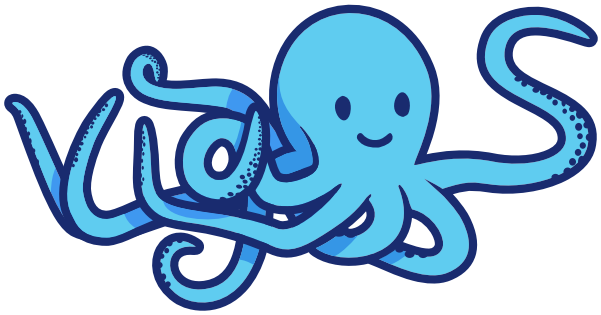

vigOS Development Plan

Versatile Instrumentation and Governance Operating Stack

1 Executive Summary

vigOS is a reusable operating stack for research instruments and data workflows, designed to make FAIR (Findable, Accessible, Interoperable, Reusable) data practices automatic at the point of data creation rather than an afterthought at publication.

2 Problem Statement

- Researchers spend 10 hours/week on data & code management
- 53% don't know any metadata standards; 30% don't know what they are
- 62% rely on manual file renaming for versioning
- Each new instrument reinvents storage, metadata, governance, and analysis infrastructure
- FAIR compliance typically happens at publication—too late to be reliable

3 Vision & Goals

3.1 Primary Goal

Build a modular, reusable infrastructure stack that enables FAIR-by-design research workflows, reducing boilerplate development time by 50% and freeing researchers from manual data management overhead.

3.2 Key Principles

- **FAIR at source** — metadata captured automatically at acquisition
- **Governance encoded** — policies in code, not spreadsheets
- **Reproducible by default** — containerized environments
- **Build once, reuse everywhere** — modular, composable components

4 User Personas & Stories

4.1 Dev User (Instrument Developer)

A developer/engineer/scientist building & operating(research) instruments, data acquisition systems, with vigOS as the foundation. As a developer I want to ...

4.1.1 Onboarding & Setup

- bootstrap a new instrument software from a template → not waste weeks on boilerplate
- have clear documentation and examples on the custom components I need to build, drivers, UI, etc. → understand how to integrate hardware with vigOS

4.1.2 Development & Integration

- pre-built adapters for common DAQ hardware functions, i.e. communication protocols, logging, slow control (telling devices what to do) and acquisition (writing measurement data to disk) → focus on unique instrument logic
- define a standardized metadata capture → automatic FAIR-compliant data output
- reproducible and easy to deploy builds → consistent behaviour across environments and or other instruments
- seamlessly upgrade/roll-back/hot-reload during development and upgrades to limit downtime → fast iteration without full local builds and tests
- be informed if tools & packages are outdated and need to be updated → keep the software stack up to date

4.1.3 Testing & Deployment

- use CI/CD pipelines configured easily → code tested and deployed hands-free
- have a staging environment available with integration tests → validate changes before production
- have rollback capabilities → quick recovery from failed deployments

4.1.4 Monitoring & Debugging

- access logs, metrics, and traces early in the development process of the instrument → debug instrument issues from the get-go
- have pro-active health alerting → notified of problems before users report/notices them

4.1.5 Compliance & Governance

- have documentation about the code and instrument that complies with regulatory requirements, ISOs, CE,.. → auto-generated & supported from CI/CD through doc-as-code, infra-as-code and test-as-code

4.1.6 Operations

- have a clear and easy to understand up-to-date documentation on the operations of the instrument → auto-generated from CI/CD
- define workflows and access control for what users can execute an do with the instrument having set permissions and guardrails
- be able to have workflows use different 'versions' of parts of the software stack → have consistency across measurement campaigns
- offer a way for users to access their acquired data and metadata in a way that is easy to understand and use and integrates with other tools and workflows
- have remote access to the instrument and the software stack for maintenance and upgrades
- remote monitoring of deployed system

4.2 Dev User Medical Device (Regulated Development)

A developer building instruments under medical device regulations (MDR, FDA, ISOs, CE,..) with strict compliance requirements. As a medical device developer I want everything the device developer wants, but also …

4.2.1 Compliance & Documentation

- Immutable audit logs of all code changes → traceability for regulatory audits

- Automated design history documentation → reduced manual compliance overhead
- Version-controlled configuration-as-code → every system state traceable and reproducible
- Requirements linked to test results → demonstrate validation coverage

4.2.2 Risk Management

- FMEA simulation containers → systematic failure mode analysis
- Safety-critical constraints defined in code → violations caught automatically
- Emergency handling workflows built-in → critical failures trigger defined responses

4.2.3 Validation & Verification

- Reproducible build environments → prove bit-for-bit identical deployments
- Automated IQ/OQ/PQ (Installation/Operational/Performance Qualification) test suites → consistent, repeatable validation
- Data integrity checks at every step → prove data not corrupted or tampered

4.2.4 Access Control & Security

- Role-based access control with audit trails → demonstrate who accessed what and when
- Separation of duties enforced by system → no single person makes unauthorized changes
- Secure credential management → sensitive keys never exposed in code or logs

4.3 Admin (System Administrator)

Responsible for user and instrument management during operations. As an admin I want to ...

4.3.1 Instrument Management

- define resource quotas and limits for each instrument → prevent instruments from starving others/ work with back-up systems/data ingestors
- be informed about the status of the instrument and the software stack → monitor system health at a glance

4.3.2 User & Access Management

- manage user authentication and authorization with SSO integration with local fallbacks → users authenticate with institutional credentials but operation of instrument is not depending on permanent access
- Role templates → new users get appropriate permissions automatically
- User activity auditing and logging → investigate security incidents & report usage
- Immediate access revocation → departing users lose access without delay

4.3.3 Monitoring & Alerting

- Unified dashboard for all instruments → monitor system health at a glance
- Alerting with escalation policies → critical issues reach the right people
- Capacity planning metrics → anticipate when to scale infrastructure
- Log aggregation across services → correlate events for troubleshooting

4.3.4 Backup & Recovery

- Automated backups with retention policies → data protected without manual work
- Tested restore procedures → confidence recovery works when needed
- Disaster recovery → having encompassing logging and being able to restore service quickly after major failures

4.3.5 Security & Compliance

- Automated vulnerability scanning → alerted to security issues in dependencies
- Network segmentation → instruments isolated from each other by default

- TLS everywhere → data in transit encrypted

4.4 Operator (Instrument User / Researcher)

End user who operates instruments and runs experiments without needing deep infrastructure knowledge.

4.4.1 Running Experiments

- Simple interface to start/stop/interrupt acquisition → focus on experiment, not software
- Real-time data quality feedback → know immediately if something is wrong
- Unable to 'break' the instrument by doing something wrong → have guardrails and safety mechanisms in place
- Be able to use recipes to run experiments → have a way to run experiments with a set of parameters and conditions that are automatically executed
- Annotate runs with experiment metadata → find and understand data later
- Pause and resume acquisition → handle interruptions without losing data

4.4.2 Data Access & Management

- Searchable dataset browser → find past experiments quickly
- Automatic data organization → no need to invent folder structures
- Share datasets with collaborators → work together on analysis
- Download in standard formats → use preferred analysis tools

4.4.3 Analysis & Reproducibility

- Pre-configured analysis environments → start analyzing without setup hassles
- Analysis linked to source data → automatic provenance
- Re-run past analyses → verify or extend previous results
- Analysis templates for common workflows → don't start from scratch

4.4.4 Publication & Export

- One-click export to data repositories → publishing data is trivial
- Automatic DOI/UUID generation → data is citable
- Bundle data, code, and environment together → others can reproduce work

4.4.5 Training & Support

- Contextual help and documentation → solve problems without waiting for support
- Example workflows → learn best practices
- Clear error messages → understand what went wrong and where to get help

5 Functional Requirements

For the medical instrument purpose, each requirement should be testable and verifiable.

5.1 Instrument Layer — Data Acquisition

5.1.1 Core Acquisition

- The system shall provide a standardized interface for starting, stopping, pausing, and resuming data acquisition.
- The system shall write acquired data to persistent storage in real-time with configurable buffering strategies.
- The system shall support configurable data formats (HDF5, Zarr, raw binary, etc.) through format adapters.

- The system shall timestamp all acquired data with synchronized, synchronized to a common time source, traceable time sources (NTP/PTP).
- The system shall support parallel acquisition from multiple data sources/channels.
- The system shall enforce data rate limits to prevent storage overflow.

5.1.2 Metadata Capture

- The system shall automatically capture instrument configuration at acquisition start.
- The system shall embed metadata directly into data files (single-source-of-truth).
- The system shall register datasets to configured metadata repositories upon acquisition completion.
- The system shall support user-defined metadata fields with validation rules.
- The system shall generate unique identifiers (UUID) for each acquisition run.

5.1.3 Error Handling — Acquisition

- The system shall detect hardware communication failures and log them with timestamps and context.
- The system shall attempt automatic reconnection to hardware after transient failures with configurable retry policies.
- The system shall preserve all data acquired before a failure occurs (no silent data loss).
- The system shall notify operators of acquisition errors in real-time through configured channels (UI, alerts).
- The system shall classify errors by severity (info, warning, error, critical) and respond accordingly.
- The system shall log the complete error context (stack trace, system state, recent events) for post-mortem analysis.
- The system shall support graceful degradation — continue acquiring from functioning channels when one fails.
- The system shall validate data integrity (checksums) during and after acquisition.
- The system shall detect and flag data quality anomalies (gaps, outliers, saturation) in real-time.

5.1.4 Hardware Abstraction

- The system shall provide protocol adapters for common communication interfaces (TCP/IP, serial, USB, GPIB, etc.).
- The system shall abstract hardware-specific details behind a unified device driver interface.
- The system shall support device discovery and enumeration where hardware allows.
- The system shall provide slow control (command/response) interfaces for device configuration.

5.2 Instrument Layer — Slow Control & Monitoring

5.2.1 Device Control

- The system shall provide interfaces for sending commands to connected devices.
- The system shall validate commands against device capabilities before execution.
- The system shall log all commands sent to devices with timestamps and operator identity.
- The system shall support recipes/scripts for automated experiment sequences.
- The system shall enforce operator permissions before executing commands.

5.2.2 Real-time Monitoring

- The system shall expose real-time metrics from all connected devices.
- The system shall provide configurable dashboards for instrument status visualization.
- The system shall support threshold-based alerting on monitored values.
- The system shall retain historical metrics for trend analysis with configurable retention.

5.3 Governance Layer

5.3.1 Authentication & Authorization

- The system shall authenticate users via SSO integration.
- The system shall support local fallback authentication when SSO is unavailable.
- The system shall implement role-based access control (RBAC) with configurable roles and permissions.
- The system shall enforce permissions at the API level — no client-side-only enforcement.
- The system shall support immediate access revocation without requiring user logout.
- The system shall enforce session timeouts and re-authentication policies.

5.3.2 Audit & Traceability

- The system shall log all user actions with timestamp, user identity, action, and outcome.
- The system shall store audit logs in append-only, tamper-evident storage.
- The system shall retain audit logs according to configurable retention policies (minimum 7 years for regulated).
- The system shall log all configuration changes with before/after values.
- The system shall provide audit log export in standard formats for compliance reporting.

5.3.3 Workflow Management

- The system shall support defining workflows as code.
- The system shall enforce workflow steps and approvals before certain actions (e.g., production deployment).
- The system shall version workflows and maintain history of changes.
- The system shall support workflow-specific access controls (who can execute, approve, abort).

5.4 Analysis Layer

5.4.1 Environment Management

- The stack shall provide reproducible analysis environments with pre-installed tools.
- The system shall version analysis environments and allow pinning to specific versions.
- The system shall support reproducible environment creation from specification files.
- The system shall isolate analysis environments from production acquisition systems.

5.4.2 Data Access

- The system shall provide a searchable catalog of all acquired datasets.
- The system shall enforce data access permissions consistent with governance policies.
- The system shall support data access via APIs and file system mounts.
- The system shall track data lineage — link derived data to source data.

5.4.3 Provenance

- The system shall record analysis provenance (code version, environment, parameters, inputs, outputs).
- The system shall enable re-execution of past analyses from stored provenance.
- The system shall link analysis results to source data immutably.

5.5 Publication Layer

5.5.1 Export & Packaging

- The system shall bundle data, metadata, code, and environment specifications for export.
- The system shall generate repository-ready packages.

- The system shall validate export packages against target repository requirements.
- The system shall support automated submission to configured repositories (e.g. ETH Research Collection, etc.).

5.5.2 Identifiers & Citation

- The system shall mint DOIs for published datasets via configured providers.
- The system shall generate citation metadata in standard formats (DataCite, Dublin Core).
- The system shall link DOIs to internal UUIDs for traceability.

5.6 Infrastructure & Operations

5.6.1 Deployment

- The system shall support infrastructure-as-code for reproducible deployments.
- The system shall support rolling updates with zero-downtime for non-acquisition components.
- The system shall support rollback to previous versions.

5.6.2 Observability

- The system shall expose metrics.
- The system shall emit structured logs (JSON) to centralized logging.
- The system shall propagate distributed traces across service boundaries.
- The system shall provide health check endpoints for all services.

5.6.3 Backup & Recovery

- The system shall perform automated backups of configuration and metadata on configurable schedules.
- The system shall support point-in-time recovery for databases.
- The system shall test backup integrity automatically.
- The system shall document and test disaster recovery procedures.

5.6.4 Security

- The system shall encrypt data in transit.
- The system shall encrypt sensitive data at rest.
- The system shall scan code and infrastructure images for known vulnerabilities.
- The system shall support network segmentation between instruments and other systems.
- The system shall rotate credentials and secrets automatically.

6 System Design

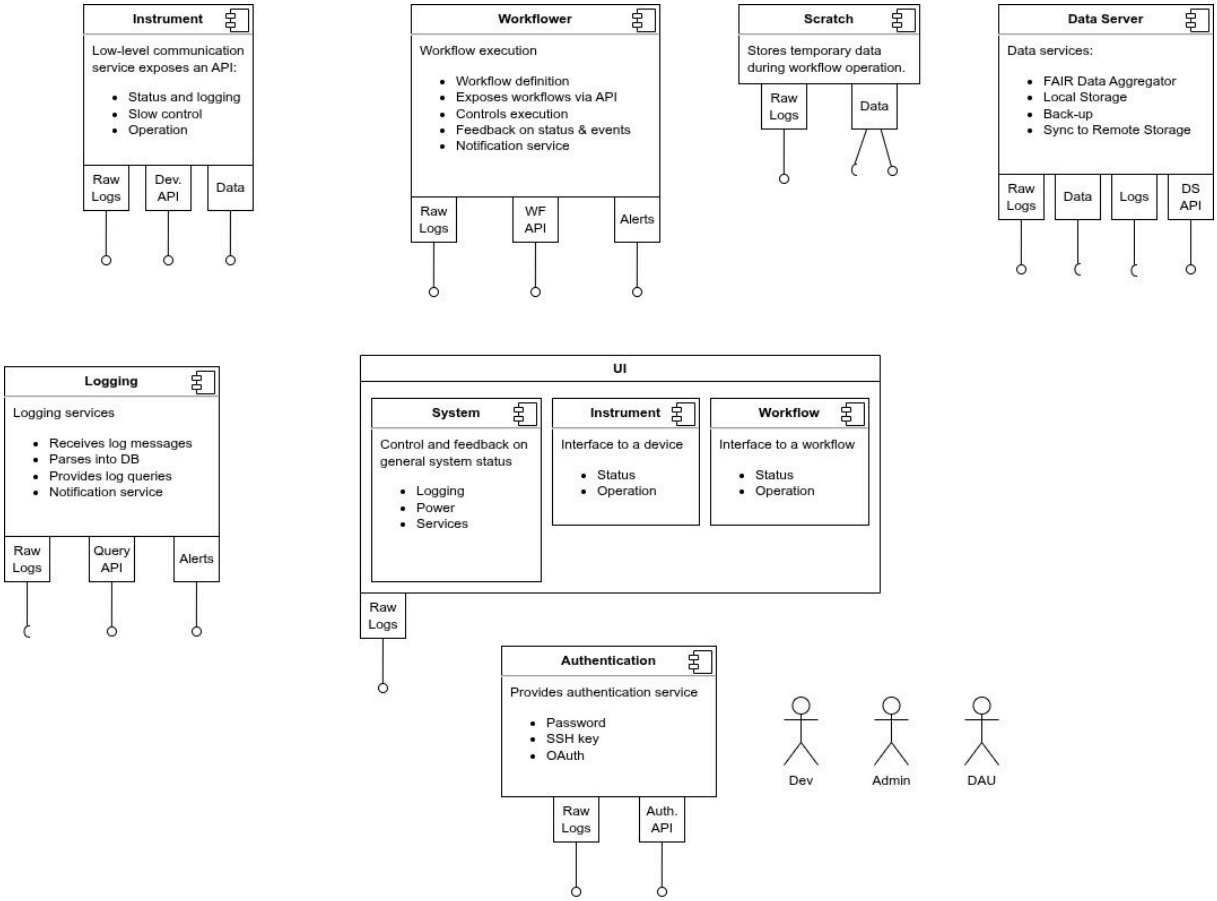

Figure 1: vigOS system architecture overview

The system is decomposed into loosely-coupled components, each with focused responsibilities. Components communicate via well-defined APIs and message passing.

6.1 Instrument Layer Components

6.1.1 DAQ Engine

Purpose: Core data acquisition — the heartbeat of the instrument.

Responsibilities:

- Start, stop, pause, resume acquisition runs
- Buffer incoming data and write to storage backends
- Enforce data rate limits and backpressure
- Timestamp data with synchronized time sources
- Coordinate parallel acquisition from multiple channels
- Detect and handle acquisition failures (reconnect, graceful degradation)
- Validate data integrity (checksums) during acquisition
- Emit acquisition events (started, paused, completed, failed) to message bus

Interfaces:

- Acquisition Control API (start/stop/pause/resume)
- Receives raw data from Device Drivers
- Publishes to Storage Adapter
- Emits events to Message Bus

—

6.1.2 Device Driver Manager / Library

Purpose: Hardware abstraction — uniform interface to heterogeneous devices.

Responsibilities:
- Load and manage device driver plugins
- Provide unified API regardless of underlying protocol (TCP, serial, USB, GPIB, etc.)
- Handle device discovery and enumeration
- Manage device connection lifecycle (connect, disconnect, reconnect)
- Translate device-specific protocols to common data formats
- Report device health and status

Interfaces:
- Device Driver Plugin API (for driver implementations)
- Unified Device API (consumed by DAQ Engine, Slow Control)
- Device Status API (consumed by Monitoring Service)

—

6.1.3 Slow Control Service

Purpose: Device command and control — tell instruments what to do.

Responsibilities:
- Send commands to devices via Device Driver Manager
- Validate commands against device capabilities and user permissions
- Queue and sequence commands
- Execute recipes/scripts for automated sequences
- Log all commands with timestamps, operator, and outcome
- Handle command timeouts and failures

Interfaces:
- Command API (send commands, check status)
- Recipe Execution API (load, run, abort recipes)
- Consumes Device Driver Manager
- Publishes command events to Audit Service

—

6.1.4 Metadata Service

Purpose: FAIR at source — capture and manage metadata automatically.

Responsibilities:
- Capture instrument configuration at acquisition start
- Embed metadata into data files (single-source-of-truth)
- Validate user-defined metadata against schemas
- Generate UUIDs for acquisition runs
- Register datasets to external metadata repositories
- Link metadata to data files immutably

Interfaces:
- Metadata Capture API (automatic and manual entry)
- Schema Management API (define, validate schemas)
- Repository Registration API (push to external catalogs)
- Subscribes to DAQ Engine events

—

6.2 Governance Layer Components

6.2.1 Auth Service

Purpose: Identity and authentication — who are you?

Responsibilities:
- Authenticate users via SSO (SAML/OIDC)
- Provide local fallback authentication when SSO unavailable
- Issue and validate session tokens
- Enforce session timeouts and re-authentication
- Support immediate session revocation

Interfaces:
- Login/Logout API
- Token Validation API (consumed by all services)
- SSO Provider Integration

—

6.2.2 Access Control Service

Purpose: Authorization — what are you allowed to do?

Responsibilities:
- Implement role-based access control (RBAC)
- Manage roles, permissions, and role assignments
- Enforce permissions at API boundaries
- Provide permission checks for other services
- Support workflow-specific access controls

Interfaces:
- Permission Check API (can user X do action Y on resource Z?)
- Role Management API (CRUD roles, assign users)
- Consumes Auth Service for identity

—

6.2.3 Audit Service

Purpose: Traceability — what happened, when, by whom?

Responsibilities:
- Log all user actions with timestamp, identity, action, outcome
- Log all configuration changes with before/after values
- Store logs in append-only, tamper-evident storage
- Enforce retention policies
- Export audit logs for compliance reporting

Interfaces:
- Audit Log API (write events)
- Audit Query API (search, filter, export)
- Subscribes to events from all services via Message Bus

—

6.2.4 Workflow Engine

Purpose: Process orchestration — define and enforce how work gets done.

Responsibilities:
- Define workflows as code
- Execute workflow steps with approval gates
- Enforce step ordering and prerequisites
- Track workflow state and history
- Version workflows and maintain change history

Interfaces:
- Workflow Definition API (create, update, version)
- Workflow Execution API (start, approve, abort) passes commands to Slow Control Service
- Workflow Status API (current state, history)

—

6.3 Analysis Layer Components

6.3.1 Environment Manager

Purpose: Reproducible compute — consistent analysis environments.

Responsibilities:
- Provision containerized analysis environments
- Version environments and support pinning
- Build environments from specification files (Dockerfile, conda, etc.)
- Isolate analysis environments from production systems
- Manage resource allocation (CPU, memory, GPU)

Interfaces:
- Environment Provisioning API (create, destroy)
- Environment Registry API (list, version, pin)
- Specification Build API (build from definition)

—

6.3.2 Data Catalog

Purpose: Discoverability — find and understand your data.

Responsibilities:
- Index all acquired datasets with searchable metadata
- Provide search and browse interfaces
- Enforce data access permissions
- Track data lineage (source ⟶ derived relationships)
- Organize datasets automatically (projects, experiments, runs)

Interfaces:
- Search API (query by metadata, full-text)
- Browse API (hierarchical navigation)
- Lineage API (trace data provenance)
- Consumes Metadata Service, Access Control Service

—

6.3.3 Provenance Tracker

Purpose: Reproducibility — trace how results were produced.

Responsibilities:

- Record analysis provenance (code, environment, parameters, inputs, outputs)
- Link analysis results to source data immutably
- Enable re-execution of past analyses
- Generate provenance reports

Interfaces:
- Provenance Recording API (log analysis runs)
- Provenance Query API (trace lineage)
- Re-execution API (replay analysis)

—

6.4 Publication Layer Components

6.4.1 Export Service

Purpose: Packaging — bundle everything for sharing.

Responsibilities:
- Bundle data, metadata, code, and environment specs
- Generate repository-ready packages (BagIt, RO-Crate, etc.)
- Validate packages against target repository requirements
- Support configurable export profiles

Interfaces:
- Export API (create bundle from selection)
- Validation API (check against repository requirements)
- Profile Management API (define export configurations)

—

6.4.2 Repository Connector

Purpose: Publication — push to external repositories.

Responsibilities:
- Submit packages to configured repositories (ETH Research Collection, Zenodo, etc.)
- Handle repository authentication and authorization
- Track submission status and outcomes
- Support draft/review/publish workflows

Interfaces:
- Submission API (submit to repository)
- Repository Configuration API (add, configure repos)
- Status API (track submission progress)

—

6.4.3 Identifier Service

Purpose: Citation — make data findable and citable.

Responsibilities:
- Generate internal UUIDs for all datasets
- Mint DOIs via configured providers
- Generate citation metadata (DataCite, Dublin Core)
- Link external identifiers to internal UUIDs

Interfaces:

- UUID Generation API
- DOI Minting API
- Citation Metadata API

—

6.5 Infrastructure Components

6.5.1 Storage Adapter

Purpose: Storage abstraction — uniform access to diverse backends with local buffering.

Responsibilities:
- Abstract storage backends (local filesystem, NFS, S3, object storage)
- Handle data format adapters (HDF5, Zarr, raw binary)
- Manage storage lifecycle (tiering, archival)
- Provide consistent API regardless of backend

Scratch/Buffer Storage:
- Manage local scratch storage on acquisition nodes for high-speed buffering
- Buffer incoming data when network/remote storage unavailable or too slow
- Spooling: async transfer from scratch to permanent storage
- Monitor scratch capacity and trigger alerts before overflow
- Guarantee no data loss during transfer (checksum verification)
- Support configurable retention and cleanup policies for scratch
- Handle scratch-to-permanent promotion with integrity verification

Interfaces:
- Read/Write API (stream-oriented and file-oriented)
- Storage Configuration API (mount backends, configure scratch paths)
- Lifecycle API (tier, archive, delete)
- Scratch Status API (capacity, pending transfers, health)
- Transfer API (manual trigger, retry failed transfers)

—

6.5.2 Message Bus

Purpose: Event backbone — decouple components through async messaging.

Responsibilities:
- Publish/subscribe event distribution
- Guarantee message delivery (at-least-once)
- Support event replay for recovery
- Route events to appropriate consumers

Interfaces:
- Publish API (emit events)
- Subscribe API (register handlers)
- Event types: acquisition, command, audit, alert, status

—

6.5.3 Config Service

Purpose: Centralized configuration — single source of truth for settings.

Responsibilities:

- Store and distribute configuration to all components
- Support environment-specific overrides (dev, staging, prod)
- Version configuration and track changes
- Validate configuration against schemas
- Notify components of configuration changes

Interfaces:
- Config Read API (get configuration)
- Config Write API (update, with audit)
- Config Watch API (subscribe to changes)

—

6.5.4 Central Logging Service

Purpose: Unified logging — all logs in one place, queryable and correlated.

Responsibilities:
- Aggregate logs from all components and instruments
- Accept structured logs (JSON) with consistent schema
- Index and store logs with configurable retention
- Correlate logs across services via trace IDs
- Support real-time log tailing and historical queries
- Forward logs to external SIEM systems if required
- Handle log volume spikes without data loss (buffering)
- Enforce log retention policies (regulatory compliance)

Log Flow:
- Components emit structured JSON logs to local agent
- Local agent buffers and forwards to central collector
- Central collector indexes, stores, and makes searchable
- Offline/disconnected operation: local buffer until reconnection

Interfaces:
- Log Ingestion API (push logs)
- Log Query API (search, filter, aggregate)
- Log Stream API (real-time tailing)
- Retention Policy API (configure per-source retention)

—

6.5.5 Observability Stack

Purpose: Visibility — metrics, traces, dashboards.

Responsibilities:
- Collect and store metrics (Prometheus)
- Collect and correlate distributed traces (Tempo/OpenTelemetry)
- Provide unified dashboards (Grafana)
- Health check aggregation across all services
- Anomaly detection on metrics

Interfaces:
- Metrics Scrape Endpoints (per component)
- Trace Propagation (OpenTelemetry context)
- Dashboard API

- Health Aggregation API

—

6.5.6 Alert Manager

Purpose: Proactive notification — know before users complain.

Responsibilities:
- Evaluate alert rules against metrics and events
- Route alerts to appropriate channels (email, Slack, PagerDuty)
- Manage alert silencing and acknowledgment
- Escalate unacknowledged alerts
- Track alert history

Interfaces:
- Alert Rule API (define thresholds and conditions)
- Alert Status API (current alerts, history)
- Notification Channel API (configure destinations)

—

6.5.7 Backup Service

Purpose: Data protection — automated, verified, recoverable backups.

Responsibilities:
- Schedule and execute automated backups (configuration, metadata, databases)
- Support multiple backup targets (local, remote, cloud)
- Implement backup rotation and retention policies
- Verify backup integrity automatically (restore tests)
- Support point-in-time recovery for databases
- Track backup history and status
- Alert on backup failures
- Support manual backup triggers
- Encrypt backups at rest

Backup Scope:
- Configuration: all system and instrument configurations
- Metadata: dataset metadata, provenance records, audit logs
- Databases: catalog, workflow state, user data
- Note: Raw acquisition data backed up via Storage Adapter replication/archival

Interfaces:
- Backup Schedule API (configure schedules, retention)
- Backup Trigger API (manual backup)
- Restore API (list backups, restore to point-in-time)
- Backup Status API (history, health, last successful)

—

6.5.8 Documentation Service

Purpose: Living documentation — auto-generated, versioned, always current.

Responsibilities:
- Generate documentation from code (doc-as-code)
- Build API documentation from OpenAPI/schemas

- Generate instrument operation manuals from configuration
- Version documentation alongside code releases
- Serve documentation via web interface
- Support full-text search across all docs
- Generate compliance documentation (design history, validation reports)
- Track documentation coverage and freshness

Documentation Types:
- Developer docs: API references, SDK guides, architecture
- Operator docs: Instrument manuals, SOPs, troubleshooting guides
- Admin docs: Deployment guides, runbooks, configuration references
- Compliance docs: Design history files, validation protocols, audit reports

Doc-as-Code Flow:
- Source: Markdown/RST in code repos, OpenAPI specs, config schemas
- Build: CI/CD generates docs on every commit
- Publish: Versioned docs deployed automatically
- Link: Docs linked to specific code/config versions

Interfaces:
- Doc Build API (trigger builds)
- Doc Serve API (web access, search)
- Doc Version API (list versions, compare)
- Coverage API (what's documented, what's stale)

—

6.5.9 Test & Simulation Service

Purpose: Verification & validation — FMEA, integration testing, failure simulation.

Responsibilities:
- Execute integration tests against staging/test environments
- Run automated IQ/OQ/PQ qualification test suites
- Simulate failure modes for FMEA analysis
- Inject faults to verify error handling (chaos engineering)
- Verify safety-critical constraints are enforced
- Generate test reports linked to requirements (traceability matrix)
- Track test coverage across functional requirements
- Schedule and execute regression test suites

FMEA Support:
- Define failure modes and expected system responses
- Simulate hardware failures (disconnection, timeout, corrupt data)
- Simulate infrastructure failures (network, storage, service crashes)
- Verify graceful degradation behaviors
- Document failure mode → detection → response → recovery paths
- Generate FMEA reports for regulatory compliance

Integration Testing:
- Spin up isolated test environments (containers)
- Mock hardware devices for reproducible testing
- End-to-end workflow testing (acquisition → storage → analysis → export)
- Performance/load testing (verify throughput requirements)

- Regression testing on every deployment

Safety Constraint Verification:
- Define safety constraints as executable checks
- Continuous verification against running system
- Alert on constraint violations
- Block operations that would violate safety constraints
- Audit trail of all constraint checks

Interfaces:
- Test Execution API (run suite, run single test)
- Test Report API (results, coverage, traceability)
- FMEA Simulation API (inject faults, observe responses)
- Constraint Definition API (define, enable, disable checks)
- Mock Device API (configure simulated hardware)

—

6.6 User Interface Components

6.6.1 Operator UI

Purpose: Experiment interface — run instruments without deep technical knowledge.

Responsibilities:
- Start/stop/monitor acquisition runs
- Display real-time data quality feedback
- Annotate runs with metadata
- Execute recipes and workflows
- Browse and access acquired data
- Clear error display with guided resolution

Consumes: DAQ Engine, Slow Control, Metadata Service, Data Catalog, Workflow Engine

—

6.6.2 Admin UI

Purpose: System management — configure and monitor the platform.

Responsibilities:
- User and role management
- Instrument registration and configuration
- Resource quota management
- System health dashboard
- Audit log viewer
- Backup and recovery controls

Consumes: Auth Service, Access Control, Config Service, Observability Stack, Audit Service

—

6.6.3 Developer Tools

Purpose: Development experience — build and debug instrument software.

Responsibilities:
- Project scaffolding from templates
- Local development environment setup

- Log and trace viewer
- Device driver debugging
- CI/CD pipeline status
- Documentation browser

Consumes: Config Service, Observability Stack, Device Driver Manager

7 MVP

Operator