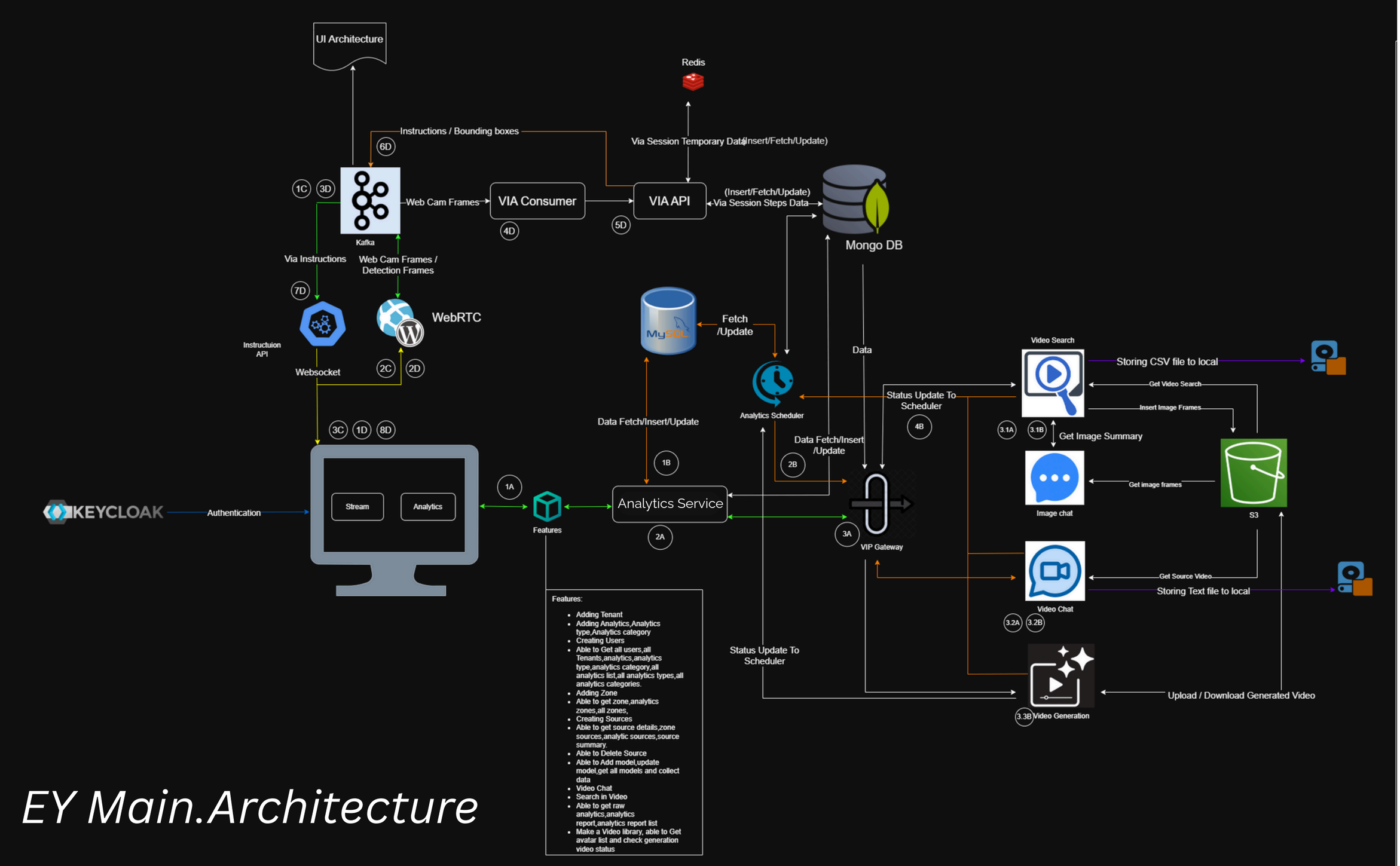
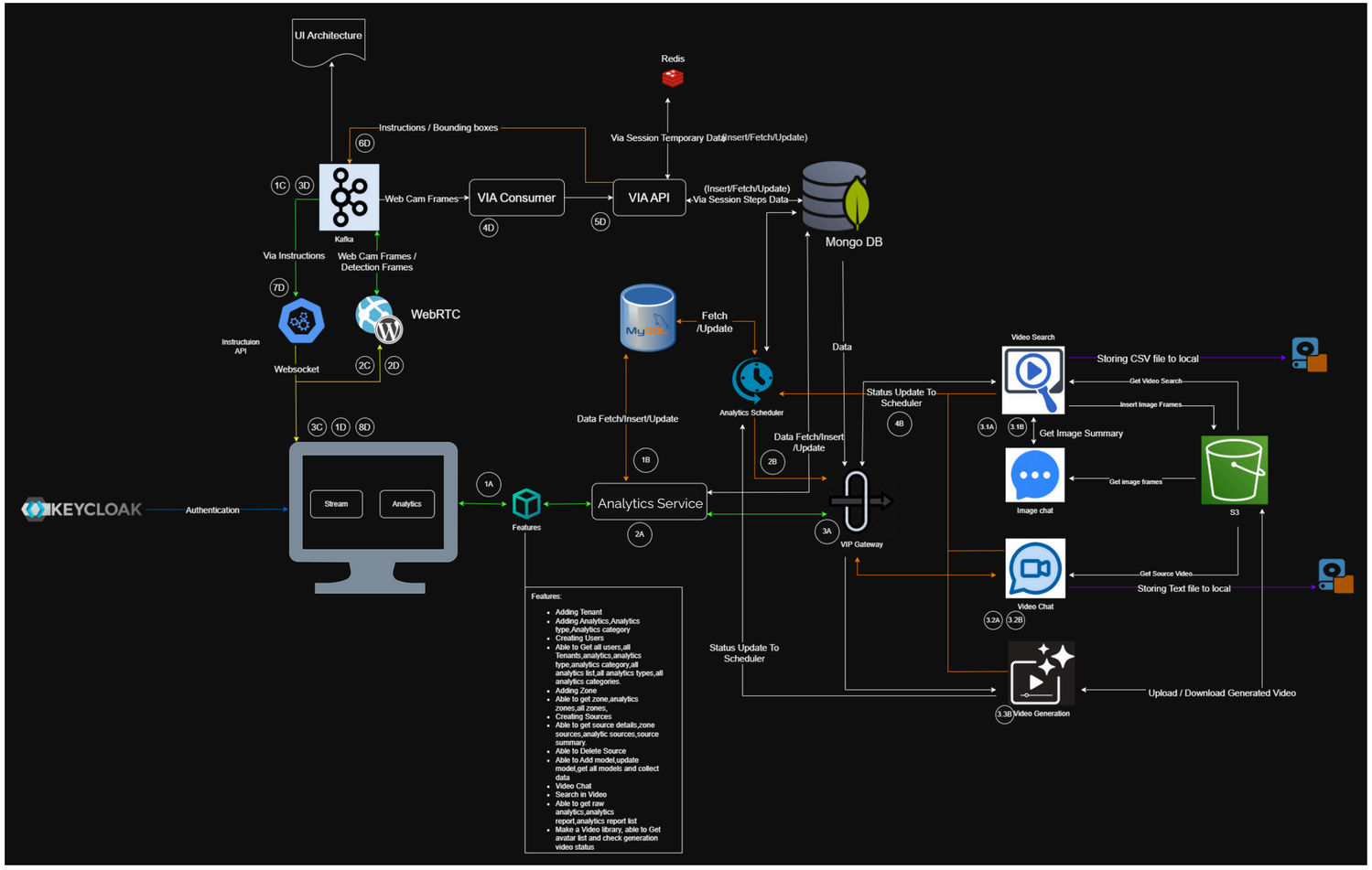
Presented by
Nikhil Akella and Pattabhi Rama
29/5/25

Next Slide



EY Main.Architecture



KEY

Process A – Data Flow

- 1A. The user initiates a request via the VIP UI (e.g., analytics, video summary, chat, or generation).
- 2A. The Analytical Service receives the request and determines what data is needed.
- 3A. The Analytical Service interacts with the VIP Gateway to handle video-specific operations.
- 3.1A. For Video Search, the VIP Gateway routes the request to the Video Search service.
- 3.2A. For Video Chat, it routes the request to the Video Chat service.

Process B – Scheduler & Backend Flow

- 1B. When a new source is added in VIP or a video generation request is made, the request and source details are stored in the SQL Database.
- 2B. The Scheduler periodically polls the SQL Database to check for:
- New entries,
- Failed requests,
- And the status of video generation, video chat, and video search requests.
- 3.1B. For Video Search, if pending, the Scheduler sends a request to the VIP Gateway, which then forwards it to the Video Search Service.
- 3.2B. For Video Chat, the Scheduler sends a request through the VIP Gateway to the Video Chat Service.
- 3.3B. For Video Generation, the Scheduler sends the request through the VIP Gateway to the Video Generation Service.
- 4B. Once each service completes the requested operation, it returns a status update to the Scheduler, which then updates the SQL Database.

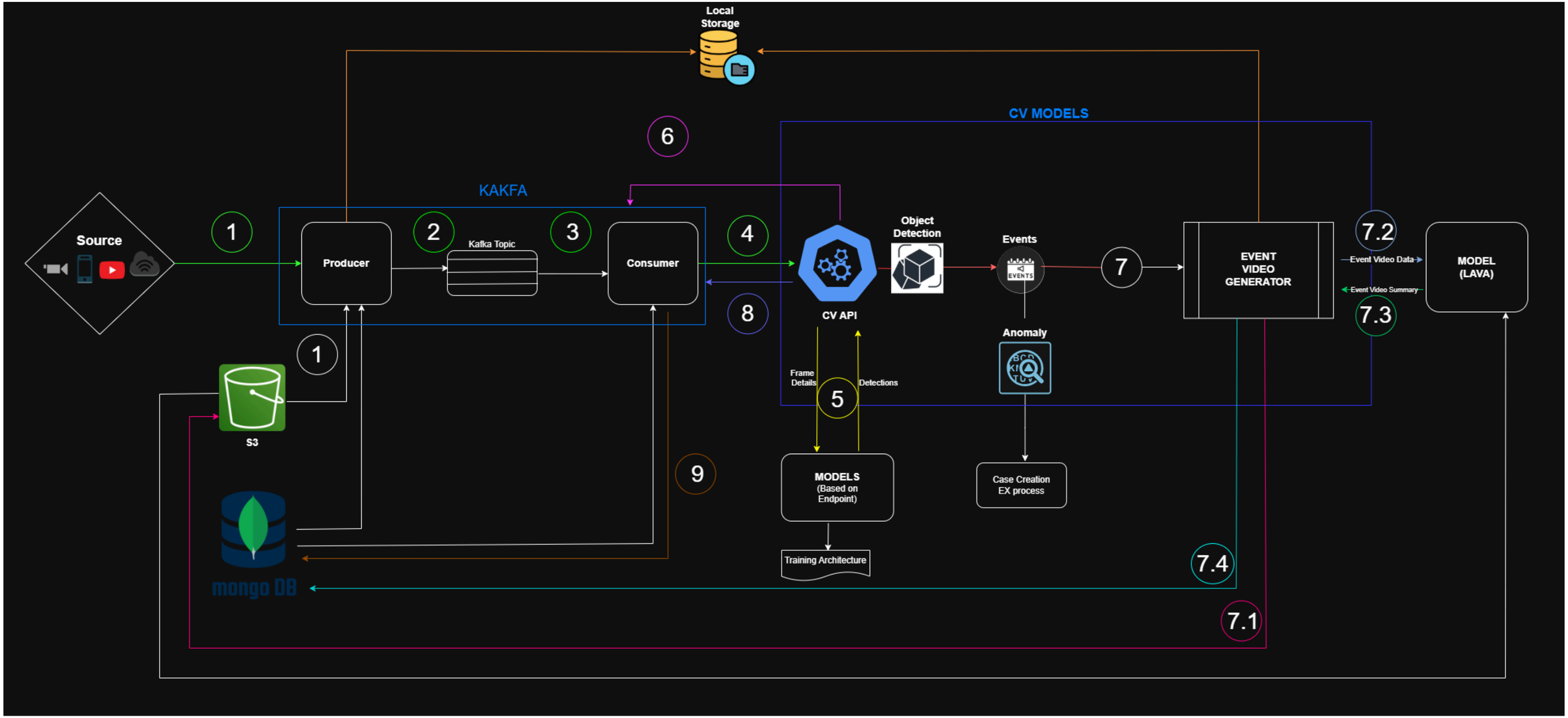
Process C

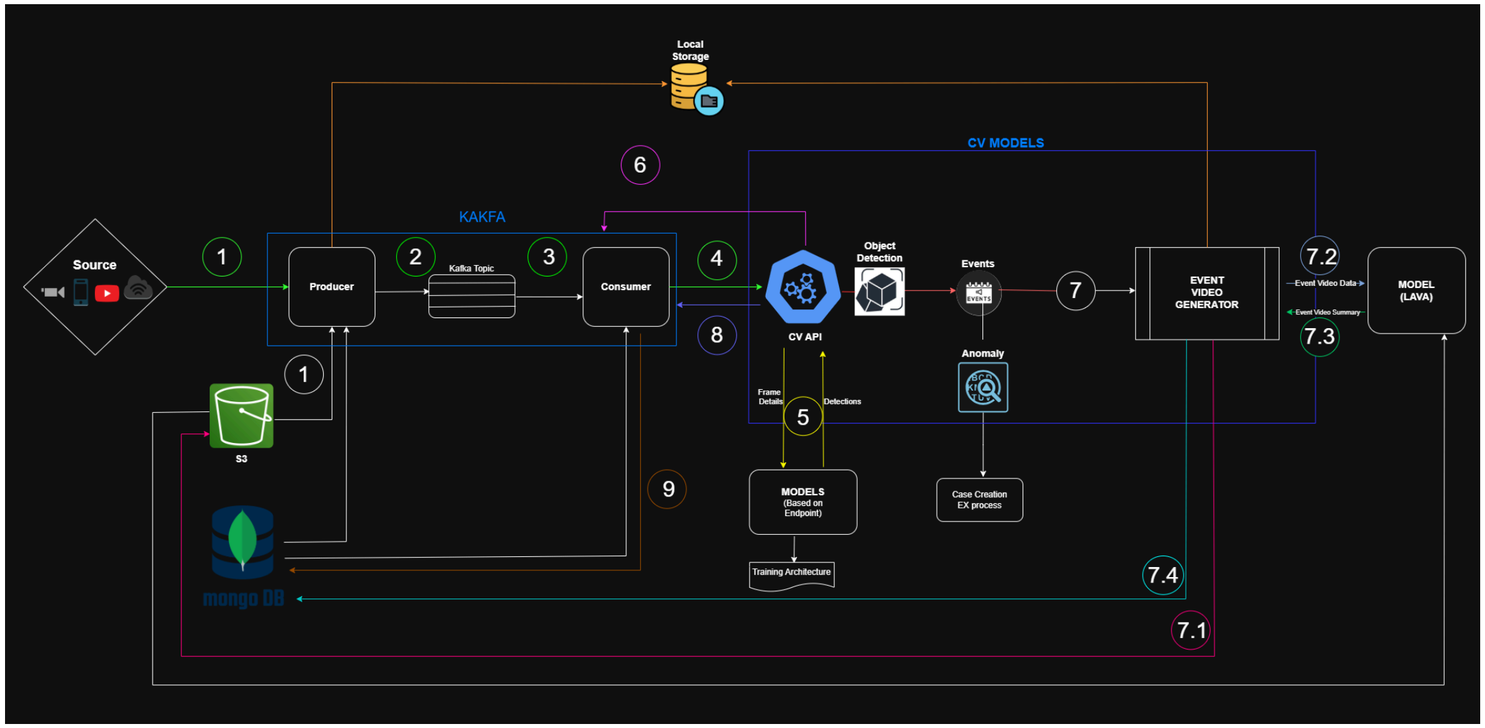
- 1C. The WebRTC Service subscribes to the Kafka Stream Topic, which carries detected frames from the EY Backend system.
- 2C. Once frames are received, the WebRTC Service processes them and prepares them for UI-compatible delivery.
- 3C. The WebRTC Service streams the processed video frames to the VIP UI, enabling live stream playback for the user.

Process D – Manual VIA Flow

- 1D. The user initiates a Manual VIA session from the VIP UI.
- 2D. The user's webcam frames are sent to the WebRTC service.
- 3D. WebRTC sends these frames to Kafka (for backend processing).
- 4D. The VIA Consumer listens to Kafka, receives the frames, and forwards them to the VIA API.
- 5D. The VIA API performs inference or calculations based on the manual type.
- 6D. The processed frames with detection results are published back to Kafka.
- 7D. The Instruction API (blue box) reads the instruction data from Kafka.
- 8D. The WebRTC sends the updated video frames to the VIP UI, while the Instruction API sends instructions to the VIP UI for user interaction.

EY Backend Architecture





KEY COMPONENTS

- Source: Origin of video input (camera, mobile, YouTube, S3, etc.).
- Producer: Extracts frames from video, adds metadata, and sends data for processing.
- MongoDB: Stores metadata, detections, and event info for APIs/UI.
- Kafka: Message queue for streaming frame data to consumers.
- Consumer: Listens to Kafka, sends frames for analysis, stores results.
- CV API: Interfaces between raw frames and ML models.
- Models: Detect objects, events, and activities in video frames.
- Event Video Generate: Creates short video clips from detected events.
- S3: Cloud storage for saving and retrieving event clips.
- Video Summary API: Summarizes event clips into short text outputs.
- Local Storage: Saves full raw video at the edge or device level.

DATA FLOW

1)Source (e.g. camera, mobile, YouTube, S3) gives the video stream which is captured by Producer.

2)Producer:

- If the source is video the store → Local Storage
- Receives source/model metadata and source type from MongoDB.
- Extracts frame bytes and sends to kafka Topic where partitioning takes place.

3)Consumer:

- Listens for frame bytes and metadata.
- Receives analytics/extracted text from CV API.
- Stores analytics & extracted data into MongoDB.

4)Consumer sends frame data to CV API for processing.

5)CV API forwards frame data to Models for visual inference.It will return objects detected . Using objects detected we get events.If anomaly occurs , a case will be created.

6)Models return:

- Frames with detections / bounding boxes for UI→ back to kafka

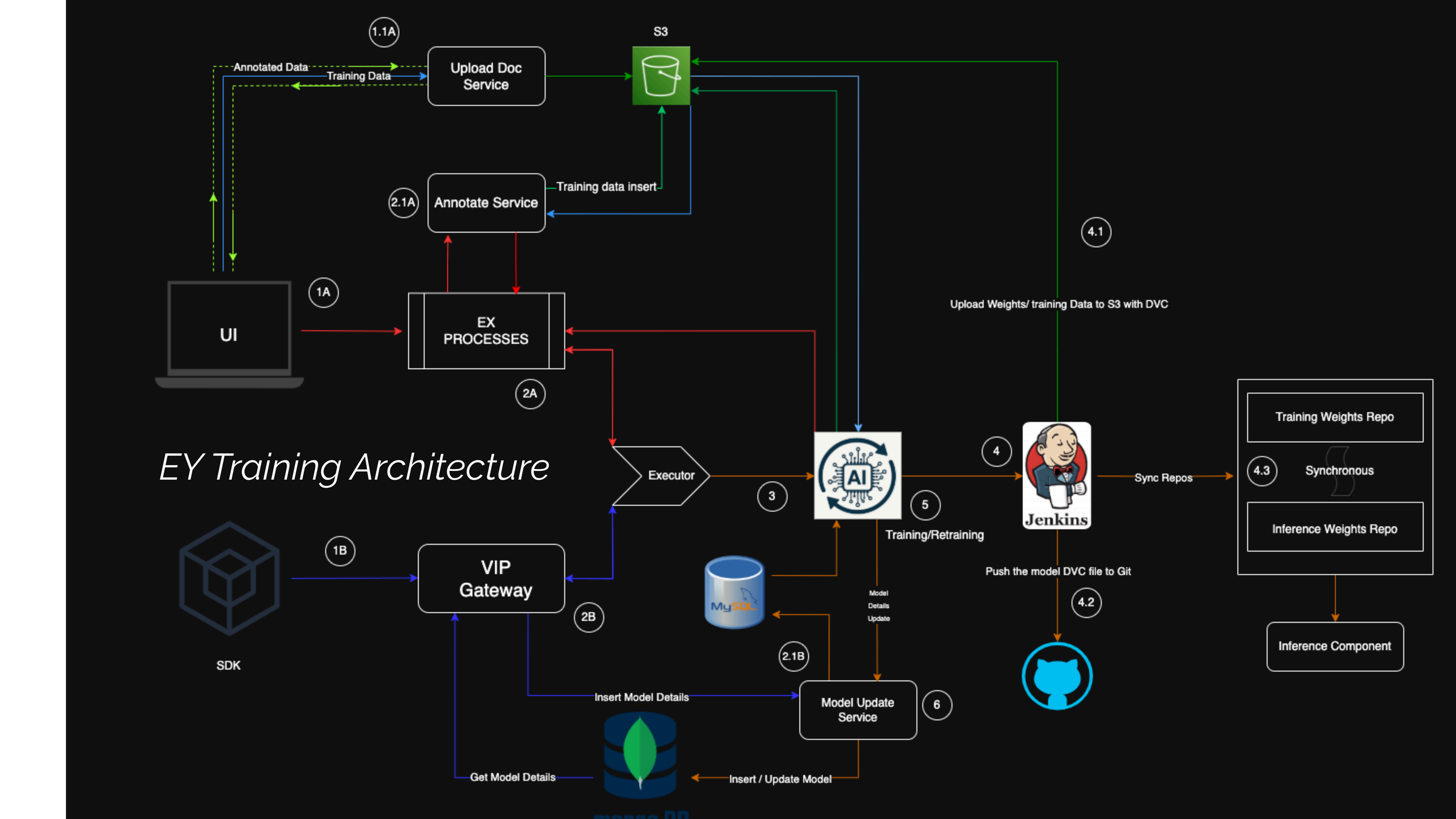
7)CV API:

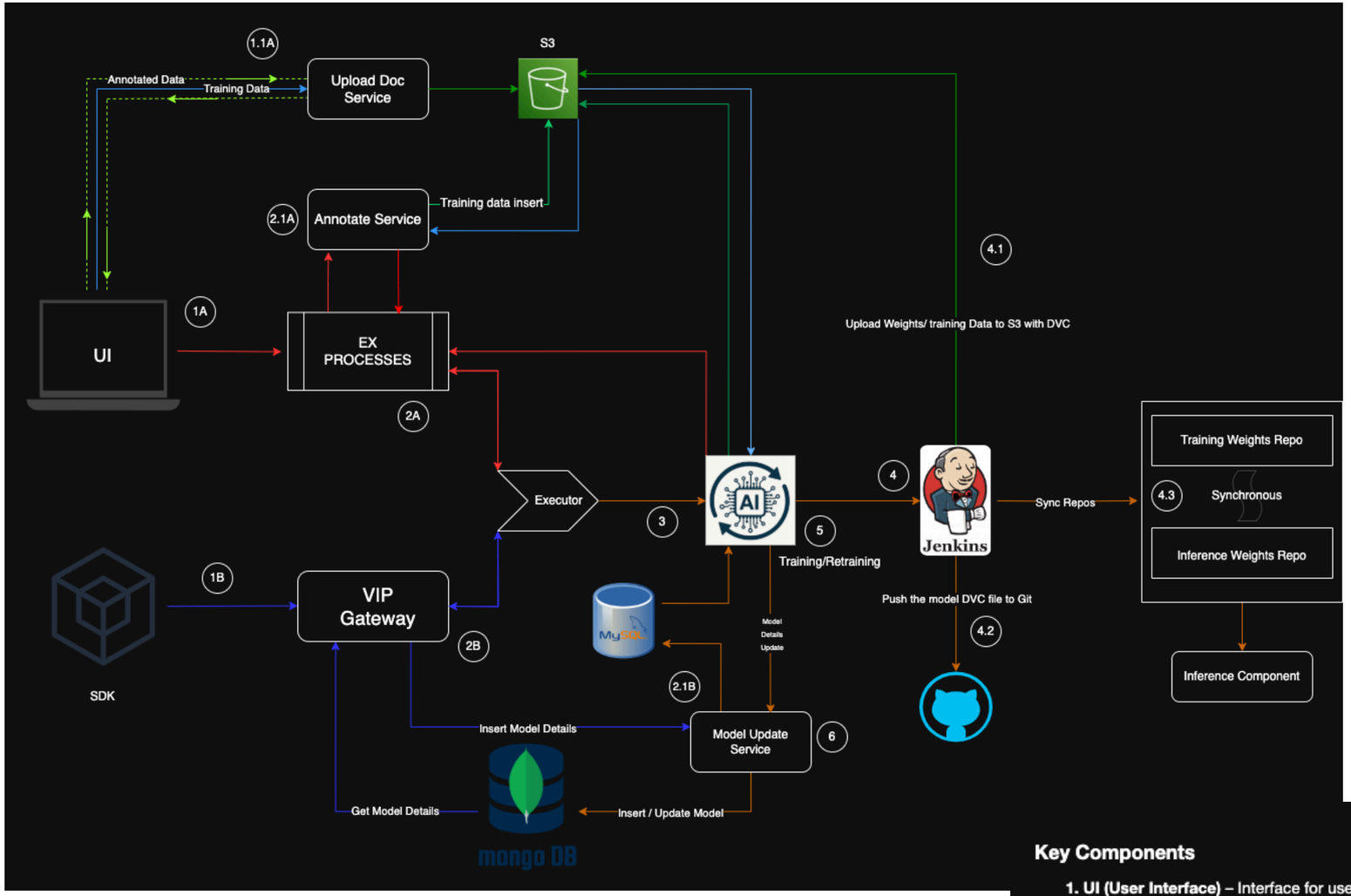
- Sends frame data → Event Video Generate
- Sends detections & bounding boxes → MongoDB for UI

7.1) Event Video Generator stores event video clips in S3
7.2) Video Summary API fetches event video data from S3
7.3) Video Summary API creates summaries and returns summary data
7.4) Event Video Generate inserts event video metadata into MongoDB

8)CV API sends analytics/extracted text → back to Consumer

9)Consumer inserts final analytics/processed results into MongoDB


EY Training Architecture
1.1A
S3
Annotated Data
Training Data
Upload Doc Service
2.1A
Annotate Service
Training data insert
4.1
Upload Weights/ training Data to S3 with DVC
UI
1A
EX PROCESSES
2A
Executor
3
AI
4
Jenkins
5
Training/Retraining
Sync Repos
Training Weights Repo
4.3
Synchronous
Inference Weights Repo
Inference Component
Push the model DVC file to Git
4.2
SDK
1B
VIP Gateway
2B
MySQL
Model Details Update
2.1B
Insert Model Details
Model Update Service
6
Get Model Details
Insert / Update Model



Data Flow Breakdown:

1A. From UI (User Interface)

- Users can annotate frames from a live video stream or upload a ZIP file containing training data.
- This request is sent to the **EX Process**.
  - **1.1A**: The annotated frames or training ZIP files are then stored in **AWS S3**.

2A. From EX Process

- The **EX Process** triggers the model's training or retraining using an **Executor**.
  - **2.1A**: If the input is annotated frames, they first go through the **Annotation Service** to be converted into a proper dataset format.

1B. From SDK

- An SDK client provides a link to a dataset ZIP file (already stored in S3) to the **VIP Gateway**.

2B. VIP Gateway

- The **VIP Gateway** triggers the training or retraining process using the **Executor**.
  - **2.1B**: Before doing this, the gateway saves model information to **MySQL** and/or **MongoDB**.

3. Executor

- The **Executor** launches the training process **asynchronously**, meaning it doesn't wait for it to complete and continues processing other tasks.

4. Jenkins Integration

- Once training completes, the **Training/Retraining Service** calls **Jenkins** to:
  - **4.1**: Upload the trained model weights and training data to **DVC** (backed by S3).
  - **4.2**: Create a DVC tracking file and push it to the **Git** repository.
  - **4.3**: Sync the **Training Weights Repo** and the **Inference Weights Repo** so that the inference system automatically picks up the latest trained weights.

5. Training Status

- **Jenkins sends the status update back to the Training/Retraining Service.**

6. Model Update

- Once status is received, the **Model Update Service** saves or updates model information in **MySQL/MongoDB** for tracking and usage.

Key Components

1. **UI (User Interface)** – Interface for users to annotate frames or upload training data.
2. **Upload Doc Service** – Uploads annotated or training data to AWS S3.
3. **S3 (Amazon Simple Storage Service)** – Cloud storage for datasets, model weights, and DVC files.
4. **EX Processes** – Handles input requests and initiates training/retraining workflows.
5. **Annotate Service** – Converts annotated images into structured dataset format.
6. **SDK** – Provides external access to trigger training using dataset links.
7. **VIP Gateway** – Mediates training requests and records model metadata.
8. **Executor** – Asynchronously starts model training or retraining jobs.
9. **Training / Retraining Service** – Manages the complete training lifecycle and workflow.
10. **Jenkins** – Automates model deployment, versioning, and repository syncing.
11. **DVC (Data Version Control)** – Tracks versions of datasets and model weights using Git and S3.
12. **Git Repository** – Stores DVC metadata files and versioned training configurations.
13. **Training Weights Repo** – Repository containing the latest trained model weights.
14. **Inference Weights Repo** – Repository used by the inference component to get model weights.
15. **Inference Component** – Uses the latest weights to perform model predictions.
16. **Model Update Service** – Updates model metadata in the database post-training.
17. **MongoDB / MySQL** – Databases that store metadata for models and training runs.