**
  - An **S3 CSI driver** is often used to mount MinIO buckets directly into pods for efficient data access

Training Execution & Tracking

- Training steps execute in isolated Kubernetes pods, leveraging **NVIDIA GPUs**.
- All experiment results (metrics, parameters, artifacts) are logged to **MLflow**.
- MLflow records run metadata and stores trained model artifacts for traceability and comparison.

CI & Reproducibility

- A **CI pipeline** (e.g., **Jenkins**) builds and version-tags Docker images for training jobs.
- This ensures **reproducible and consistent training environments** across experiments and environments.

Output & Summary

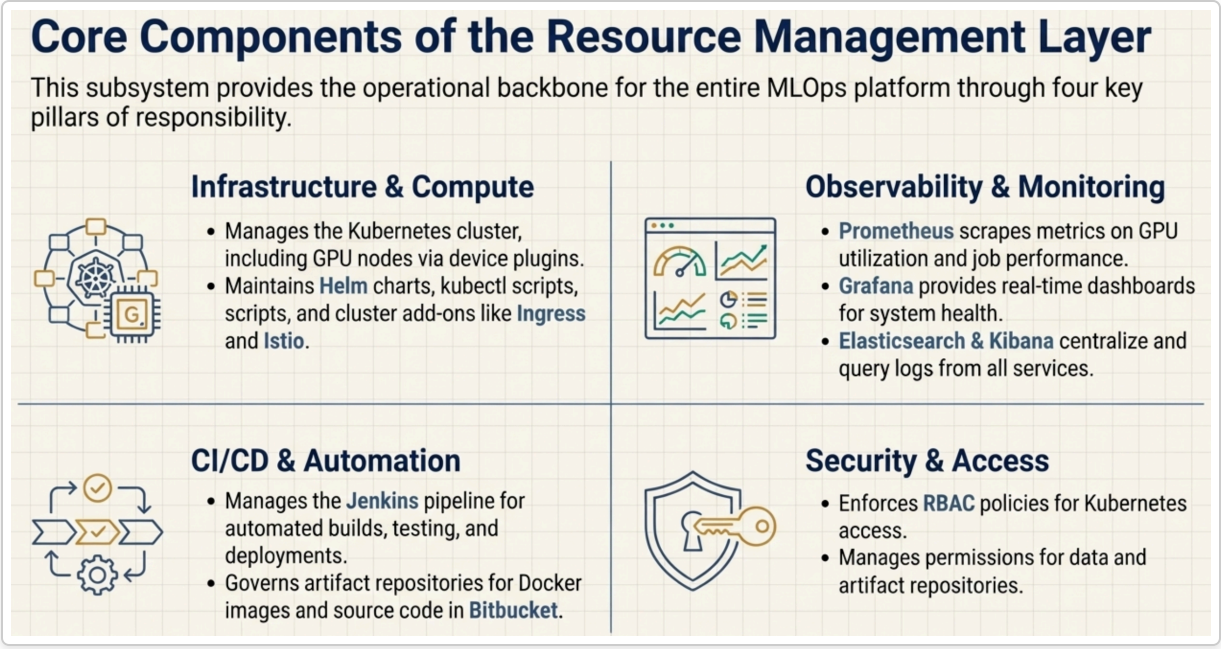
- Each experiment produces:
  - A registered model in the **MLflow Model Registry**
  - Associated metrics and metadata
- Models are ready for deployment or further evaluation.

Experiment Management provides a structured framework to **develop, orchestrate, and track model training** from code commit to trained model, supporting both **ad-hoc experimentation** and **reliable production pipelines**.

3. Resource Management

Resource Management oversees the **infrastructure and operational tooling** that supports both data collection and experiment pipelines.

Figure 6: Resource Management



Area	Components	Purpose
Infrastructure	Kubernetes cluster, GPU nodes, Helm charts, Istio	Core execution platform for training and data processing
Platform Services	Kubeflow, MLflow, Collection APIs	ML workflow orchestration and experiment tracking
Storage & Artifacts	MinIO, Bitbucket, Container Registry	Datasets, source code, and container images

Area	Components	Purpose
Monitoring	Prometheus, Grafana, Elasticsearch, Kibana	Metrics collection, visualization, and log analysis
CI/CD	Jenkins	Automated build, test, and deployment
Security	RBAC, Access Policies	Cluster access control and data permissions

Resource Management delivers **infrastructure stability, observability, security, and automation** for the platform.

What William Jiang Did

Collection Management (Primary)

- Built Python ETL pipelines for image and label ingestion
- Python FastAPI for connection between microservices
- Developed backend APIs for dataset management
- Implemented Collection Management Web UI

Resource Management

- Built backend APIs for resource reporting
- Managed Kubernetes infrastructure and CI/CD pipelines
- Set up monitoring with Prometheus and Grafana

Experiment Management (Supporting)

- Integrated Kubeflow Pipelines for training workflows
- Used MLflow for experiment tracking
- Supported experiment analysis