Software Requirements Specification

For

Federation Architecture for Composed Infrastructure Services (FACIS)

Federation Architecture Pattern IoT & AI

FACIS.FAP_IoT_AI

Version 1.0 (November 2025)

Published by
eco – Association of the Internet Industry (eco – Verband der Internetwirtschaft e.V.)
Lichtstrasse 43h
50825 Cologne, Germany

Table of Contents

Conformance Language

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC 2119[1].

1. Executive Summary

This Federation Architecture Pattern (FAP) Specification defines the technical requirements, architecture, interfaces, and acceptance criteria for implementing a federated IoT & AI data pipeline with IoT sensing and data collection, data transfer via OT/IT connectors using data connectors compliant with the Eclipse Dataspace Protocol (DSP)[2], data aggregation within a data lake tech-stack and LLM-based data enrichment to accompany dashboard data charts visualization.

The specification of this FAP describes how data moves from raw sensing to governed insight via visual dashboards in a way that is interoperable, auditable, and scalable. The process flow includes IoT sensoring which generates operational data that is collected in a data sink, coming from the IoT source. DSP-compatible connectors establish a bi-directional connection between data provider (IoT sources) and data consumer (data lake up to dashboard).

On the receiving side, the data should be transferred in a data lake/lakehouse setup that separates ingestion, curation, and serving into Bronze, Silver, and Gold data buckets/layers. The data lake/lakehouse is the single source of truth for the data which will be visualized in the dashboards while querying the data to render charts, for example, time-series views of sensor values, while an optional LLM service adds explanations and contextual guidance on the data. The data lake/lakehouse exposes governed interfaces that both the dashboard and the LLM can use in batch as well as in streaming mode, and it maintains the structure and metadata needed for humans and systems to understand and trust the data.

Technically, the solution separates the process flow cleanly. The Eclipse Dataspace Protocol (DSP) provides the interoperable control plane for catalogue discovery, policy-aware negotiation, and data transfer initiation, while the data plane is pluggable and implementation specific according to the technical scenario. For this implementation, only the data plane is in scope but the option to define sub-flows to the control plane must be considered. Additionally, there will be a clear data structure within the data lake/lakehouse which provides a clean data storage and access according to the use case scenarios. And finally, the dashboard with support of the LLM functionality can access the IoT data and provide the respective charts for human readability and data understanding.

The federated architecture pattern describes an overall system which enables therefore a secure, sovereign data exchange from IoT sensors at the edge through dataspace connector functionality to big data platforms in the cloud.

[1] https://www.rfc-editor.org/rfc/rfc2119
[2] https://projects.eclipse.org/projects/technology.dataspace-protocol-base

With regards to IoT sensing and data collection the specification covers seven primary components:

- IoT Sensors, Protocols (e.g. MQTT, OPC-UA, Modbus)
- Data Sink: Centralized data collection and normalization of IoT Data
- Provider Data Connector: Maintains a catalogue of available data in the Data Sink and exposes the DSP data plane (transfer services) and simulate the control plane (catalogue, negotiation,)
- Consumer Data Connector: receives data plane access credentials
- Data Connectors will transfer the data on a peer-to-peer base between data sink and data lake/lakehouse
- Data lake/lakehouse provides the interfaces to store and process the data according to the use case scenario with respective meta data and data policies and provides access to the data for the LLM up to the dashboard
- LLM agents and the dashboarding will enrich and visualize the data for data interpretation and exploration

The FAP specification and respective implementation follows open standards including W3C[3], HTTPS[4], Eclipse Dataspace Protocol, and aligns with federated dataspace architectures. The design supports multiple data profiles such as HTTPS pull, or Kafka for streaming and includes comprehensive security, trust, and governance controls.

With regards to the data lake/lakehouse the specification describes the following core functionalities and corresponding components:

- **Storage**
  (centralized) object-based cloud storage (S3, HDFS, Azure Blob, GCS)
- **Data Format**
  uniform data storage: Storage of structured, semi-structured and unstructured data in the same system (e.g. Parquet, ORC, JSON, CSV, images, logs, etc.) based on open storage format and schema information management (metadata)
- **Table Format**
  Transactional data management (ACID) supporting ACID transactions on data in the lake and enabling insert/update/delete/merge operations as in classic warehouses.
- **Data Processing**
  A common engine can execute batch and streaming jobs. Streaming data can be written incrementally to the lake and queried consistently with batch data. Supports near real-time analytics.
- **Query Engine**
  SQL-based and advanced analysis: Native support for SQL queries on lake data. Integration with BI tools, support for machine learning and data science. Enables uniform data queries across multiple formats and sources

For the AI supported Dashboard the AI integration will leverage open standards for prompt-based inference to ensure interoperability. Relevant data from the data lake/lakehouse (e.g.,

[3] https://www.w3.org/TR/did-1.0/ and https://www.w3.org/TR/vc-data-model/
[4] https://www.rfc-editor.org/info/rfc2818

via SQL queries or REST APIs) are retrieved and embedded into prompts and sent to an OpenAI-compatible endpoint. This approach maintains data sovereignty by processing data minimally and on-demand, with outputs governed by the same policies as the underlying datasets. Data outputs will leverage the same open standards and visualize either directly via LLM and/or through a use case optimized method.

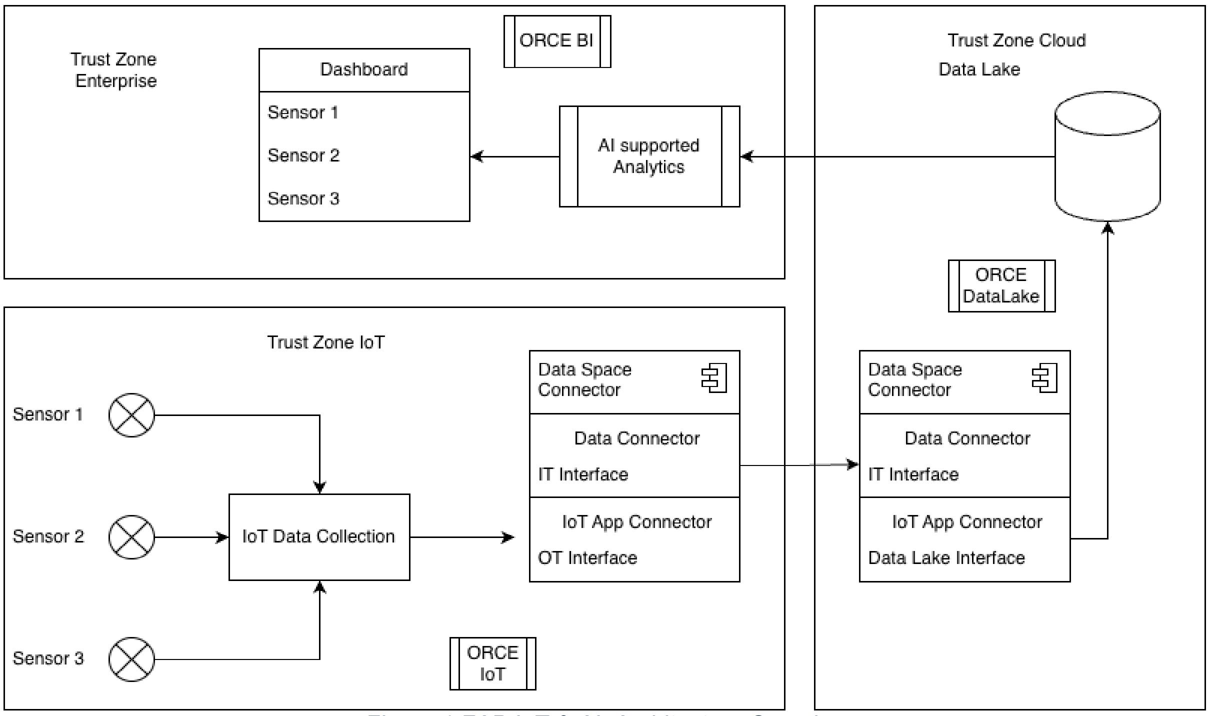


Figure 1 FAP IoT & AI: Architecture Overview

The purpose of this tender is to commission software development services for Lot "Federation Architecture Pattern IoT & AI". The delivery of the software is provisionally by Q2 of 2026. The final clarification will be communicated after the award of the tender.

2. Background & Context

Modern industrial and IoT environments require standardized mechanisms to share operational data across organizational boundaries while maintaining data sovereignty and usage control. Traditional point-to-point integrations lack scalability, auditability, and policy enforcement capabilities required for federated ecosystems. The Eclipse Dataspace Protocol provides a foundation for interoperable, policy-driven data exchange. This FAP specification applies the principles of the DSP to the IoT domain, bridging operational technology (OT) environments with enterprise IT and cloud analytics platforms through standardized connectors.

Enterprises need a data lake/lakehouse since it unifies data storage, processing, and analytics in one scalable, governed architecture once it is enabling a faster, and more reliable, and more cost-effective data-driven decisions. A data lake/lakehouse is an architecture that combines the low-cost, open storage of data lakes with the data management, governance, and performance features of data warehouses which are available all in a unified platform. This makes it just as suitable for analytics and BI as it is for IoT or machine learning and AI use cases.

In addition to robust data storage and processing, modern IoT ecosystems benefit from AI-driven analysis to derive actionable insights from aggregated data. Large Language Models (LLMs) can enrich raw or curated datasets by generating summaries, detecting anomalies, or providing contextual explanations, such as interpreting time-series sensor trends in natural language. This integration of IoT Data coming from IoT sensors, stored and managed in a data lake/lakehouse setup, occurs via standardized interfaces, like OpenAI-compatible APIs for prompt-based inference, ensuring compatibility across providers while adhering to governance policies for data minimization and auditability.

Finally, in this FAP IoT & AI Dashboards serve as the final visualization layer, rendering governed data from the data lake/lakehouse into interactive charts, such as real-time time-series views or aggregated metrics. This enables stakeholders to monitor operations, respond to changes (e.g., detecting modified sensor data within seconds), and leverage AI-enriched outputs for informed decision-making.

LLM outputs SHALL be structured (e.g., JSON with summaries, anomalies, and metadata) for direct integration into dashboard charts, enabling dynamic updates without re-querying the lake.

Together, these elements form a sovereign end-to-end pipeline that supports federated ecosystems without compromising scalability or security.

Key Drivers

This section highlights the fundamental drivers and design principles that shape the FAP for IoT & AI data pipelines. These key drivers establish the foundation for a secure, scalable, and interoperable system architecture that addresses the critical challenges in industrial IoT environments. To implement such a beforehand described system, we see key drivers which are essential to setup the end-to-end running system. The drivers below explain why a protocol-driven, federated architecture is preferable to integrations and guide both the scope and the acceptance criteria of this specification.

Data Sovereignty: Organizations must be able to maintain control over usage policies and access terms for their data. In federated systems where multiple organizations collaborate, maintaining control over sensitive operational data while enabling selective sharing is essential, whereas centralized approaches create security risks and sovereignty concerns.

Solution: The bi-directional control-plane connection between data providers (IoT sources) and data consumers (data lake/lakehouse) provides control over the data.

Streaming and Batch Processing: IoT systems must support both real-time monitoring (streaming) and batch analysis.

Solution: This FAP accommodates both processing paradigms by allowing dataspace connectors to push real-time sensor data for immediate dashboard updates and alerting, while scheduled data ingestion pipelines handle batch data needed for trend analysis and model training. These capabilities ensure that organizations can address operational monitoring and strategic analytics use cases with a single, coherent infrastructure.

Interoperability: Standardized protocols enable ecosystem-wide data exchange without vendor lock-in. Industrial IoT environments typically involve heterogeneous sensor networks using diverse communication protocols (MQTT, OPC-UA, Modbus, etc.). The lack of standardization creates integration challenges and vendor lock-in.

Solution: The Eclipse Dataspace Protocol (DSP) provides a unified, protocol-agnostic layer for data exchange while implementing DSP-compliant dataspace connectors for data transfer.

Scalability: Federated architecture supports thousands of data sources and consumers across trust zones. IoT systems generate massive volumes of high-velocity time-series data. Coupling data transfer with control operations creates bottlenecks and limits system scalability.

Solution: The clear separation between control plane and data plane enables an independent scaling.

Edge-to-Cloud Continuum: Modern IoT architectures must balance edge processing capabilities with cloud-scale analytics, dynamically adapting to varying latency, bandwidth, and processing requirements.

Solution: The specification supports a flexible deployment model where IoT sensors at the edge perform initial data generation and protocol-specific operations, edge-level dataspace connectors prepare this data for secure transfer, and centralized analytics platforms such as the data lake/lakehouse setup provide enterprise-scale storage and AI/ML capabilities. Tiered data management across Bronze (raw), Silver (curated), and Gold (analytics-ready) layers ensures that data is refined and optimized for diverse operational and analytical use cases.

Real-Time Analytics and Readability: Business users need immediate insights from IoT data without requiring deep technical expertise in data engineering or query languages.

Solution: The LLM-powered dashboard layer could enable natural language queries so users can ask questions in plain language and receive relevant visualizations, while AI-driven exploration helps to understand patterns, anomalies, and relationships in sensor data.

3. Scope

This chapter defines the overall scope of the FAP. It clarifies what capabilities are delivered end-to-end, from device data collection, through DSP-governed data transfer and exchange, to governed consumption in the data lake/lakehouse and what items/topics are excluded. Furthermore, the visualisation in dashboards and enrichment by LLMs. The intent is to give implementation teams a concrete build plan while enabling business stakeholders to understand feasibility, effort, and boundaries.

3.1 In Scope

Below is defined what the implementation must deliver to be compliant, maintainable, and useful in production.

Transfer Process Protocol, implementations MUST implement the Transfer Process Protocol as defined in DSP 2025-1 RC4. This includes:

- The complete state machine for transfer processes
- Message exchanges for Transfer Request, Transfer Start, Transfer Suspension, Transfer Completion, and Transfer Termination
- Support for both push and pull transfer types
- Support for both finite and non-finite data transfers

Transfer Process HTTPS Binding, implementations MUST expose the transfer API as REST over HTTPS with the binding's error semantics. Specifically:

- HTTP 400 responses MUST be returned on invalid state transitions
- HTTP 400 responses MUST include a Transfer Error payload as defined in DSP 2025-1 RC4
- All other error semantics defined in the HTTPS binding MUST be implemented

Agreement Prerequisites, implementations MUST assume that a valid agreement exists before initiating a transfer. This FAP does not define contract negotiation workflows. Implementations MUST require an Agreement ID to be available before starting any transfer process.

Data Sink and Catalogue, implementations MUST:

- Provide a Data Sink component positioned near data sources
- Derive the Provider connector catalogue from that Data Sink
- Catalogue publication MAY remain minimal as contract negotiation is outside the scope of this FAP

Consumer Connector, implementations MUST provide a Consumer connector that:

- Starts transfers using a valid Agreement ID
- MAY implement discovery capabilities (discovery is OPTIONAL)

Data Plane, implementations MUST provide data-plane implementations that are separate from the DSP control plane. The data plane MUST support:

- Pull mechanisms for files and objects
- Batch delivery capabilities
- Streaming data delivery

Note: Specific tooling and technology choices are not mandated; examples provided in this specification are illustrative only.

Protocol-to-Streaming Bridge, implementations MAY provide a protocol-to-streaming bridge for sensor data streams. This capability is OPTIONAL.

Data Ingestion and Medallion Architecture, implementations MUST provide receiver-side data ingestion that:

- lands incoming data into a bronze layer (raw data storage)
- provides curation capabilities to transform data into Silver (validated, enriched) and gold (business-ready) layers
- implements the Medallion architecture pattern

Data lake/Lakehouse Storage, implementations MUST provide data lake/lakehouse storage with the following characteristics:

- use of a transactional table format (e.g., Delta Lake, Iceberg, Hudi)
- storage in open columnar file formats
- practical partitioning strategies for query performance

Batch and Stream Processing, implementations MUST support both batch and streaming processing modes. The Gold layer MUST contain curated business models and KPIs suitable for analytics consumption.

Governance and Identity, implementations MAY provide:

- policy enforcement hooks for data usage purpose, time-to-live, and rate limiting
- identity validation mechanisms
- claims and verifiable credentials (VC) validation capabilities
- Policy based data access control

Audit and Observability, implementations MUST provide audit and observability capabilities across:

- transfer processes
- data ingestion operations
- data transformations

Semantic and API Access, implementations MUST provide semantic and API access for dashboards that support:

- KPI queries
- time-series data access
- MAY provide LLM-ready aggregates and metadata

LLM Enrichment, implementations MAY provide LLM enrichment capabilities with the following constraints:

- Retrieval-Augmented Generation (RAG) SHOULD operate only over allowed metadata and curated aggregates
- structured outputs SHOULD be stored in separate tables
- LLM access MUST NOT widen data access beyond the negotiated scope defined in the Agreement
- this capability is entirely OPTIONAL

3.2 Out of Scope

The following capabilities and features are explicitly out of scope for this FAP specification.

Contract Negotiation, contract negotiation workflows are out of scope. This specification treats contract negotiation as a prerequisite that supplies an Agreement ID. The negotiation process itself is not specified here.

Dataspace Federation Governance, data federation, governance and marketplace operations are out of scope for this specification.

OT Protocol Implementations, deep Operational Technology (OT) protocol server implementations are out of scope.

Complete Analytics Platform, building a full analytics platform, ML training pipelines, or dashboard products are out of scope. This specification focuses on integration via APIs rather than providing complete end-user applications.

Vendor-Specific Implementations, binding the architectural design to specific vendors is out of scope. Examples provided in this specification remain illustrative and technology-agnostic.

Advanced Security Features, security features such as Hardware Security Modules (HSM) and confidential computing are out of scope beyond the basic security requirements defined in this FAP.

Enterprise Monitoring Stacks. production-grade enterprise monitoring stack selection and implementation are out of scope. This specification defines requirements for exposing logs and metrics, but platform selection is left to implementers.

Advanced Data Governance, advanced data governance capabilities beyond the technical controls specified in above section are out of scope.

High Availability Design, detailed backup, failover, and high availability (HA) design specifications are out of scope.

Event-Driven Dashboard Updates, implementations provide eventing mechanisms so that dashboards can refresh only the visual components affected by data changes, rather than requiring full page refreshes.

DevOps Specifics, DevOps implementation details including Infrastructure as Code (IaC) and CI/CD pipelines are out of scope for this specification.

4. Conceptual Architecture

The system follows a federated architecture with clear separation between data collection (IoT sensors, protocols, Data Sink), control plane (DSP for discovery/negotiation/coordination), data plane (actual data transfer via HTTPS/SFTP/Kafka), and data consumption (data lake ingestion). The architecture spans IoT/OT, Enterprise IT, and

Cloud trust zones, with dataspace connectors mediating secure data exchange through the Eclipse Dataspace Protocol.

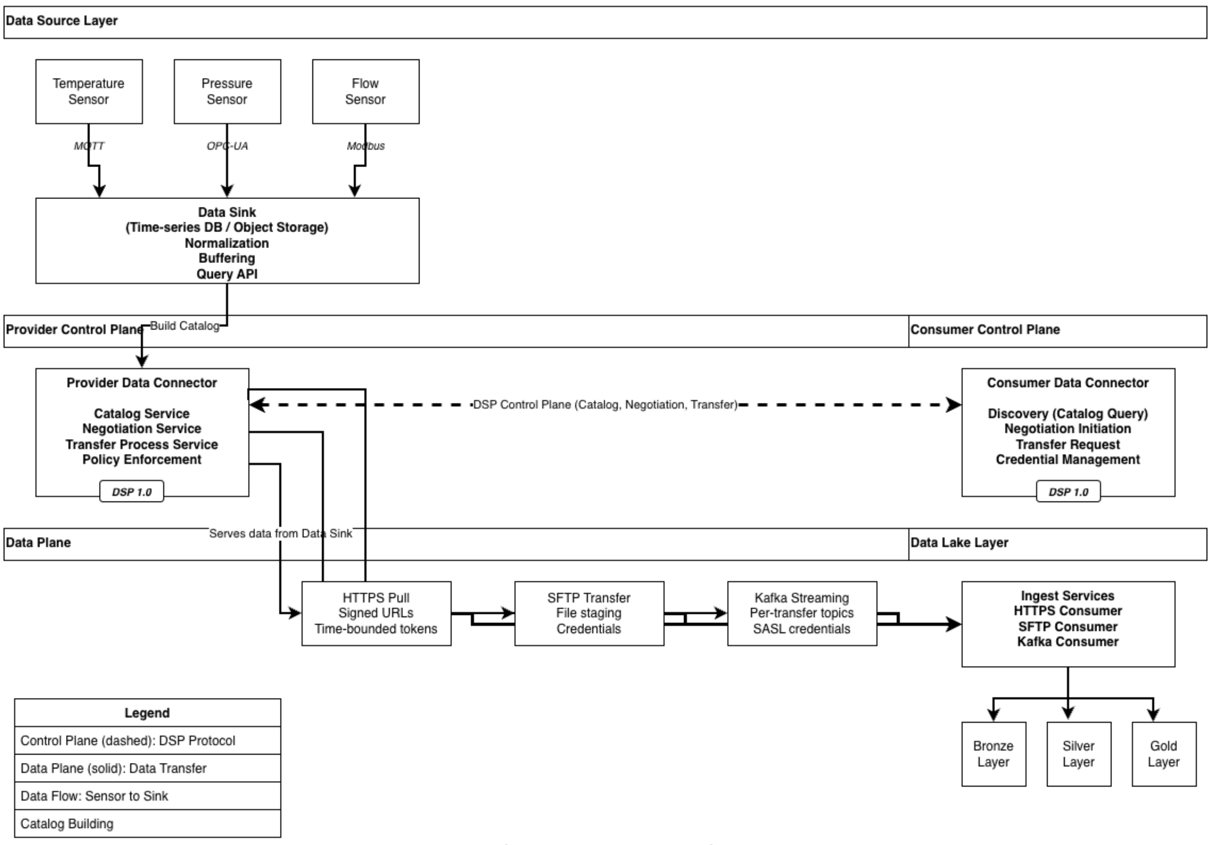
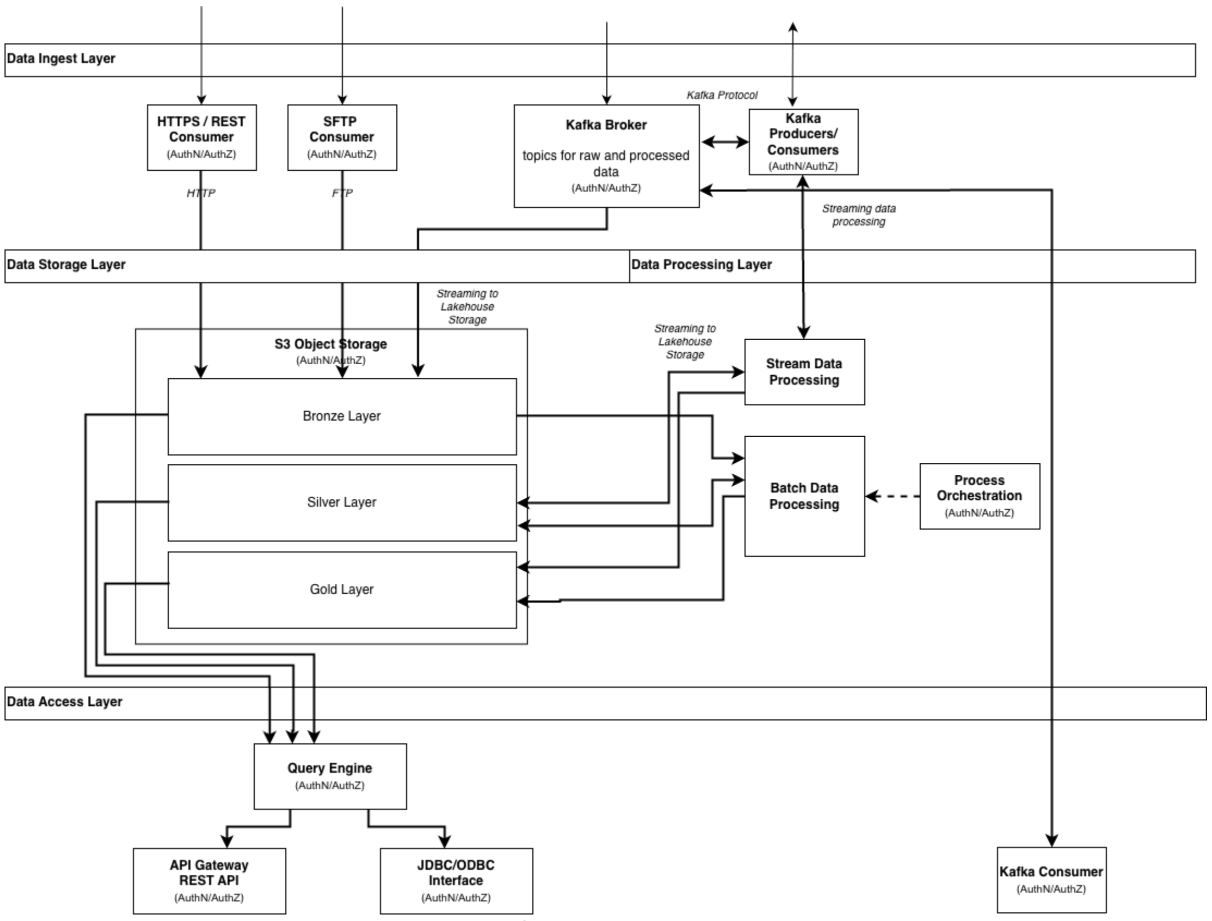

Data Source Layer
Temperature Sensor
Pressure Sensor
Flow Sensor
MQTT
OPC-UA
Modbus
Data Sink (Time-series DB / Object Storage) Normalization Buffering Query API
Provider Control Plane
Build Catalog
Consumer Control Plane
Provider Data Connector
Catalog Service
Negotiation Service
Transfer Process Service
Policy Enforcement
DSP 1.0
DSP Control Plane (Catalog, Negotiation, Transfer)
Consumer Data Connector
Discovery (Catalog Query)
Negotiation Initiation
Transfer Request
Credential Management
DSP 1.0
Data Plane
Serves data from Data Sink
Data Lake Layer
HTTPS Pull Signed URLs Time-bounded tokens
SFTP Transfer File staging Credentials
Kafka Streaming Per-transfer topics SASL credentials
Ingest Services HTTPS Consumer SFTP Consumer Kafka Consumer
Bronze Layer
Silver Layer
Gold Layer
Legend
Control Plane (dashed): DSP Protocol
Data Plane (solid): Data Transfer
Data Flow: Sensor to Sink
Catalog Building

Figure 2 Overview IoT Data Connector

The conceptual architecture of the data lake follows a typical data lakehouse architecture and supports as well streaming (Kafka) as batch (HTTPS / SFTP) data ingestion and processing.



Data Ingest Layer
HTTPS / REST Consumer (AuthN/AuthZ)
SFTP Consumer (AuthN/AuthZ)
Kafka Broker
topics for raw and processed data (AuthN/AuthZ)
Kafka Protocol
Kafka Producers/ Consumers (AuthN/AuthZ)
HTTP
FTP
Streaming data processing
Data Storage Layer
Data Processing Layer
Streaming to Lakehouse Storage
S3 Object Storage (AuthN/AuthZ)
Streaming to Lakehouse Storage
Stream Data Processing
Bronze Layer
Silver Layer
Gold Layer
Batch Data Processing
Process Orchestration (AuthN/AuthZ)
Data Access Layer
Query Engine (AuthN/AuthZ)
API Gateway REST API (AuthN/AuthZ)
JDBC/ODBC Interface (AuthN/AuthZ)
Kafka Consumer (AuthN/AuthZ)

Figure 3 Overview Data Lakehouse

4.1 Trust Zones

4.1.1 IoT/OT Zone

Operational technology environment with sensors, PLCs, and field devices. IoT sensors connect via MQTT, OPC-UA, Modbus, or HTTP protocols to a centralized Data Sink. The Data Sink normalizes incoming data, buffers for reliability, and provides queryable APIs. The Provider DSP Connector reads from the Data Sink to build the dataset catalogue and exposes DSP control plane endpoints (catalogue, negotiation, transfer services).

4.1.2 Enterprise IT Zone

Corporate network hosting enterprise applications, dashboards, and business logic. Dataspace Protocol based data connector can act as both consumer (pulling from edge) and provider (exposing curated datasets). Identity Hub manages organizational credentials.

4.1.3 Cloud Zone

Cloud-hosted data platform with data lake, analytics services, and AI/ML capabilities. Consumer DSP Connector performs discovery (queries provider catalogue), negotiates contracts using DSP protocol, and requests data transfers. After receiving data plane access credentials from the provider, the Consumer coordinates Data Lake Ingest Services which consume data via HTTPS pull, SFTP file transfer, or Kafka streaming (separate from the DSP control plane). Ingest services land data to bronze layer, with Governance services tagging, cataloguing, and enforcing retention policies.

4.2 Key Components

Table 1 Key Components for FAP IoT & AI

Component	Responsibility
IoT Data Collection	Sensor data normalization, buffering, windowing, aggregation
OT/IoT Connector	Protocol adapters (OPC UA, MQTT, Modbus), bridge to HTTP/Kafka
DSP Connector	Control plane (catalog, negotiation, transfer), policy enforcement
DSP Connector Data Plane	HTTP API (signed URLs), Kafka topics (SASL/mTLS credentials)
Identity Hub	Store/verify Verifiable Credentials, Decentralized Claims Protocol presentation/verification
Policy Agent	Enforce rate limits, purpose restrictions, time-bounded access
Data Lakehouse Ingest	HTTP validator, Kafka consumers, schema registry, Bronze/Silver/Gold Batch Ingest SFTP, HTTPS
Data Lakehouse Storage	S3 Object Storage, Columnar file storage format (Parquet, ORC); ACID transactions, schema evolution, time travel, Apache (Iceberg), Medallion architecture data design pattern for organizing data
Data Lakehouse data processing	ETL/ELT processes, batch and near real time, batch process orchestration
Data Lakehouse data access	SQL query engine, JDBC/ODBC interfaces, REST Interfaces
Governance	Authentication, Authorization, RBAC/ABAC, Access policies
LLM Inference Service	API calls to OpenAI-compatible service (e.g., IONOS AI Model Hub) for generative processing and API Exposure for Output

5. Functional Requirements

5.1 Dataspace Protocol Control Plane

FR-DSP-001: DSP Catalogue Service

Priority: SHOULD HAVE
Description: The Provider connector SHOULD expose a catalogue endpoint that returns datasets and offers metadata conforming to DSP 1.0 specification (see Appendix B).

Acceptance Criteria:
- POST /dsp/catalogue/request returns HTTP 200 with valid JSON
- Response includes datasets array with id, metadata, and offers
- Pagination supported with nextCursor field
- Filtering by assetType, dataset IDs accepted
- Passes DSP TCK catalogue suite

FR-DSP-002: DSP Contract Negotiation

Priority: MAY HAVE
Description: The connector MAY implement async contract negotiation state machine per DSP 1.0 (REQUESTED → FINALIZED → TERMINATED states).

Acceptance Criteria:
- POST /dsp/negotiations creates negotiation and returns 202 with negotiation Id
- GET /dsp/negotiations/{id} returns current state and agreement Id when FINALIZED
- POST /dsp/negotiations/{id}/terminate sets TERMINATED state
- State transitions logged with correlation IDs
- Passes DSP TCK negotiation suite

FR-DSP-003: DSP Transfer Process

Priority: MUST HAVE
Description: The connector MUST support transfer process creation, state tracking, and access parameter provisioning per DSP 1.0.

Acceptance Criteria:
- POST /dsp/transfers with agreementId creates transfer and returns 202 with transferId
- GET /dsp/transfers/{id} returns state and access object when COMPLETED
- Access object contains data plane parameters (URL or Kafka connection)
- Error states include reason field
- Passes DSP TCK transfer suite

5.2 Data Plane Profiles

FR-DP-001: DSP HTTP Pull Profile

Priority: MUST HAVE
Description: The system MUST support HTTP pull data plane for time-windowed snapshots with signed URLs.

Acceptance Criteria:

- Transfer with format: http-pull returns { url, token, expiresAt }
- URL accepts from/to query parameters for time windows
- Token validates and expires per TTL
- Consumer can fetch data within validity window
- Rate limits enforced per agreement

FR-DP-002: Streaming Profile

Priority: Should HAVE
Description: The system MUST support Kafka streaming data plane with per-transfer topics and SASL credentials.

Acceptance Criteria:

- Transfer with format: Kafka returns { bootstrap, topic, sasl, expiresAt }
- Per-transfer topic created on-demand
- SASL PLAIN or SCRAM credentials provisioned
- Consumer can subscribe and receive events
- Credentials expire per TTL and ACLs cleaned up

FR-DP-003: MQTT to Kafka Bridge

Priority: SHOULD HAVE
Description: The system SHOULD provide a bridge service forwarding MQTT messages to Kafka topics.

Acceptance Criteria:

- Bridge subscribes to MQTT topic pattern (e.g., iot/#)
- Messages forwarded to Kafka with enriched metadata (timestamp, source topic)
- Configurable topic mapping and message transformation
- Handles backpressure and reconnection

5.3 Identity & Trust

FR-IAM-001: Organization Identity

Priority: MUST HAVE
Description: Organizations MUST be identified by W3C DIDs and hold Verifiable Credentials for participant role.

Acceptance Criteria:

- Each organization has DID (did:web[5] or did:key[6])
- Participant VC issued per OIDC4VCI flow
- VC contains claims: id, name, roles, trust framework reference
- Credentials stored in Identity Hub with query API

FR-IAM-002: DCP Integration

Priority: SHOULD HAVE
Description: The system SHOULD support Decentralized Claims Protocol for VC presentation and verification.

Acceptance Criteria:

- Negotiation request includes VP with participant VC
- Connector verifies VP signature and VC validity
- Expired or revoked credentials rejected
- Verification result logged for audit

5.4 Data Lake Integration: Data Ingest

FR-DL-001: HTTP Ingest Service

Priority: MUST HAVE
Description: The data lake MUST provide HTTP ingest endpoint validating signed URLs and storing objects to Bronze layer.

Acceptance Criteria:

- Validates token and URL expiry
- Fetches data from provider endpoint
- Stores to object storage with partition key (e.g. agreementId, timestamp, ...)
- Registers landed data with governance catalog
- Tags with metadata (provider, asset, agreement, policy)

FR-DL-002: Kafka Ingest Service

Priority: MUST HAVE
Description: The data lake MUST provide Kafka consumer groups landing stream data to Bronze layer.

Acceptance Criteria:

- Subscribes using transfer access parameters (bootstrap, topic, credentials)
- Validates message schema against registry
- Writes to Bronze in Parquet format with hourly partitions
- Commits offsets after successful write
- Handles late-arriving data and duplicate detection

[5] https://w3c-ccg.github.io/did-method-web/
[6] https://w3c-ccg.github.io/did-key-spec/

FR-DL-003: File Ingest Service

Priority: MUST HAVE
Description: The data lake MUST provide an SFTP file service

Acceptance Criteria:
SFTP protocol is supported for file ingest

5.5 Data Lakehouse Data Storage

FR-DL-004 Data Storage

Priority: MUST HAVE

Description: The data lake MUST provide an S3-compatible object storage and use efficient storage formats

Acceptance Criteria:
- Files can be stored in an S3 Object Storage
- At least in silver and gold layer data are stored using a columnar file storage format (Parquet, ORC)
- Data storage supports ACID transactions, schema evolution and time travel, Apache (Iceberg)

FR-DL-005: Data Storage: Logical Lake Layering

Priority: SHOULD HAVE
Description: The data lake SHOULD implement Medallion architecture (Bronze/Silver/Gold) architecture for progressive refinement. The Medallion Architecture is an architecture pattern used in modern data lakehouse architectures to organize data into progressive quality layers — typically called Bronze, Silver, and Gold – to improve data quality, structure, and usability step by step.

Acceptance Criteria:
- **Bronze:** Raw ingested data with no / minimal transformation, accessible by role data engineer; data are partitioned by load timestamp
- **Silver:** Cleansed, validated, and enriched data, accessible by roles data engineer, data scientist
- **Gold:** Business-domain specific aggregates ready for data products: analytics, ML, BI, applications
- Data is stored in open formats (e.g. Parquet, ORC, Delta, Iceberg)
- Retention policies enforced per agreement terms

5.6 Data Lakehouse Data Processing

FR-DL-006: Realtime Data Processing

Priority: MUST HAVE

Description:
Near real-time ETL/ELT pipelines are the operational backbone of a data lakehouse if data updates are time critical.
Data are ingested, transformed, and organized in real time ensuring that every layer (Bronze, Silver, Gold) is kept accurate, fresh, and analysis-ready.

Acceptance Criteria:
- Trigger: data processing (ingestion) is triggered event-driven
- Data Source: Can subscribe to message bus topics, read from (log) event streams
- Processing: Continuous, record-by-record or micro-batch
- Window functions

FR-DL-007: Batch Data Processing

Priority: MUST
Description:
Automated ETL/ELT pipelines are the operational backbone of a lakehouse. They ingest, transform, and organize data in regular time intervals ensuring that every layer (Bronze, Silver, Gold) is kept accurate, fresh, and analysis-ready.

Acceptance Criteria:
- Trigger: Data processing time-based (hourly, daily, weekly) or event-based batch
- ETL jobs to promote data between layers, persistent view where appropriate
- Data Sources: Files, databases, APIs
- Processing: Process chunks (batches) of data periodically

FR-DL-008: Process Orchestration

Priority: SHOULD HAVE
Description:
Process automation is a key enabler in a data lakehouse architecture, because it ensures that data flows, transformations, and quality checks happen reliably, repeatably, and at scale without manual intervention.
Without automation, a lakehouse would quickly become unreliable or inconsistent. Automation ensures that:
- Data arrives on time (e.g., IoT, logs, APIs)
- Transformations run in the correct order and dependencies
- Quality and governance rules are applied
- Pipelines recover from failures
- The system can scale without human intervention

Acceptance Criteria:
- Orchestration Manage and schedule data pipelines (order, dependencies, retries)
- Ingestion Automation: Automatically pull data from sources

- Run data transformations (i.e. data cleaning, enrichment, aggregation jobs) on schedule or trigger
- Mechanism for pipelines recover from failures (“backfill”)

5.7 Data Lakehouse data access

FR-DL-009 SQL Query Engine

Priority: MUST HAVE
Description:
The query engine is a core component of any data lakehouse architecture because it enables SQL-style access and analytics directly on data stored in open formats (e.g. Parquet, ORC, Delta, Iceberg). The query engine is responsible for:

- Reading data from the lake’s open storage (S3, HDFS, etc.)
- Interpreting metadata (from a metastore or catalogue like Hive, Glue, or Unity Catalog)
- Optimizing and executing queries (SQL or DataFrame-based)
- Returning results to BI tools, notebooks, applications or LLMs

Acceptance Criteria:

- SQL Query Execution: Run ad-hoc or scheduled SQL queries over large datasets
- Cost-based Optimization (CBO): Chooses efficient query plans and file reads
- Predicate Pushdown: Reads only the necessary data blocks for a query
- Federation: Optionally joins data across multiple sources (Lake, DB, APIs)
- Caching / Materialization: Speeds up repeated queries
- Concurrency: Supports many parallel users and queries

FR-DL-010 Data Access

Priority: MUST HAVE
Description:
Data access in a lakehouse is the controlled ability to interact with data, encompassing who can access which datasets, under what conditions, and through which interfaces, while ensuring security, governance, and operational efficiency.

Acceptance Criteria:

- Data are accessible via REST API
- Data are accessible via ODBC / JDBC protocol
- (CAN HAVE) Data Access via MCP protocol
- (CAN HAVE) Data Access via BI tool connectors
- Streaming ingestion/consumption
- Data access is restricted by role/policy-based access control, Granularity: Table/Column/Row level
- Authentication: Integration with identity providers (Keycloack), Support for service accounts or API tokens for applications (CAN HAVE) multi-factor authentication (MFA)

5.8 Data Governance

FR-DL-011 Data Access Control

Priority: SHOULD HAVE
Description:
Data governance is the set of processes, policies, and technologies that ensures data within the lakehouse is accurate, consistent, secure, discoverable, and used responsibly across the organization, while meeting regulatory, legal, and operational requirements. In the scope of this FAP we only consider technical functions that support data security and data access policies (authentication / authorization) for reasons of simplicity.

Acceptance Criteria:

- Authentication to store data via SSI
- Policy based data lake access to store data (on organization level)
- Authentication to access data via SSI
- Policy based data lake access to access data
- Policies refer to roles or attributes. RBAC/ABAC

5.9 AI Processing

FR-AI-001: Data Retrieval from Data Lake

Priority: MUST HAVE
Description: The system MUST retrieve data from Silver/Gold layers via REST API or ODBC/JDBC (per FR-DL-010)

Acceptance Criteria:

- Queries filter by timestamp, assetId, or agreementId.
- Detects data modifications.
- Handles policy-based access control, rejecting unauthorized requests with 403.
- Response time < 500ms for 1000 records at p95.

FR-AI-002: LLM Inference via OpenAI-Compatible API

Priority: MUST
Description: The system MUST send retrieved data to an OpenAI-compatible LLM service (e.g., IONOS AI Model Hub) via its API for inference (e.g., summarization, anomaly detection).

Acceptance Criteria:

- Uses configurable HTTPS POST to /v1/chat/completions (or equivalent) with API key authentication.
- Payload: JSON with model (e.g., “llama-2-7b-chat”), messages array including data as prompt.
- Handles responses: Extracts choices[0].message.content as output.
- Rate limits: Enforce per agreement (e.g., 10 req/min) via policy hooks.
- Error handling: Retry on 429/5xx, log failures.
- Provides Output via API

FR-AI-003: Governed LLM Access to Data Lake

Priority: SHOULD HAVE
Description: The system SHOULD enable LLMs to access data lake content via standardized, OpenAI-compatible API prompts that incorporate queries to lake interfaces (e.g., REST API or ODBC/JDBC per FR-DL-010). Prompts SHALL be formatted to include contextual data (e.g., 'Query the Silver layer for sensor data from [timestamp] and summarize anomalies'), with responses governed by policies (e.g., RBAC/ABAC from FR-DL-005).

Acceptance Criteria:

- Prompts use JSON structures compatible with OpenAI's /v1/chat/completions endpoint.
- Access rejects unauthorized queries per policy (e.g., HTTP 403).
- Supports batch (e.g., daily summaries) and streaming (e.g., real-time anomaly detection) modes.

6. Technical Architecture

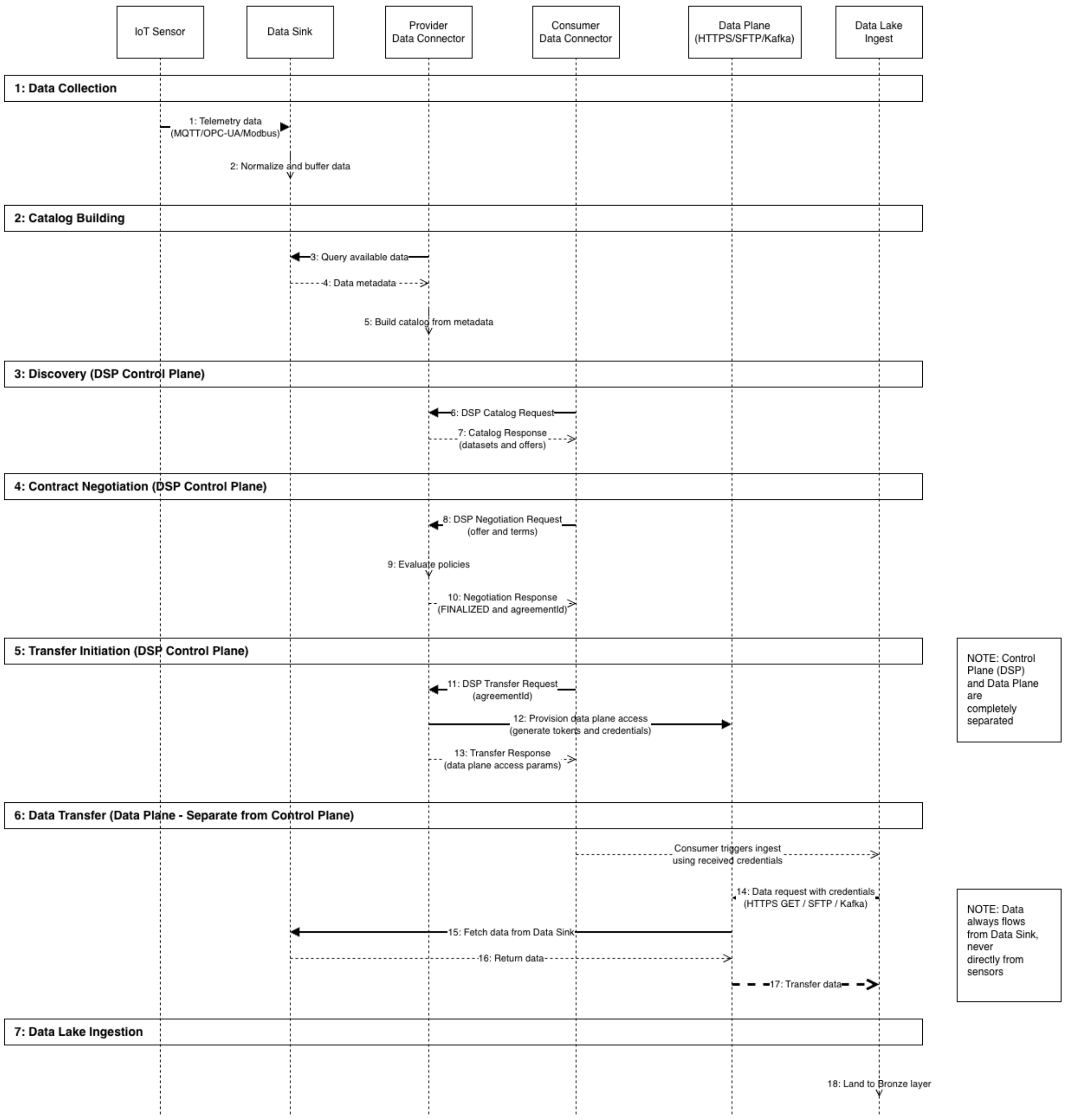

IoT Sensor
Data Sink
Provider Data Connector
Consumer Data Connector
Data Plane (HTTPS/SFTP/Kafka)
Data Lake Ingest
1: Data Collection
1: Telemetry data (MQTT/OPC-UA/Modbus)
2: Normalize and buffer data
2: Catalog Building
3: Query available data
4: Data metadata
5: Build catalog from metadata
3: Discovery (DSP Control Plane)
6: DSP Catalog Request
7: Catalog Response (datasets and offers)
4: Contract Negotiation (DSP Control Plane)
8: DSP Negotiation Request (offer and terms)
9: Evaluate policies
10: Negotiation Response (FINALIZED and agreementId)
5: Transfer Initiation (DSP Control Plane)
NOTE: Control Plane (DSP) and Data Plane are completely separated
11: DSP Transfer Request (agreementId)
12: Provision data plane access (generate tokens and credentials)
13: Transfer Response (data plane access params)
6: Data Transfer (Data Plane - Separate from Control Plane)
Consumer triggers ingest using received credentials
14: Data request with credentials (HTTPS GET / SFTP / Kafka)
NOTE: Data always flows from Data Sink, never directly from sensors
15: Fetch data from Data Sink
16: Return data
17: Transfer data
7: Data Lake Ingestion
18: Land to Bronze layer

Figure 4 IoT Data Connector Sequence

6.1 Control Plane Architecture

The DSP control plane follows a microservices architecture with clear separation of concerns. The Provider Data Connector exposes three main DSP services:

(1) Catalogue Service – builds and exposes dataset catalogue from Data Sink,
(2) Negotiation Service – handles contract negotiation with policy evaluation,
(3) Transfer Process Service – coordinates transfer initiation and provisions data plane access credentials.

The Consumer Data Connector implements corresponding client services for discovery, negotiation, and transfer requests.

CRITICAL: The control plane NEVER transfers data – it only coordinates access and authorization.

Table 2 FAP IoT & AI Service Implementation

Service	Implementation (exemplary)
Catalogue Service	FastAPI REST endpoint with PostgreSQL backend, Redis cache
Negotiation Service	State machine with PostgreSQL persistence, async callbacks via webhook
Transfer Service	Orchestrates data plane provisioning, stores access params in PostgreSQL
Policy Agent	OPA (Open Policy Agent) for policy evaluation and enforcement
Identity Hub	Verifiable Credential storage with DCP verification endpoints

The implementation of the control plane is not in scope but must be prepared to be included.

6.2 Data Plane Architecture

6.2.1 HTTP Pull Data Plane

Components:

- **URL Signer Service:** Generates HMAC-signed URLs with embedded expiry and scope
- **Data API:** FastAPI endpoint serving IoT data with time window filters
- **Rate Limiter:** Redis-backed token bucket per agreement

6.2.2 Streaming Data Plane

Components:

- **Kafka Cluster:** Distributed log with SASL SCRAM authentication
- **Topic Manager:** Creates per-transfer topics with retention and ACLs
- **SCRAM Provisioner:** Generates short-lived credentials and cleans up on expiry
- **Schema Registry:** Validates message schemas and enforces compatibility

6.3 Data Lake Architecture

6.3.1 Ingest Layer

- **HTTP Ingest:** Service validating tokens and fetching from signed URLs Apache NiFi (Low Code), REST Service, Trino
- **Kafka Consumer: Reading from Kafka topic and store in Bronze layer** Spark structured streaming, Flink to Parquet file format
- **Schema Validation:** Enforces data contracts and rejects malformed data
- SFTP file transfer, Apache NiFi (Low Code), Trino

6.3.2 Storage Layer

- **Bronze:** S3 object storage (e.g. MinIO, Garage, CephFS) with Hive partitioning (agreement_id, year, month, day, hour, ...)
- **Silver:** Delta Lake / Iceberg tables with data quality rules and deduplication
- **Gold:** Curated aggregates materialized as views or tables

6.3.3 Data Processing Layer

- Batch data processing low code: Apache NiFi
- Realtime data processing low code: Apache NiFi
- Orchestration: Apache Airflow
- Apache Spark for large scale data analytics, (Pyspark, Java, Scala)

6.3.4 Data Access Layer

- Data retrieval from S3 storage: SQL query engine: Trino
- Interface for SQL-tools JDBC / ODBC
- Interface for REST protocol: API-Gateway (Apache Apisix)

6.3.5 Governance Layer

- Policy Engine for data access, and fine-grained access control (RBAC/ABAC) and auditing:
    - Open Policy Agent (OPA)
    - Alternative: Apache Ranger for fine-grained access control
- Retention Manager: Enforces agreement-based TTL and deletion (e.g. implemented as batch data pipeline)

6.4 AI Processing Architecture

The AI processing layer SHALL be implemented as an orchestration service that retrieves data from the data lakehouse, formats it into prompts, invokes LLM inference via an OpenAI-compatible API, and exposes outputs via a RESTful API. This layer MAY support polling, event-driven triggers (e.g., via webhooks on data changes), or streaming integration to enable near-real-time enrichment. Security and governance SHALL align with Sections 9 and FR-DL-005 (e.g., anonymizing sensitive data in prompts).

7. Interfaces & Data Models

7.1 DSP Control Plane Interfaces

7.1.1 Catalogue Request

Endpoint: POST /dsp/catalog/request
Request Body:
{ "providerId": "did:web:provider.example", "filter": { "assetType": "iot.timeseries" }, "page": { "limit": 50, "cursor": null } }
Response (200 OK):
{ "datasets": [{ "id": "dataset:temp-01", "metadata": {...}, "offers": [{ "id": "offer:temp-01:read", "policySummary": {...} }] }], "nextCursor": null }

7.1.2 Contract Negotiation

Open Negotiation: POST /dsp/negotiations
{ "counterparty": "did:web:provider.example", "offerId": "offer:temp-01:read", "callbackAddress": "https://consumer.example/callback", "proposedTerms": { "purpose": "analytics", "duration": "PT2H" } }
Response (202 Accepted):
{ "negotiationId": "neg-123" }
Get State: GET /dsp/negotiations/neg-123
{ "id": "neg-123", "state": "FINALIZED", "agreementId": "agr-789" }

7.1.3 Transfer Process

Start Transfer: POST /dsp/transfers
{ "agreementId": "agr-789", "assetId": "dataset:temp-01", "format": "http-pull", "parameters": { "windowFrom": "2025-10-01T00:00:00Z", "windowTo": "2025-10-01T01:00:00Z" } }
Response (202 Accepted):
{ "transferId": "tp-456" }
Get Access: GET /dsp/transfers/tp-456
{ "id": "tp-456", "state": "COMPLETED", "access": { "url": "https://provider.example/api/data/temp-01?from=...&to=...", "token": "eyJ...", "expiresAt": "2025-10-01T03:00:00Z" } }

7.2 Data Plane Interfaces

7.2.1 HTTP Pull Access

GET https://provider.example/api/data/{assetId}?from={ts}&to={ts}&sig={hmac}
Authorization: Bearer {token}
Response: JSON array of sensor readings

7.2.2 Streaming Access

Connection Parameters:
{ "bootstrap": "kafka.provider.example:9092", "topic": "iot.dataset.temp-01.tp-456", "sasl": { "mechanism": "SCRAM-SHA-256", "username": "user_tp_456", "password": "..." }, "expiresAt": "2025-10-01T03:00:00Z" }

7.3 Data Lakehouse interfaces

7.3.1 Data Lakehouse ingestion interfaces

- Kafka consumer
- REST interface
- SFTP interface

7.3.2 Data Lakehouse access interfaces

- SQL interface (JDBC)
- REST interface
- Kafka consumer

7.4 Data Models

Table 3 IoT Entities Key Attributes (1)

Entity	Key Attributes
Participant	id (DID), name, endpoints, credentials[], roles[]
Dataset	id, metadata{}, schema_ref, retention_days, data_planes[]
Offer	id, dataset_id, policy{purpose, rate_limit, ttl, restrictions[]}
Negotiation	id, state, offer_id, consumer_id, provider_id, callback_address, history[], created_at, updated_at
Agreement	id, negotiation_id, terms{}, start_time, end_time, signed_at
Transfer	id, state, agreement_id, asset_id, format, parameters{}, access{}, created_at, expires_at
AI-Output	id, input_data, llm_model, output_text, timestamp

Table 4 IoT Entities Key Attributes (2)

Entity	Key Attributes
Sensor Measurement	fact_id, timestamp, asset_id, submodel_id, sensor_id state, offer_id, consumer_id, provider_id, callback_address, history[], cre
Agreement	id, negotiation_id, terms{}, start_time, end_time, signed_at
Transfer	id, state, agreement_id, asset_id, format, parameters{}, access{}, created_at, expires_at
AI-Output	id, input_data, llm_model, output_text, timestamp

7.5 Data Model Guidelines for the Data Lakehouse

The specific data modeling depends heavily on the individual use case. Nevertheless, there are some key design patterns for data modelling for a data lakehouse, which are listed below.

7.5.1 Follow the Medallion Architecture

This layered pattern (Bronze → Silver → Gold) is fundamental for organizing and modeling data in a lakehouse. Each layer builds trust and quality, creating a natural modeling progression from raw → refined → consumable.

Table 5 Data Layers in FAP IoT & AI

Layer	Pupose	Data Type	Modeling Focus
Bronze	Raw data ingestion	Unstructured, streaming, IoT	Schema discovery, metadata capture
Silver	Cleaned, standardized, enriched data	Structured & semi-structured	Conformed schema, keys, relationships
Gold	Curated, business-ready data	Analytical / aggregated	Dimensional models, KPIs, subject areas

7.5.2 Evolutionary Schema Design

- Lakehouses must handle dynamic, evolving data.
  This allows your model to stay stable even as data sources change over time.
- Use schema evolution features (supported in Delta Lake, Apache Iceberg, Hudi)

7.5.3 Dimensional Modeling for Analytical Layers

Even though data lakes are flexible, dimensional modeling (Kimball-style) remains powerful for analytics and BI. This pattern enables intuitive, performant dashboards and consistent KPIs.

- Use fact tables (transactions, sensor events, measurements)
- Use dimension tables (assets, time, location, customers)
- Create star or snowflake schemas in the Gold layer
- Apply surrogate keys to stabilize joins
- Store data in open formats (Parquet, Delta) for high-performance queries

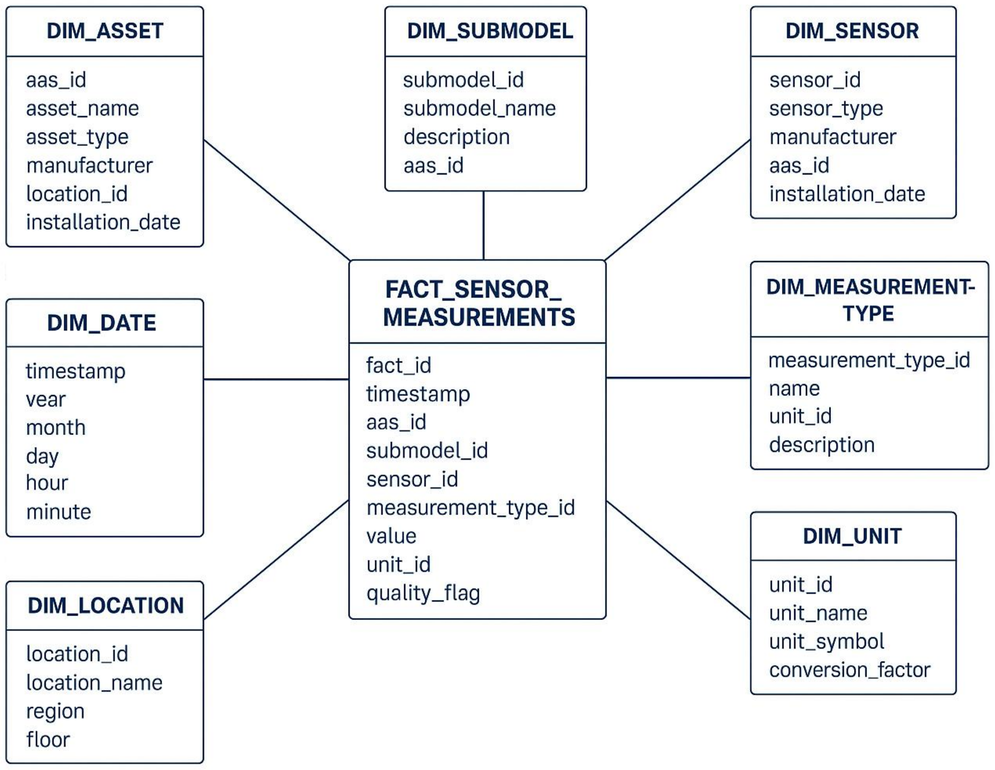


Figure 5 Star Schema for Sensor Measurements (Example)

7.5.4 Time-Series and IoT Data

Time-based modelling is essential in IoT and industrial lakehouses and optimizes performance for large-scale sensor data.

- Partition data by time intervals (e.g., daily/hourly folders) for efficient queries
- Use append-only design for streaming data (no in-place updates)
- Apply downsampling or summarization in the Silver or Gold layers (e.g., 1s → 1min → 1h)
- Keep metadata like device_id, timestamp, and status normalized
- Use windowing functions for time-based aggregations (Spark/Flink)

7.5.5 Physical Layout and Partitioning

Because lakehouse data lives on object storage, physical modeling matters. This pattern helps to balance storage cost, query speed and maintainability.

- Partition by high-cardinality fields like date, region, or device type
- Use Z-ordering / clustering (in Delta) or partition pruning (in Iceberg) for query efficiency
- Avoid over-partitioning (too many small files) — use compaction jobs
- Store data in columnar formats (Parquet) for analytics workloads

7.5.6 Multi-Modal Data Access

The data model must support multiple access pattern. Logical views or APIs tailored to different use cases, avoiding data duplication if possible

- SQL interface (JDBC)
- REST interface
- Access to aggregates and KPIs (Star schemas, Gaold Layer)
- Access to feature-level data (Denormalized views, Silver Layer)
- Access for Operational Systems (Stream processing, materialized views)

7.6 AI Processing Interfaces

- Data Lake Query: GET /api/data/{layer}/{assetId}?from={ts}&to={ts} (per FR-DL-006).
- LLM API: POST {configurable_base_url}/v1/chat/completions (e.g., https://api.ionos.com/inference-openai) Body: { "model": "llama-2-7b-chat", "messages": [{"role": "user", "content": "Summarize: [data]"}] } Response: { "choices": [{"message": {"content": "output"}}] }
- LLM Output API: GET /api/ai/outputs/{id} for JSON outputs;"

8. Security & Trust

8.1 Transport Security

- All DSP control plane endpoints MUST use TLS 1.2 or higher
- Kafka brokers MUST support SASL SCRAM-SHA-256 or mTLS
- HTTP data plane endpoints MUST validate HMAC signatures
- Certificates managed via cert-manager in Kubernetes deployments
- API calls to LLM service MUST use TLS 1.3.
- Node-RED flows MUST enforce data minimization (e.g., anonymize PII before LLM).
- Audit logs include LLM prompts/responses for traceability.

8.2 Identity & Authentication

- Organizations identified by W3C DIDs (did:web or did:key methods)
- Participant Verifiable Credentials issued per OIDC4VCI specification
- VC presentation via OIDC4VP in negotiation requests (optional for MVP)
- Token-based access for data plane (short-lived JWTs or HMAC tokens)
- Single Sign On to access Graphical User Interfaces using OAuth/OIDC using didweb identities

8.3 Authorization & Policy Enforcement

- Policy Agent evaluates usage constraints before transfer approval
- Rate limits enforced via Redis token bucket per agreement
- Purpose restrictions validated against VC claims
- Time-bounded access via token expiry and credential TTL
- Kafka ACLs scoped to consumer principals with auto-cleanup on expiry

- Policy enforcement SHALL include AI-specific constraints, such as limiting prompt data volume or restricting LLM usage to approved purposes (e.g., 'analytics' only)
- Policy Agent evaluates usage constraints when accessing data in the data lake

8.4 Data Protection

- Data at rest encrypted in object storage (S3 SSE or MinIO KMS)
- Data in transit encrypted via TLS/SASL
- Secrets stored in external KMS (HashiCorp Vault, OpenBoa or Secrets Manager)
- PII handling per GDPR data minimization principles

8.5 Audit & Observability

- All DSP operations logged with correlation IDs
- State transitions emit structured events (JSON) to event bus
- Audit trail includes negotiation lifecycle, transfer access grants, data plane access attempts
- Logs aggregated in OpenSearch stack, ELK stack or Loki for querying
- Metrics exported to Prometheus (request rates, latencies, error rates)
- Audit logs SHALL capture access to data lakehouse
- Audit logs SHALL capture LLM prompts, responses, and associated data lake queries for traceability and compliance.

9. Deployment & Operations

The client will provide all technical base line resources as pre-configured Kubernetes cluster with three dedicated ORCE deployments and Keycloack provider, as well as database and data lake as a service and AI Model hub.

9.1 Deployment Profiles (Exemplary)

9.1.1 Docker Compose (Development) or Kubernetes (K3S, KIND)

Single-host deployment for local development and testing:

- IoT/OT Zone
  - Backend (FastAPI connector)
  - Frontend (React UI)
  - Kafka
  - Mosquitto (MQTT)
  - MQTT-Kafka Bridge
  - PostgreSQL
  - Redis

- Enterprise IT Zone
  - Dashboard Service
  - Keycloak
  - OCM/PCM/AAS
  - CAT

- Cloud Zone
  - Message Broker (Kafka)
  - S3 Storage (MinIO)
  - Data Ingest (Apache NiFi)
  - Data Catalog (Apache Hive)
  - Data Processing (Spark)
  - Query Engine (Trino)
  - Data Access (Apache Apisix)
  - PostgreSQL

9.1.2 Kubernetes (Production)

Multi-zone, highly available deployment:

- Helm charts or Kubernetes Operators for all services
- Separate namespaces per trust zone
- Horizontal Pod Autoscaling for connector and ingest services
- Ingress with TLS termination
- External Secrets Operator for KMS integration

9.2 Operational Requirements

9.2.1 Availability

- Control plane: 99.9% uptime SLA
- Data plane: 99.95% uptime SLA
- Health checks at /health and /ready endpoints
- Circuit breakers for external dependencies

9.2.2 Scalability

- Connector scales horizontally (stateless)
- Kafka scales to 1000+ partitions
- PostgreSQL with read replicas
- Redis cluster for distributed caching

9.2.3 Monitoring & Alerting

- Prometheus metrics for all services
- Grafana dashboards for operational visibility
- Alertmanager for SLA breach notifications

10. Standards & Protocols

Table 6 Standards & Protocols

Standard	Purpose	Status
Eclipse DSP 1.0	Control plane protocol	MUST
W3C DID Core	Decentralized identifiers	MUST
W3C VC Data Model	Verifiable credentials	MUST

Standard	Purpose	Status
OIDC4VCI	Credential issuance	SHOULD
OIDC4VP	Credential presentation	SHOULD
DCP	Decentralized Claims Protocol	SHOULD
OAuth 2.0	Authorization framework	MUST
OIDC	Authentication protocol	
OpenAPI 3.0	API specification	SHOULD
JSON-LD	Linked data metadata	MAY
OpenAI API Compatible	LLM inference (e.g., IONOS)	SHOULD
JDBC	Java Database Access Protocol	MUST
MQTT	Message Queuing Telemetry Transport	MUST
REST	Representational State Transfer	MUST

11. Validation & Acceptance Criteria

11.1 Protocol Conformance

1. Eclipse DSP TCK[7] transfer suite passes with 100% success rate
2. All Data lake/ Protocol endpoints validate against OpenAPI specification

11.2 Functional Tests

3. End-to-end test: Catalog discovery → Negotiation (preset) → HTTP Pull transfer → Data landed in Bronze
4. End-to-end test: Catalog discovery → Negotiation (preset) → Kafka streaming → Data in Bronze Parquet
5. MQTT to Kafka bridge test: Publish to MQTT → Verify in Kafka topic within 5 seconds
6. Policy enforcement test: Rate limit exceeded returns 429 with Retry-After header
7. Token expiry test: Expired access token rejected with 401
8. End-to-End test data processing: ETL processes data: Bronze --> Silver --> Gold Layer
9. Data access test: Data can be queried by query engine

[7] https://github.com/eclipse-dataspacetck/dsp-tck

11.3 Security Tests

10. TLS certificate validation passes for all endpoints
11. Invalid VC presentation rejected during negotiation
12. Tampered HMAC signature rejected on HTTP data access
13. Kafka SASL authentication failure logged and access denied
14. Data access: Authorization rules are followed during data retrieval via JDBC interface
15. Data access: Authorization rules are followed during data retrieval via REST interface

11.4 Performance Tests

16. Catalog request responds < 500ms at p95
17. Negotiation completes within 5 seconds
18. HTTP data fetch completes < 2s for 1000 sensor readings
19. Kafka throughput sustains 10,000 msg/sec per topic
20. Data access: data retrieval via JDBC interface < 1 sec
21. Data access: data retrieval via REST interface < 1 sec

11.5 Operational Tests

22. Docker Compose deployment starts successfully
23. Kubernetes Helm chart deploys without errors
24. Health checks report healthy within 30 seconds of startup
25. Rolling update completes without downtime
26. Audit logs queryable for all DSP operations within 1 minute
27. End-to-end test: Data lake query → LLM inference → Visualization update within 2s.
28. Performance test: Modify data in lake → API reflects changes < 10s.
29. Security test: Unauthorized API call rejected.
30. Performance test: Handle 1 req/min with < 5s latency at p95.
31. LLM response matches expected structure (e.g., JSON) and accuracy (>90% on sample anomalies).

12. Requirements Traceability Matrix

Once the requirements highlighted in Table 7 are fulfilled, all other requirements marked as MUST in this overall specification document are automatically inherited and considered part of the validation process. In other words, meeting these top-level requirements ensures that the dependent MUST, SHOULD etc. requirements are also respected.

Table 7 Requirements Matrix

Req ID	Component	Interface/API	Test Case
FR-DSP-001	Catalogue Service	POST /dsp/catalog/request	TC-CAT-001: Catalogue discovery
FR-DSP-002	Negotiation Service	POST /dsp/negotiations	TC-NEG-001: Contract flow
FR-DSP-003	Transfer Service	POST /dsp/transfers	TC-TRX-001: Transfer init
FR-DP-001	HTTP Data Plane	GET /api/data/{asset}	TC-HTTP-001: Pull data

Req ID	Component	Interface/API	Test Case
FR-DP-002	Kafka Data Plane	Kafka Consumer API	TC-KFK-001: Stream data
FR-DP-003	MQTT Bridge	N/A (Internal)	TC-MQTT-001: Bridge flow
FR-IAM-001	Identity Hub	POST /trust/credentials	TC-IAM-001: VC issuance
FR-IAM-002	DCP Verifier	Internal verification	TC-DCP-001: VP verification
FR-DL-001	HTTP Ingest	Internal service	TC-ING-001: HTTP land
FR-DL-002	Kafka Ingest	Consumer service	TC-ING-002: Kafka land
FR-DL-003	Lake Layering	ETL jobs	TC-LAKE-001: Bronze→Gold
FR-AI-001	Orchestrator	GET /api/data/{layer}/{assetId}	TC-AI-001: Data retrieval
FR-AI-002	LLM Inference Service	POST /v1/chat/completions	TC-AI-002: Inference flow

13. Appendices

Appendix A: Glossary

Table 8 Terms & Abbreviations

Term	Full Name	Definition
DSP	Dataspace Protocol	Eclipse Foundation protocol for interoperable data exchange
DCP	Decentralized Claims Protocol	Protocol for conveying organizational identities and trust
DID	Decentralized Identifier	W3C standard for decentralized digital identities
VC	Verifiable Credential	W3C standard for tamper-evident credentials
VP	Verifiable Presentation	Package of one or more VCs for presentation to verifier
TCK	Dataspace Protocol Technology Compatibility Kit	Test suite for protocol conformance
OT	Operational Technology	Hardware and software controlling industrial operations
SASL	Simple Authentication and Security Layer	Framework for authentication in protocols
SCRAM	Salted Challenge Response Authentication Mechanism	Password-based authentication

Appendix B: References

1. Eclipse Dataspace Protocol Specification 1.0 - https://projects.eclipse.org/projects/technology.dataspace-protocol-base
2. Eclipse Dataspace Components (EDC) Documentation - https://eclipse-edc.github.io/docs
3. Eclipse DSP TCK Repository - https://github.com/eclipse-dataspacetck/dsp-tck
4. W3C DID Core Specification - https://www.w3.org/TR/did-core/
5. W3C Verifiable Credentials Data Model - https://www.w3.org/TR/vc-data-model/
6. OpenID Connect for Verifiable Credential Issuance (OIDC4VCI) - https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html
7. OpenID Connect for Verifiable Presentations (OIDC4VP) - https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
8. RFC 2119: Key words for use in RFCs to Indicate Requirement Levels https://www.rfc-editor.org/rfc/rfc2119
9. OpenAI Chat Completions API Specification - https://platform.openai.com/docs/api-reference/chat

Appendix C: Alignment with FAP IoT & AI

This specification realizes the FAP IoT & AI objectives by providing:

- **Unified data pipeline** from sensor to dashboard across federated ecosystems
- **Standardized data transfer** via DSP control plane and HTTP/Kafka data planes
- **Cloud-enabled aggregation** with Bronze/Silver/Gold lake architecture
- **Trust services integration** with DCP, W3C DIDs, and Verifiable Credentials
- **Full abstraction layers** for data gathering, transfer, aggregation, and visualization