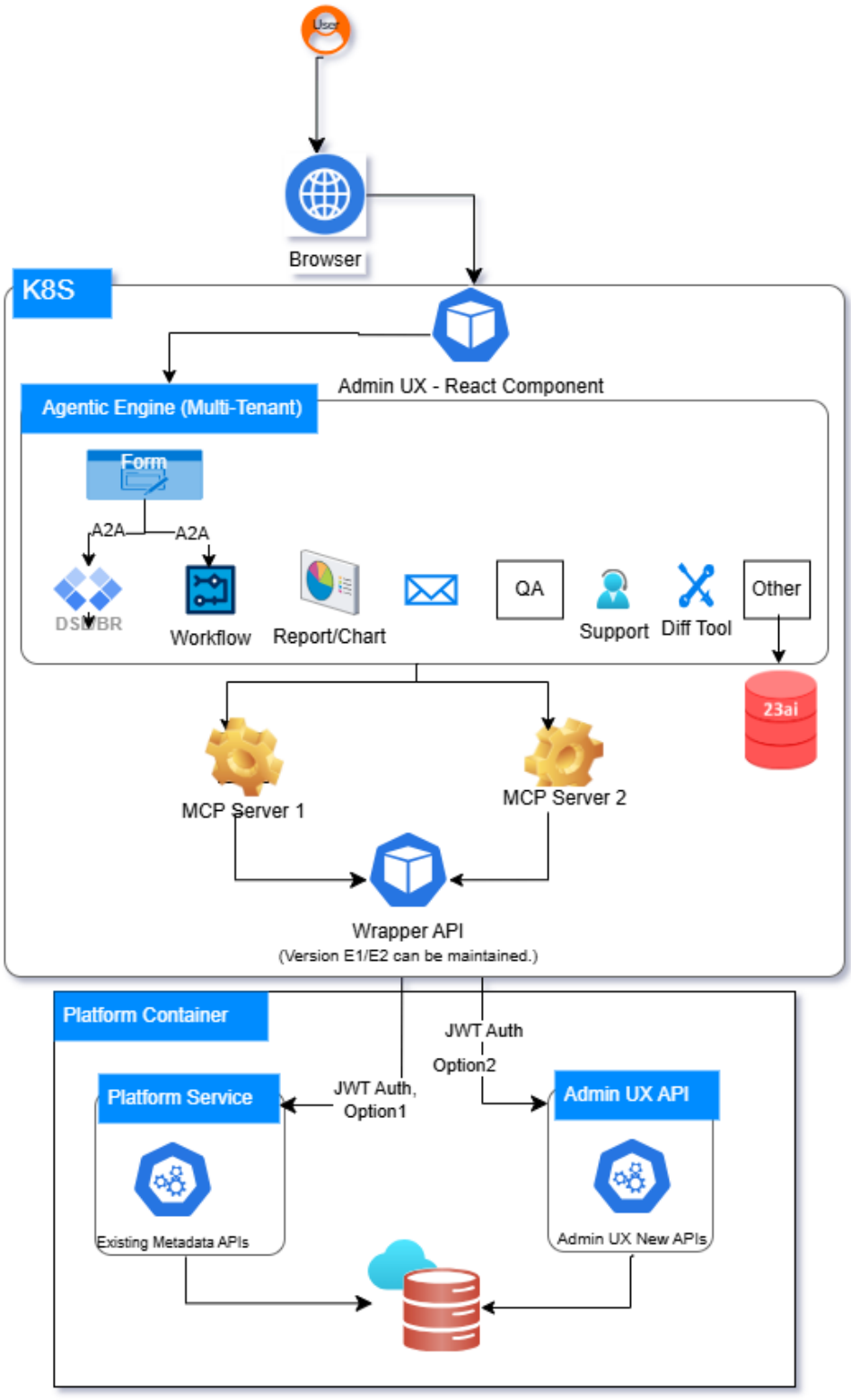
Agentic Solution - Architecture

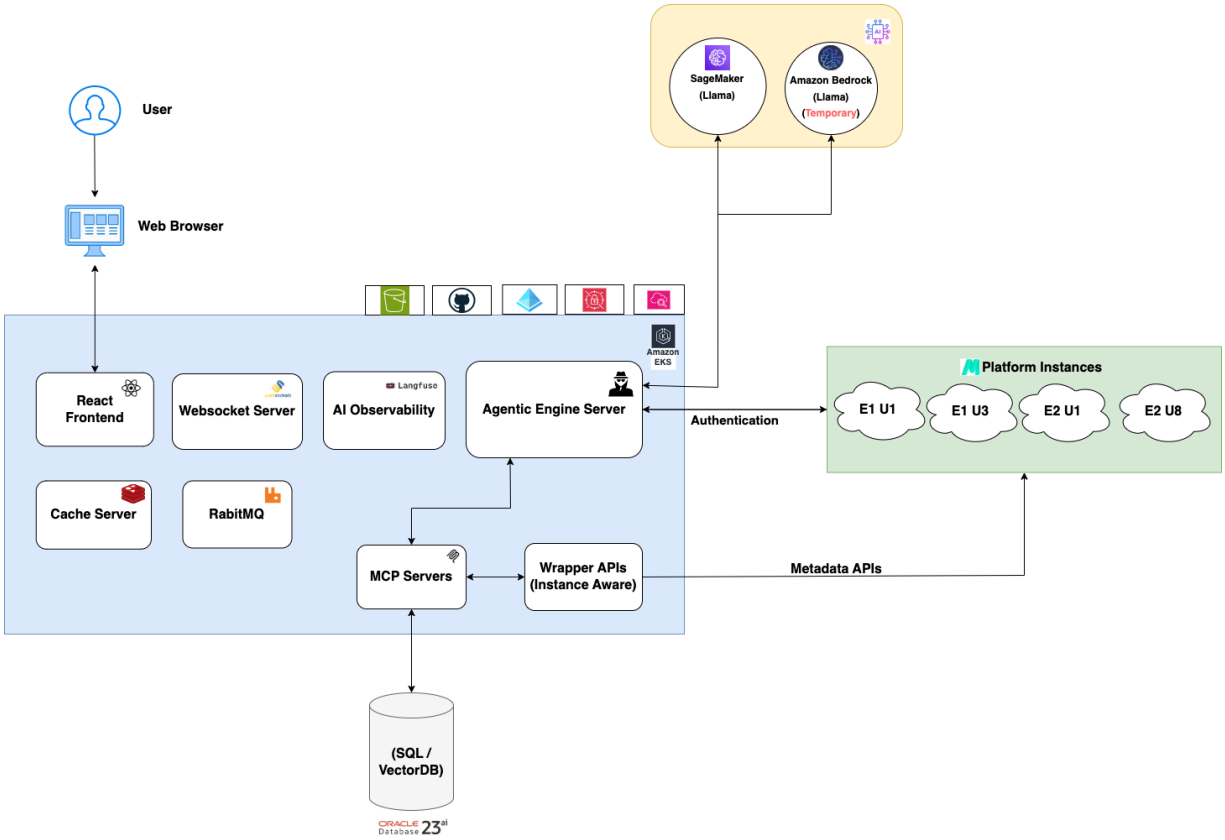
High Level Architecture



Option1: Wrapper API will connect to Platform APIs, which are existing in system and will work for all the use cases of the Agentic Solution.

Option2: Write all Wrapper APIs needed to reduce no. of rounds Agentic Solution will take to get result, copy the existing APIs from Platform and rewrite here (reducing the unnecessary sections and write performant code)

System Architecture - Components



System Architecture Overview

This architecture depicts a **modular, containerized GenAI-driven platform**, orchestrated via **Amazon EKS**, designed for **multi-instance, user-aware** operations, likely to support dynamic form handling, intelligent workflows, and generative AI-driven automation.

1. User Interaction & Frontend Layer

- The **User** interfaces with the system via a **React-based web application**, served to the **Web Browser**.
- Real-time interaction capabilities are provided through a **WebSocket Server**, enabling bi-directional event-driven communication (e.g., agent responses, notifications).
- Frontend acts as a thin client, delegating heavy lifting to backend services and the agentic layer.

2. Orchestration & Backend Logic (EKS Cluster)

At the heart lies the **Kubernetes-based application core**, where containerized microservices are deployed with clear separation of concerns:

a. Agentic Engine Server

- Acts as the **central orchestrator**, interpreting user intent and orchestrating downstream calls to internal services,

external model providers (LLMs), and MCP servers.

- Manages **contextual flow**, **state tracking**, and AI-driven decisions.
- Integrated with a **Kubernetes-native logging and observability stack** for traceability, debugging, and operational metrics.

b. Wrapper APIs

- Abstracts access to downstream dev instances (E1, E2, etc.) with **tenant- and user-level awareness**.
- Enforces **instance isolation**, **security boundaries**, and **data contextualization** per session/user.

c. MCP Servers

- Provides a standardized way of accessing data sources through AI agents. MCP servers will help us connect with the instances for fetching metadata through wrapper APIs.

d. Cache Layer

- Provides high-throughput low-latency access to frequently accessed state, session info, or agent context (e.g., Redis). It will help us cache the user session related data as well as prompts and other important info.

e. RabbitMQ

- Supports **decoupling of components** through asynchronous task queues—useful for background tasks, retries, or fanout operations across platform modules. This will help us handle retries in case there are failures. It will also help us fanout multiple jobs in parallel. RabbitMQ will have persistent storage attached to it.

3. AI/LLM Integration Layer

- The Agentic Engine leverages **multi-provider LLM routing**, integrating:
  - **Amazon SageMaker (LLaMA)** – SageMaker allows us to self-host the models we need. The pricing of SageMaker is hourly/monthly. This helps us steer away from token-based costing.
  - **Amazon Bedrock (LLaMA)** – Bedrock, though token based, is ideal for development purposes as it has no setup involved. We will be using Bedrock for phase 1 development.

4. Platform Instance Layer

- This represents the dev instances deployed by MetricStream for their respective clients.
- The primary use cases of the Agentic Solution (e.g., **Product Development** and **Upgrade**) are intended for the **developer community**.
  - In this flow, developers make changes in the **Dev environment**, which are then promoted to **STG → QA → UAT → Prod** through the standard process.
  - Since our **Dev environments are not on-prem**, the **on-prem scenario is not applicable** here.
  - Moreover, allowing on-prem usage—especially with Admin UX—could lead to **inconsistent change management** if changes are made in one environment and not replicated properly across others.

- To maintain governance and ensure auditability, **on-prem changes via Admin UX should not be allowed**.

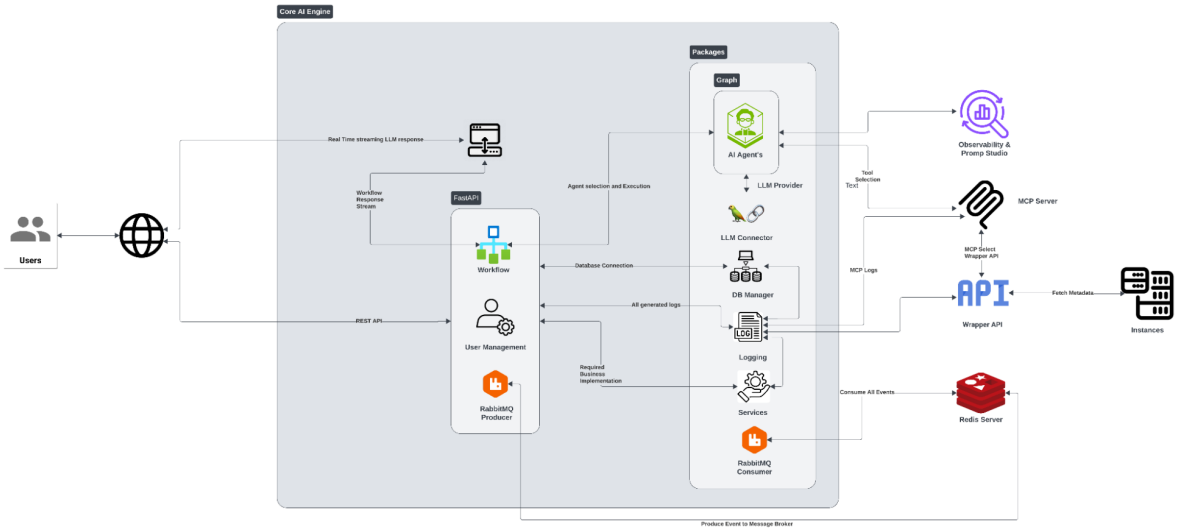
5. Persistence Layer

- Backed by **Oracle 23**, with capabilities for:
  - **Traditional SQL-based persistence**
  - **Vector storage** (suggesting embedding support for AI-driven similarity search, RAG, or rule retrieval)

6. AI Observability

- Integrated with **Langfuse**, ensuring:
  - Tracing, prompt/result capture, latency tracking
  - Provides accountability, reproducibility, and evaluation metrics for LLM outputs. Critical in production AI systems.

High Level Overview of the Agentic Engine Components and their interaction



1) User

- **Description:**
  Internal authenticated users of the organization accessing the AI system through a web application.
- **Capabilities:**
  - Start conversations with AI.
  - Upload files (documents, code, etc.) for AI-based processing.
- **Authentication:**
  - Authenticated via **Active Directory**.
  - Every action (query/file upload) is audited.

- **Interaction Points:**
  - Communicates with the **Browser (Frontend App)**.
  - User identity and session are validated at the Browser level and passed securely to backend services.

2) App

- **Description:**
  React.js based web application acting as the primary user interface.
- **Responsibilities:**
  - Provides a responsive user interface for chat/file uploads.
  - Manages user sessions, API calls, WebSocket connections.
  - Displays real-time streaming LLM responses.
- **Interaction Points:**
  - **REST API calls** to FastAPI backend.
  - **WebSocket connections** for receiving streamed LLM outputs.
  - Securely maintains user identity tokens to send with requests.

3) Core AI Engine Block

This is the **heart** of the backend system where all the business logic, AI processing, workflow orchestration, and data management happen.

---

a) Real-Time Streaming LLM Response using Websockets

- **Description:**
  Enables near real-time delivery of LLM outputs to the user, improving responsiveness and user experience.
- **Responsibilities:**
  - Establish WebSocket sessions between Browser and Backend.
  - Send token-by-token streaming response from LLM to Browser.
  - Handle large or long-running outputs without overloading browser memory.
- **Interaction Points:**
  - Connects with Browser.
  - Internally fetches responses from **Packages Block → AI Agents** and **LLM Connector**.

b) FastAPI Block

This block handles the **core API gateway** layer for the backend services.

---

i) Workflow

- **Description:**
    - Graph-based dynamic orchestration of multi-agent flows.
    - Selects, sequences, and executes appropriate AI Agents depending on user task (chat vs file upload).
- **Interaction Points:**
    - Invokes **AI Agents** for task-specific processing.
    - Calls **LLM Connector** to fetch LLM responses.
    - Stores/updates execution results into **DB Manager**.
    - Triggers **Observability & Prompt Studio** for logging prompts and responses.

ii) User Management

- **Description:**
    - Handles user authentication validation (Active Directory tokens).
    - Audits all user actions (input, uploads, activities).
- **Interaction Points:**
    - Interfaces with Active Directory.
    - Logs to **DB Manager** for historical records.

iii) RabbitMQ Producer

- **Description:**
    - Produces tasks (messages) to a RabbitMQ queue for asynchronous or long-running processes.
- **Interaction Points:**
    - Triggered by **Workflow** or **Services** layer.
    - Queues tasks for processing by **RabbitMQ Consumer**.
    - Associated metadata is logged via **Logging** module.

c) Packages Block

This contains the **primary business logic and AI processing units**.

---

i) AI Agents

- **Description:**
    - Specialized intelligent components/Agents performing specific functions:
        - Planner
        - Code Extraction
        - Business Rule Generation

        - Business Rule Review
        - Test Case Generation
        - Retrieval from Vector DB
        - **and more**
    - Each agent can call external tools/APIs via **MCP Server** if needed.
- **Interaction Points:**
    - Directly use **LLM Connector** to fetch AI model outputs.
    - Query **DB Manager** for historical knowledge base retrieval.
    - Use **MCP Server** to call external systems like **MetricStream** via **Wrapper APIs**.

ii) LLM Connector

- **Description:**
    - Manages dynamic connections to LLMs hosted on AWS Bedrock or SageMaker.
- **Capabilities:**
    - Supports multi-model connection.
    - Load balancing and fall-back if model fails.
    - Manages API keys, endpoint management, request optimization.
- **Interaction Points:**
    - Provides AI inference service to **AI Agents**.
    - Works with **Observability & Prompt Studio** to trace prompts/responses.

iii) DB Manager

- **Description:**
    - Manages all database interactions.
- **Data Storage:**
    - Relational Tables (Oracle 23):
        - User activities, sessions, logs
    - Vector Store (Oracle 23):
        - Knowledge base embeddings
- **Interaction Points:**
    - Used by **Workflow**, **AI Agents**, **User Management**, and **Services**.

iv) Logging

- **Description:**

- Captures all actions and events:
  - User actions
  - AI agent activities
  - API call results
  - Errors and latency

v) Services

- **Description:**
  - Business logic for non-AI tasks like metadata handling, retries, etc.
- **Interaction Points:**
  - Triggers RabbitMQ Producer for background workflows.
  - Uses DB Manager for storing intermediate results.

vi) RabbitMQ Consumer

- **Description:**
  - Consumes tasks from the RabbitMQ queue.
  - Executes background jobs triggered from user workflows.
- **Interaction Points:**
  - Interacts with **Services** and **AI Agents** to complete processing.
  - Updates status back into **DB Manager**.

4) Observability & Prompt Studio

- **Description:**
  - Central system to monitor and analyze all user/ai agent interactions.
- **Capabilities:**
  - Capture full prompt/response lifecycle traces.
  - Track user sessions, agent execution paths, errors.
  - Identify bottlenecks, model issues, latency.
- **Interaction Points:**
  - Dashboard and tracing interface available for debugging and optimization.

5) MCP Server

- **Description:**
  - Model Context Protocol Server managing external tools/APIs.

- **Capabilities:**
  - Catalog of all external services/tools metadata.
  - Dynamic invocation of tools/APIs at runtime by AI Agents.
- **Interaction Points:**
  - Called by **AI Agents** during task execution.
  - Connects with **Wrapper APIs** for MetricStream integration.

6) Wrapper APIs

- **Description:**
  - Abstract layer over MetricStream APIs.
- **Capabilities:**
  - Groups multiple low-level APIs into a single call.
  - Provides cleaned, structured results to AI Agents.
- **Interaction Points:**
  - Called only via **MCP Server** (not directly).
  - Fetches data from **MetricStream Instance** backend systems.

7) Redis Server

- **Description:**
  - Caching layer.
- **Usage:**
  - Temporary caching of frequently accessed data.
  - Session state management.
  - Short-term task results caching.
- **Interaction Points:**
  - Accessed by **Workflow**, **Services**, and **AI Agents**.

8) MetricStream Instance

- **Description:**
  - Client-specific instances of the MetricStream platform.
- **Purpose:**
  - Acts as a source of metadata.

- **Interaction Points:**
  - Accessed via **Wrapper APIs**.
  - Underlying access triggered dynamically via **MCP Server** during AI Agent workflows.