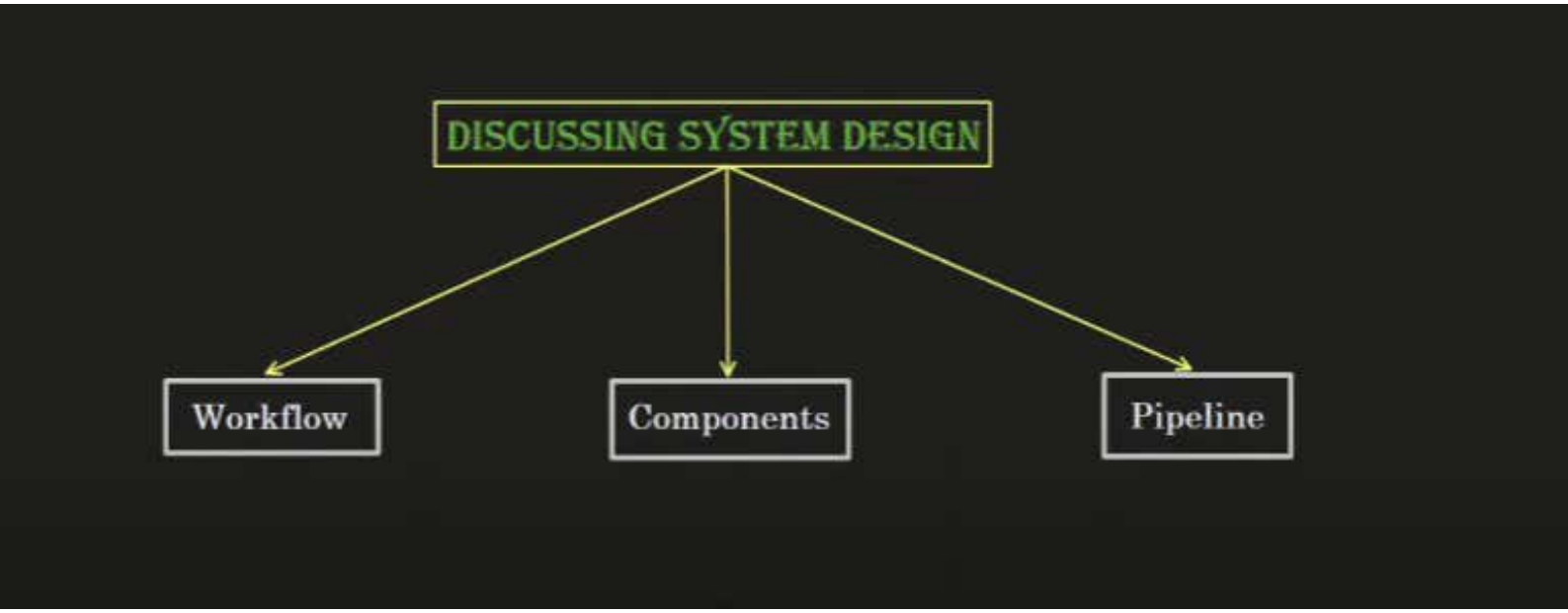
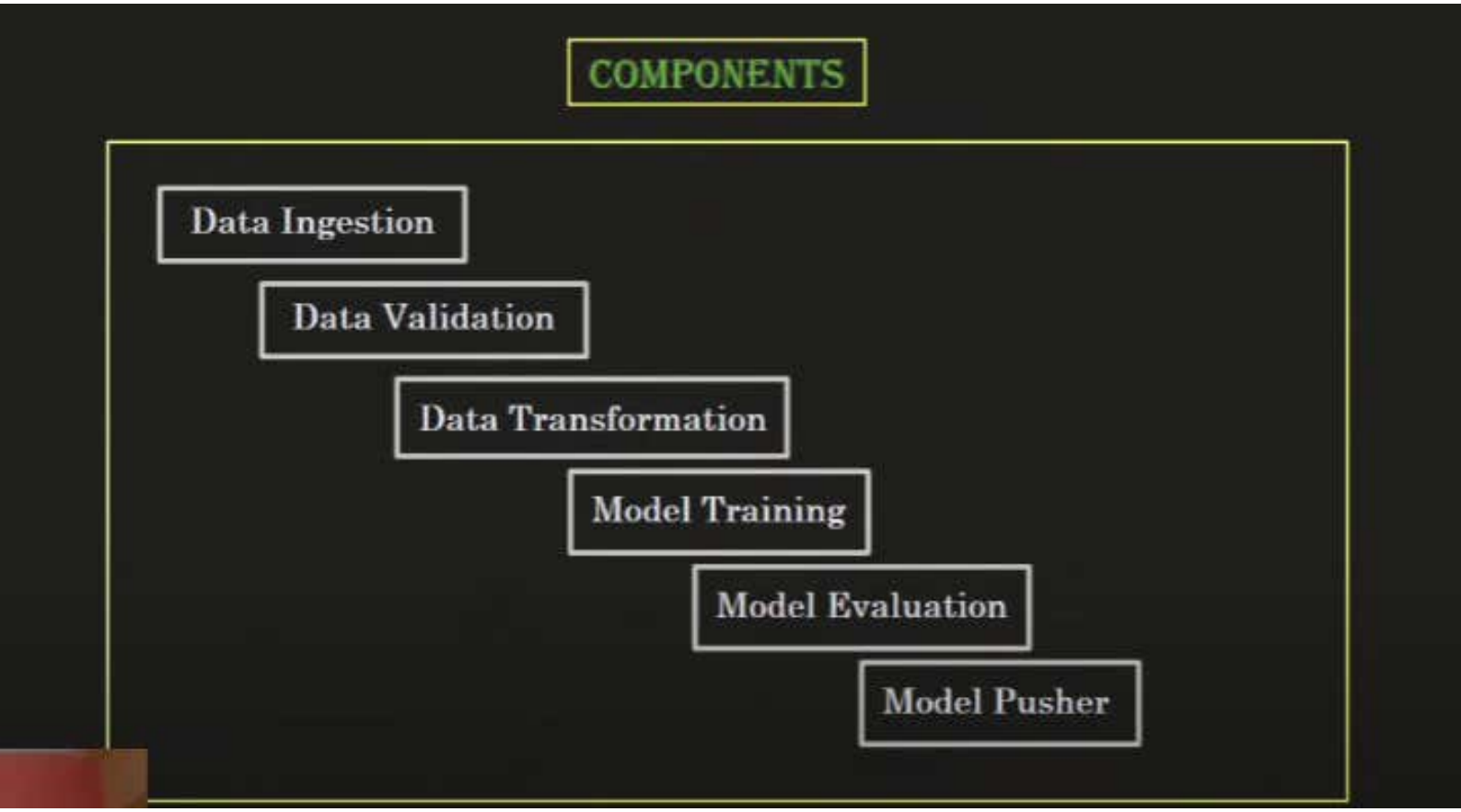
MLOPS-PROJECT-INTRODUCTION

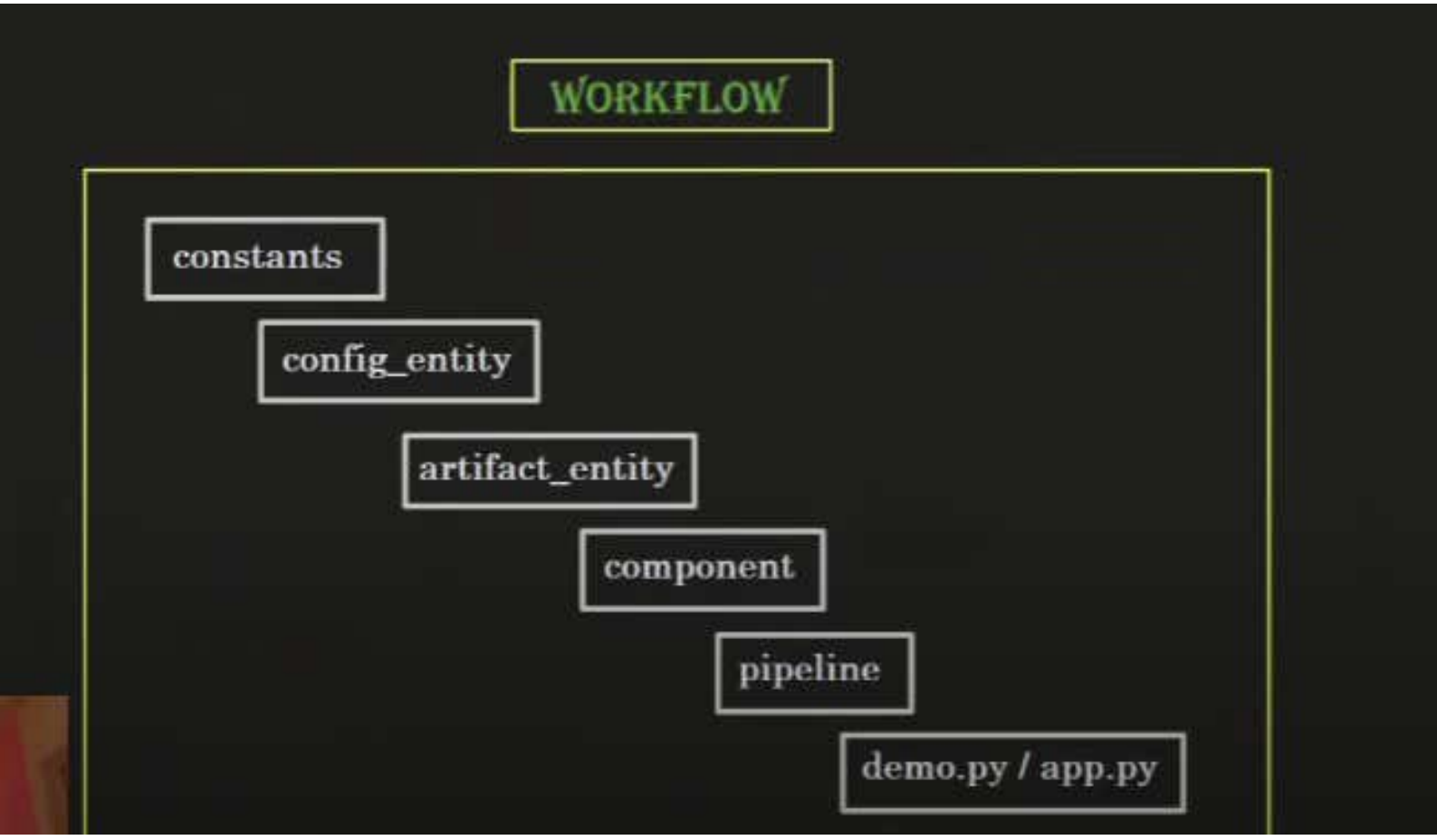
- Project1 - Vehicle Insurance Domain
- Heavy on MLOps instead of Data Science.
- Strong exposure to SDE.
- Coverage end to end.
- Reusable template format.
- Various crash courses.
- Expect difficulties (a lot).

TOOLS & TECHNIQUES USED

- MongoDB
- Robust pipeline - Training/Prediction.
- FastAPI
- CICD - Github Actions.
- AWS - IAM, ECR, EC2.
- No conventional mlops tools used.


DISCUSSING SYSTEM DESIGN
Workflow
Components
Pipeline


COMPONENTS
Data Ingestion
Data Validation
Data Transformation
Model Training
Model Evaluation
Model Pusher


WORKFLOW
constants
config_entity
artifact_entity
component
pipeline
demo.py / app.py

Data Stores: A Theoretical Overview

Data stores are fundamental components of any modern application or data architecture. They are specialized systems designed to store, manage, and retrieve data efficiently. The choice of data store depends heavily on the nature of the data, the required access patterns, scalability needs, consistency requirements, and cost considerations.

Here's a theoretical expansion of the ones you listed:

1. SSMS (SQL Server Management Studio)

- **What it is (Theoretically):** SSMS is not a data store itself. It is a **client-side administrative and development tool** that provides a graphical user interface (GUI) for interacting with and managing **Microsoft SQL Server databases**. Think of it as a sophisticated remote control for SQL Server.
- **Underlying Data Store Theory:** SSMS primarily connects to and manages **Relational Database Management Systems (RDBMS)**, specifically Microsoft SQL Server.
    - **Relational Model:** RDBMS are based on the relational model, where data is organized into tables (relations). Each table has a defined schema (columns) and rows (records).
    - **Structured Query Language (SQL):** Interaction with the data is primarily through SQL, a declarative language used for defining, manipulating, and querying data.
    - **ACID Properties:** RDBMS typically strongly enforce ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data integrity and reliability, especially in transactional workloads.
    - **Schema-on-Write:** Data must conform to a predefined schema before it is written to the database. This provides strong data integrity but can be less flexible for rapidly evolving data structures.
    - **Normalization:** Data is often normalized to reduce redundancy and improve data integrity.
- **Key Characteristics:**
    - **Transactional:** Excellent for OLTP (Online Transaction Processing) workloads where high concurrency, data integrity, and complex transactions are critical.
    - **Strict Consistency:** Ensures that data is always in a valid state.
    - **Scalability:** Traditionally scales vertically (more powerful server), but modern SQL Server versions offer horizontal scaling options (e.g., Always On Availability Groups, sharding).
    - **Management Features:** SSMS provides tools for database design, querying, debugging, performance tuning, security management, backups, and more.
- **Example Use Cases:** Financial systems, inventory management, e-commerce transaction processing, CRM (Customer Relationship Management) systems.

2. S3 Bucket (Amazon S3 - Simple Storage Service)

- **What it is (Theoretically):** Amazon S3 is an **object storage service**. Unlike traditional file systems or databases, S3 stores data as "objects" within "buckets." An object is a flat file (of any type: images, videos, backups, documents, log files, data lake files, etc.) combined with its metadata (e.g., creation date, content type, custom properties) and a unique key (its name/path within the bucket).
- **Underlying Data Store Theory:** S3 adheres to an **object storage model**.
    - **Flat Address Space:** There's no traditional hierarchical file system structure (directories within directories, though objects can have prefixes that mimic folders). Each object has a unique identifier within a bucket.
    - **Eventual Consistency (for some operations):** While many S3 operations are strongly consistent, some (like overwriting an existing object) might exhibit eventual consistency, meaning it might take a short time for the change to propagate globally.
    - **HTTP/REST API:** All interactions with S3 are typically done via HTTP/REST APIs, making it highly accessible from virtually any programming language or tool.
    - **Highly Distributed:** Data is replicated across multiple devices and facilities within an AWS region for high availability and durability.
- **Key Characteristics:**
    - **Massive Scalability:** Designed for virtually unlimited storage capacity. You don't provision storage; you just put objects in.
    - **High Durability:** Data is replicated across multiple availability zones (AZs) within a region, offering 99.999999999% (11 nines) durability.
    - **Cost-Effective:** Pay-as-you-go pricing, often tiered based on storage class (Standard, Infrequent Access, Glacier, etc.).
    - **Serverless Integration:** Integrates seamlessly with AWS Lambda and other serverless services.
    - **Static Website Hosting:** Can directly host static websites.
- **Example Use Cases:** Data lakes, backups and archives, serving static content (images, videos), big data analytics source, disaster recovery, cloud-native application storage.

3. Amazon Redshift

- **What it is (Theoretically):** Amazon Redshift is a fully managed, petabyte-scale **cloud data warehouse service**. It's designed specifically for analytical workloads, not for day-to-day transactional processing.
- **Underlying Data Store Theory:** Redshift is a **Massively Parallel Processing (MPP) columnar database** built on PostgreSQL.

    - **Columnar Storage:** Instead of storing data row-by-row (like traditional OLTP databases), Redshift stores data column-by-column. This is highly optimized for analytical queries that often read specific columns across many rows (e.g., calculating the sum of sales for a year). It allows for much higher compression and faster query execution for aggregated queries.
    - **MPP Architecture:** It distributes data and query processing across multiple compute nodes, allowing for parallel execution of complex analytical queries.
    - **Optimized for OLAP (Online Analytical Processing):** Designed for complex queries involving large aggregations, joins, and filters over vast datasets.
    - **Semi-Structured Data Support:** While primarily relational, it has features to handle semi-structured data (e.g., JSON) using specific data types.
- **Key Characteristics:**
    - **Analytical Focus:** Not suitable for high-volume, low-latency transactional writes.
    - **Scalability:** Scales horizontally by adding more compute nodes.
    - **Cost-Effective for Analytics:** More economical than traditional data warehouses for large datasets, especially as data volume grows.
    - **Data Compression:** Columnar storage allows for significant data compression, reducing storage costs and improving query performance.
    - **Workload Management:** Provides features to manage query priorities and resource allocation.
- **Example Use Cases:** Business intelligence (BI) reporting, big data analytics, operational analytics, consolidating data from multiple sources for insights.

4. MongoDB

- **What it is (Theoretically):** MongoDB is a popular **NoSQL (Not only SQL) document-oriented database**. It diverges from the traditional relational model by storing data in flexible, JSON-like documents.
- **Underlying Data Store Theory:** MongoDB follows a **document database model**.
    - **Document-Oriented:** Data is stored as BSON (Binary JSON) documents. Each document is a self-contained unit and can have a different structure from other documents in the same collection.
    - **Collections:** Documents are grouped into collections, which are analogous to tables in relational databases, but without rigid schemas.
    - **Schema-on-Read (or Schema-less):** There's no predefined schema. You can insert documents with varying fields, and the application interprets the schema when reading. This offers immense flexibility for evolving data models.
    - **Eventually Consistent (Configurable):** MongoDB offers various consistency levels, from eventual to strong, allowing developers to choose the right balance between consistency and availability/performance based on application needs.

    - **Horizontal Scalability (Sharding):** Designed for horizontal scaling through sharding, where data is distributed across multiple servers (shards) to handle high write and read loads.

- **Key Characteristics:**
    - **Flexibility:** Excellent for rapidly evolving data models and agile development.
    - **Scalability:** Highly scalable horizontally, making it suitable for large and growing datasets.
    - **High Performance:** Optimized for specific use cases, often fast for reads and writes within a single document.
    - **Rich Query Language:** Provides a powerful query language for filtering, sorting, and aggregating documents.
    - **Denormalization:** Data is often denormalized (duplicated) within documents to optimize read performance and reduce the need for joins.
- **Example Use Cases:** Content management systems, mobile applications, real-time analytics, personalization engines, IoT (Internet of Things) data, catalog management.

Each of these data stores offers distinct advantages and trade-offs, making the choice dependent on the specific requirements of the application and its data.