ePIC Workflow Management System

A rendering of Read the Docs as of July 11, 2026

[ToC]

Table of contents

[ToC]

1. ePIC Workflow Management System

The ePIC Workflow Management System (WFMS) provides the shared workflow, data-management, monitoring, automation, and operations layer for ePIC computing.

The system spans post-DAQ streaming processing, global production campaigns, validation workflows, calibration, distributed CI, and analysis workflow support. These are workflow domains on one common distributed platform rather than separate systems. Two implementation fronts operate today — the streaming workflow testbed and the epicprod automated production system — building toward the full model.

1.1 Documentation Structure

- **Foundations** defines the WFMS scope, requirements basis, streaming computing model alignment, computing use cases, Echelon model, and butterfly model
- **Concepts** defines the working vocabulary of the system: streaming data units, configuration and identity, requests, campaign tasks, campaigns, and their lifecycles
- **Architecture** describes the overall WFMS design: common system structure, data/control flow, human-in-the-loop automation, AI integration
- **WFMS Platform** covers the realization of the architecture with specific technology and implementation choices
- **Streaming Workflows** documents post-DAQ workflows during datataking
- **Production System** documents the epicprod automated production system
- **Physics Configuration System** documents PCS, where physicists meet the production system: the tag catalog, dataset composition, composed-name identity, and the browse and compose interface
- **Validation** documents WFMS support for ePIC validation activities
- **Calibration** documents WFMS support for calibration workflows, the multi-step orchestration domain
- **Distributed CI** documents ePIC software validation running on PanDA-accessed distributed resources
- **Distributed Analysis** describes the support for distributed and managed analysis workflows
- **Operations** describes how the WFMS is operated
- **Organization** describes the organization of WFMS efforts and how the work is coordinated
- **Timeline** describes the development and use of the system as Collaboration needs evolve over the coming decade, culminating in physics datataking in the mid-2030s
- **References** collects reference documents, glossary terms, external links, API references, MCP references, and requirements material
- **Diagram Gallery** collects all the system diagrams

Suggested entry points by reader: physicists requesting or consuming production data — Concepts, Production System, Physics Configuration System, and Distributed Analysis; production and validation operators — Operations and the runbook links in References; DAQ and streaming collaborators — Foundations and Streaming Workflows; contributors and facility participants — Architecture, WFMS Platform, and Organization.

This documentation is the system description; per-component design and operational detail lives with the implementations, linked from References. The documentation is maintained in the eic/epic-wfms-docs repository.

[ToC]

2. The System

2.1 Foundations

This section defines the WFMS scope, requirements basis, streaming computing model alignment, computing use cases, Echelon model, and butterfly model. It explains what the system must accomplish and why it is structured this way.

The WFMS foreseen by ePIC for physics operations is not presently realized, nor should it be a decade prior to datataking. ePIC is building towards it, focused on current needs, with an operating streaming workflow testbed evaluating an agentic and PanDA/Rucio based infrastructure for E0-E1-E2 datataking workflows, and an operating production system integrating PanDA, Rucio and AI systems through an agentic infrastructure. The trajectory of these developments is aligned with the end goal, if all goes well. ePIC decision points in the future will determine the trajectory. This document describes the end goal, the present technical implementations, and the planned timeline (at a high level, appropriate to this stage of the project).

2.1.1 WFMS Scope

The ePIC Workflow Management System (WFMS) is the shared workflow, data management, monitoring, automation, and operations layer for ePIC computing, from datataking to global production. It encompasses streaming and production — including workflows that combine aspects of both — in a common WFMS platform for the experiment, spanning post-DAQ E0-E1 streaming processing, global production campaigns, validation workflows, distributed analysis support, calibration, distributed CI, and future datataking operations.

The present system has two active implementation fronts, building from a common platform for a coherent and efficient overall WFMS capability:

- the **streaming workflow testbed**, prototyping E0-to-E1 streaming workflow and dataflow orchestration for the documented ePIC streaming computing model[2]
- **epicprod**, the automated production system. It gathers and integrates production requests from the community, establishes the physics configurations that define request production tasks, and documents and manages present, past and future production campaigns. It configures execution-ready tasks and executes them on a distributed computing infrastructure using PanDA and Rucio, with comprehensive monitoring from drill-down expert diagnostics to user-friendly overviews. Maximal automation and AI integration throughout the system enable high quality production operations with minimal human operations effort.

2.1.2 Requirements

The WFMS is guided by the ePIC distributed workflow management system requirements[1] and the ePIC streaming computing model.[2]

The requirements establish the main design obligations:

- support all major ePIC processing use cases, from streaming post-DAQ processing through production, reprocessing, validation, calibration, and analysis
- operate across heterogeneous distributed resources, including E1 host-lab facilities, E2 facilities around the globe, E3 user environments, and opportunistic or specialized resources
- integrate workflow management with distributed data management, so data availability and data placement can drive processing
- provide web, REST, programmatic, and operational interfaces suitable for users, operators, facilities, and automated services
- maintain provenance, bookkeeping, metadata, monitoring, diagnostics, and logs as integral parts of the system
- support high-quality operations with maximal automation and minimal routine human effort
- allow fast-turnaround development and deployment without losing operational control

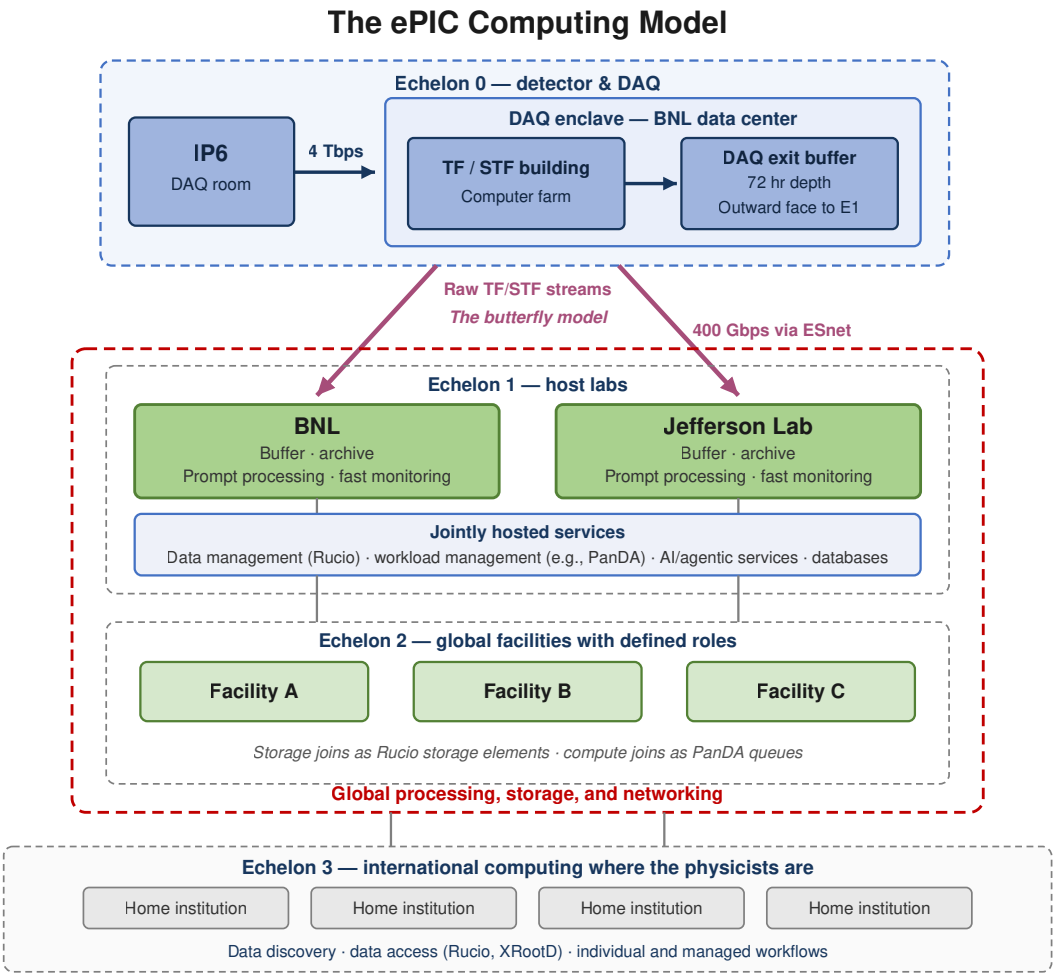
The WFMS computing use cases include stored data streaming and monitoring, alignment and calibration, prompt reconstruction, first-pass reconstruction, reprocessing, simulation production, validation, analysis workflows, and modeling or digital-twin workflows.

[ToC]

The ePIC computing model is organized by Echelons:

- **E0**: detector and DAQ environment at the experiment site and extending to a 'DAQ enclave' in the BNL computing center
- **E1**: the two host-lab computing facilities at BNL and JLab
- **E2**: global ePIC processing and data facilities with well defined responsibilities in ePIC computing
- **E3**: home-institute and user-side computing environments: computing where ePIC physicists are

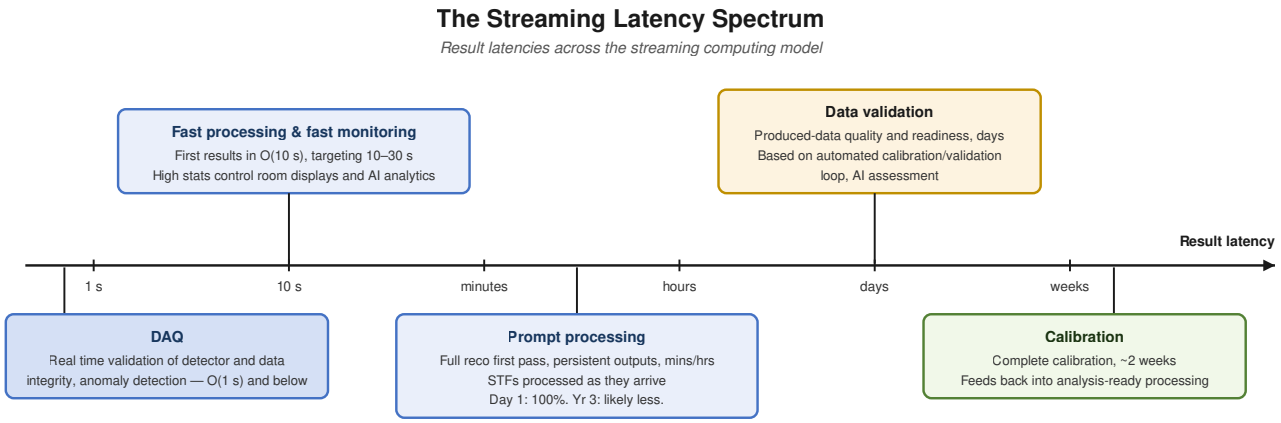
The host-lab symmetry of the 'butterfly model' is central to the E1 design. E0 is necessarily bound to BNL where the EIC is located, but post-DAQ, the butterfly model expresses symmetry in computing capability between the two host labs. Raw data flows from E0 to both E1 sites, establishing geographically separated raw-data copies and preserving flexibility in how ePIC assigns downstream processing roles between BNL and JLab, and other facilities.



2.1.3 Streaming Computing Model Alignment

The ePIC streaming computing model drives the WFMS architecture from the start. ePIC data processing begins with a continuous post-DAQ data stream during datataking, delivering first results from E1 processing in O(10 s) to inform control room operations and AI tools of current detector and machine performance, and delivering more complete and digested prompt processing from bulk E1 (and possibly E2) resources over minutes and hours. The system must be capable of delivering evaluation, validation and calibration results at the latencies ePIC demands, from seconds for detector and data evaluation/validation to ~2 weeks for full calibration cycles. The WFMS therefore has to support data-driven operation, near-real-time monitoring, prompt processing, calibration feedback, and distributed execution as baseline system capabilities.

[ToC]



The streaming workflow testbed exercises this model directly. It prototypes the E0 to E1 interface, time frame based processing, fast monitoring, Rucio data handling, PanDA task execution, a fast agentic message-based infrastructure, monitor-backed deep operational state, and REST and MCP services that inform collaborating software, AIs and people. These are the early implementation of WFMS behavior needed for datataking, to inform ePIC's decision making on streaming WFMS over the next several years.

The epicprod production system exercises the same platform from a different direction. It uses PanDA, monitoring, task cataloging, AI-based assessments and diagnostics, human-in-the-loop control and curation, and data product cataloging to run the simulation/reconstruction campaign operations that ePIC needs today, automated to the greatest extent possible to minimize the demand on the scarce operations effort ePIC has available. The production experience feeds the same WFMS design: operational clarity, automation, provenance, site awareness, and reliable human control.

Further workflow domains are foreseen and are starting to be addressed: distributed CI using PanDA-accessed resources, calibration workflows combining orchestration tools like Snakemake with the ePIC WFMS platform, and analysis workflows that can benefit from the platform, e.g. by using distributed resources and/or centrally managed execution.

The foundation is therefore one platform serving multiple workflow domains. Streaming, production, validation, calibration, distributed CI, and analysis have distinct operational needs, but they should converge on shared services, shared data models where practical, shared monitoring, and shared operational discipline.

---

1. ePIC Distributed Workflow Management System requirements. https://www.overleaf.com/project/67bdf89a3d44a138da503dea ↩
2. The ePIC Streaming Computing Model. https://zenodo.org/records/14675920 ↩↩

[ToC]

2.2 Concepts

The working vocabulary of the WFMS: the data units, configuration entities, identities, and lifecycles used throughout this documentation and in the system's interfaces. Definitions here are the reference; the domain sections describe how the concepts are used.

2.2.1 Streaming Data Units

Time frame (TF) — the atomic unit of ePIC streaming data: a contiguous, self-contained slice of the detector data stream.

Super time frame (STF) — an aggregate of consecutive time frames into a file-sized unit. The STF is the unit of registration, transfer, bookkeeping, and bulk processing: it is what Rucio registers and moves, what run datasets collect, and what prompt processing consumes.

TF slice — the parallel work unit of fast processing: a subsample of an STF comprising contiguous time frames. Slice size is set by the target latency for control room plots and statistics and for AI assessments — a TF count whose reconstruction processing time matches the target, 10-30 seconds in present thinking.

Run — a datataking period. The run lifecycle — run imminent, run start, pause and resume, run end — is broadcast from the DAQ side and drives downstream orchestration. At run start a Rucio run dataset is created; arriving STFs are registered into it.

2.2.2 Production Configuration and Identity

Tag — a named parameter set capturing the configuration of one production stage: physics (p), event generation (e), simulation (s), and reconstruction (r), with physics tags grouped by category. A fifth type, background (k), is an orthogonal overlay — a named background configuration composed into a dataset alongside the stage tags, keeping signal and background configurations independent and avoiding tag growth across signal-background combinations. A tag is `draft` (editable) or `locked` (immutable, a one-way transition), so a configuration used in production never changes meaning.

Dataset — the concrete production unit: one sample, produced by a single task. Its identity composes the classification tags with any sample-variant discriminator into the composed name. A sample normally registers as a single Rucio dataset, with planned provision for multiple datasets mapping to one task: Rucio limits a dataset to 100k files, and production tasks have exceeded that, so a large sample subdivides into Rucio block datasets (`.bN`) under one logical identity.

Sample variant — a named discriminator for samples that share a physics configuration but are produced as separate datasets, such as the angular ranges of a single-particle sample. A variant is a name plus the submission parameters that produce that sample; each variant materializes as its own dataset and task, distinguished only by the variant segment of the name.

Production config — a reusable template of execution-side settings: software stack, resource targets, splitting, site and data-handling parameters. Production configs are deliberately mutable working templates; the PanDA task and job records are the immutable record of what actually ran.

Composed name — the identity that runs through the whole system:

```
{scope}.{det_version}.{det_config}.{p_tag}.{e_tag}.{s_tag}.{r_tag}[.{k_tag}][.{sample_name}]
```

The optional segments are the background overlay tag (`k`), present when the dataset composes a background configuration, and the sample-variant name. A whole-task rerun appends a further optional segment, `.tryN` , to the name itself — the PanDA task name and the output dataset name — which is what keeps repeated submissions clash-free; the catalog resolves any `.tryN` name back to the same campaign task (see logical and physical names below).

The composed name states the physics configuration and carries the identity in a compact form usable in entity naming: it serves as the task's identity in catalog pages, links, and the API, and names the PanDA task, the produced Rucio datasets, and the files within them. The version segment is the detector version — it describes the conditions of the produced data; campaign membership is bookkeeping and does not rename the identity.

Logical and physical names — the composed name is the stable logical identity. Physical names append reserved suffixes when uniqueness requires them: `.tryN` when an entire task is deliberately rerun, and `.bN` for Rucio block subdivision of very large datasets. Parsing strips the terminal suffixes, so every physical name resolves back to its logical identity.

[ToC]

2.2.3 Requests, Campaign Tasks, and Campaigns

Production request — the structured record of a community request, typically submitted by physics working groups and detector subsystem collaborations. A request is deliberately more abstract than an output — it states the physics, beam energies, and kinematic range — and one request commonly leads to several produced datasets. Requests specify the requestor, so task status and completion notifications can reach them. Provenance is by reference: a task resolves through its request to the originating submission.

Campaign task — the submit-ready binding of tags, dataset, and production config; the unit the catalog stages, submits, and accounts for. Its lifecycle:

```
draft → ready → submitted → completed | partial (recoverable) | failed
```

`draft` covers every incomplete state; a task moves to `ready` under readiness checks and operator confirmation, `submitted` on submission, and PanDA drives the outcome: completed, partial — recoverable by completing the tail — or failed.

PanDA task association — the record of one physical PanDA submission of a campaign task, preserving the complete submission history on the task record. Retry is native to PanDA: failed or missing jobs are retried within the existing PanDA task, under the same name, and the efficient course is always to complete the tail of the existing task. Only the special case of deliberately rerunning an entire task — essentially cloning it — creates a new submission under a `.tryN` name, a case that serves mainly system testing on small test tasks.

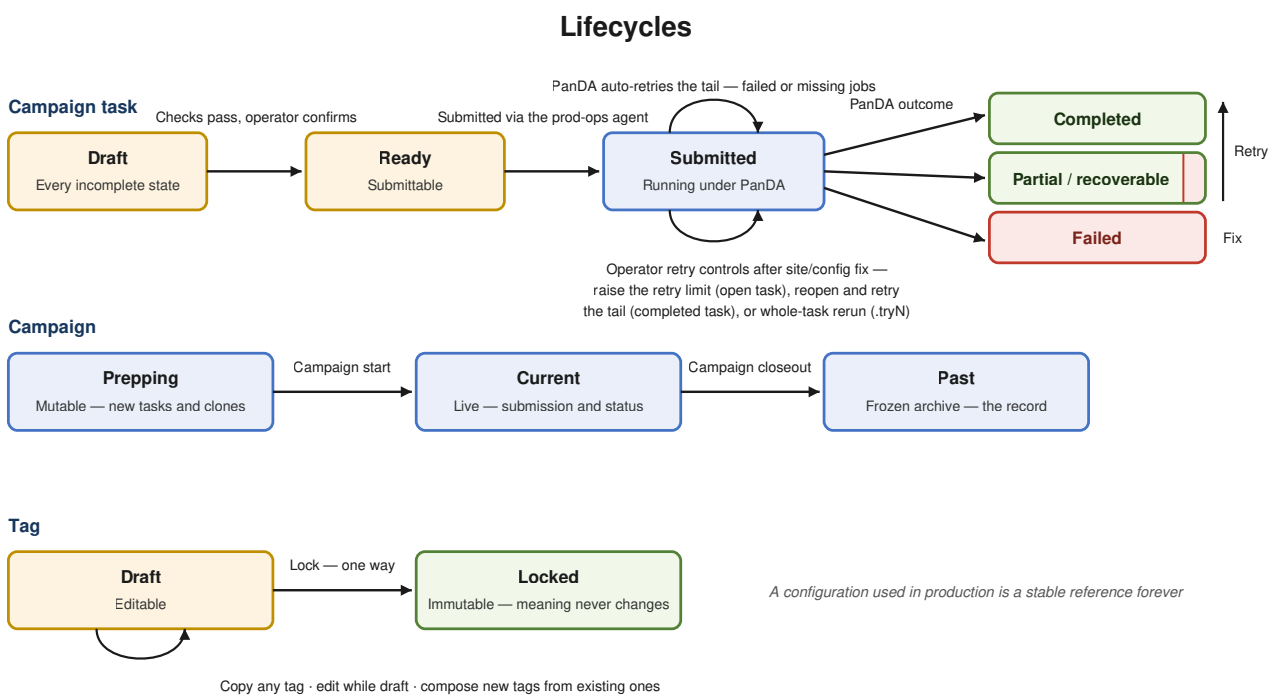
Campaign — a time-ordered grouping of production tasks, named by year and month (25.10, 26.06). Campaigns stage the catalog:

```
prepping (mutable) → current (live) → past (frozen archive)
```

Prepping campaigns accept new tasks and clones; the current campaign is where submission and live status happen; past campaigns preserve all task parameters as the production record.

Campaign narrative — a human-authored account of a campaign's goals, priorities, and evolution. It is shared context for operators and LLMs: daily reports and assessments reason against it.

The lifecycles above in one view:



[ToC]

2.2.4 Execution

PanDA task — the executable unit in PanDA, identified by its JEDI task ID; PanDA breaks it into **jobs** brokered to **queues** at computing sites, executed by workers that Harvester provisions. The PanDA vocabulary is documented in full in the PanDA basic concepts.

2.2.5 Platform Operational Entities

Agent — a message-driven service process built on the shared agent base, registered and heartbeating in the monitor. Agents implement testbed workflow roles and the credentialed production operations executor.

Namespace — the isolation boundary that lets concurrent users and services run workflows side by side; agents filter messages by namespace.

Workflow execution — one tracked instance of a running workflow, recording its runs, files, messages, and state in the monitor database.

[ToC]

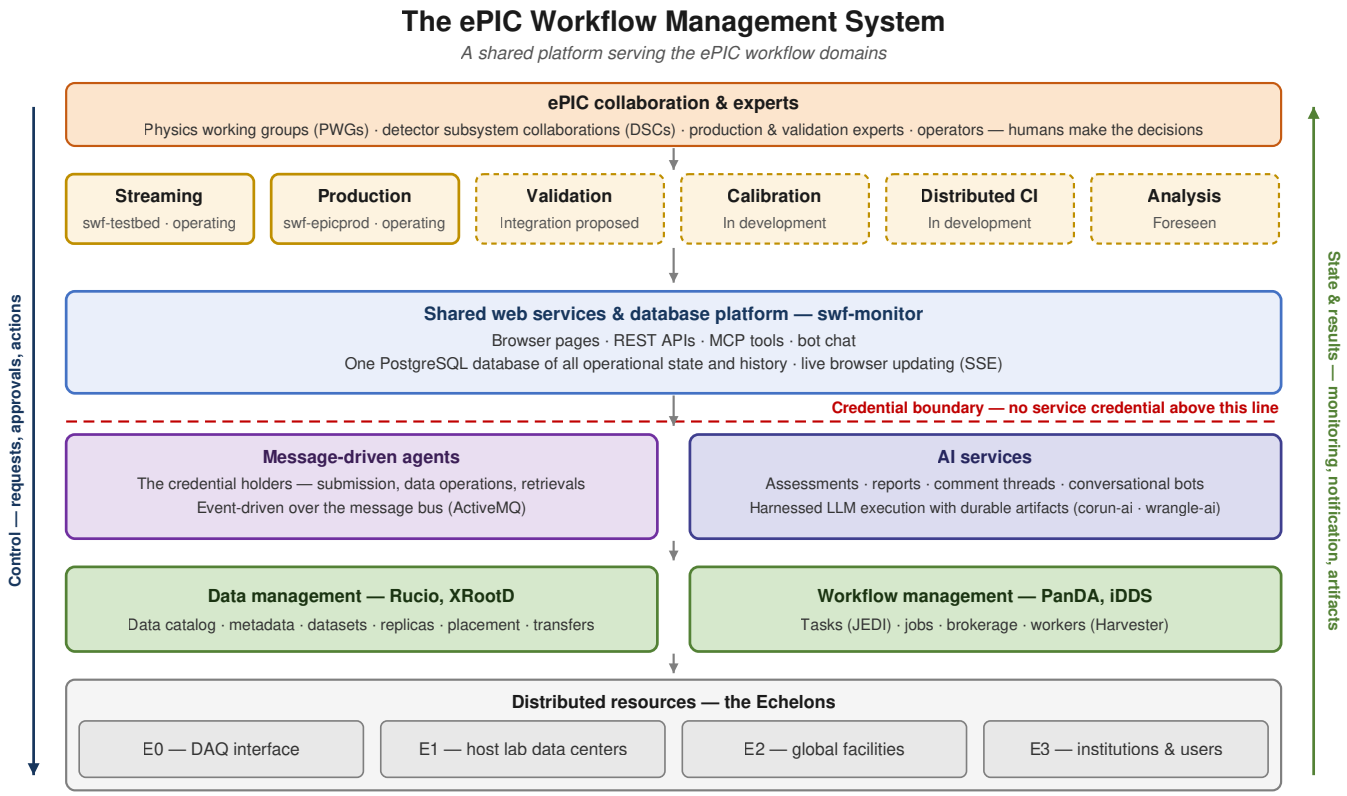
2.3 Architecture

This section describes the overall WFMS design: common system infrastructure, data/control flow, human-in-the-loop automation, and AI integration. It is the conceptual map of the system, sitting between the foundational requirements and the specific implementation details.

2.3.1 System Overview

The ePIC WFMS architecture is organized around a shared platform serving the multiple workflow domains described in the previous section. The domains have varying operational requirements, but the architecture converges on common services where practical, from workflow orchestration and data management through monitoring, human controls, and AI assistance.

The platform/domain separation is carried through to the code organization: platform code lives in the common library and the common web and database service, and each workflow domain is a peer application with its own repository — the testbed in `swf-testbed`, production in `swf-epicprod` — consuming the platform through its service interfaces. Repository organization is described in Platform.



The architecture is agent and service oriented, and prioritizes full access to system information at all detail levels by operators, users and AI agents. Information management begins with a comprehensive, centralized back end database capturing all system information and state. Persistent databases and document/artifact stores operating on a common database service hold operational state and history, metadata, curated knowledge, assessments, comments, and reports.

Browser pages present operational views and controls, from high level summaries to deep drill-down with full presentation of filterable metadata. Browser interfaces always capture all state in the URL, for reproducibility, bookmarking and transparency.

REST APIs support system workflow interrogation and control, service-to-service interaction, and automation. MCP tools expose structured system context to AI clients and bots, and, selectively and under controls, active functionality that AIs can exercise safely and reliably.

[ToC] Agent services drawing on MCP and REST perform credentialed or long-running actions away from the web tier. They can be activated in several ways: regular cron-like invocation, integration as discrete steps in a workflow, human triggered by natural language interaction with a bot or LLM, or human triggered via web interface. Agents operate using no-latency messaging to/from an asynchronous worker queue, accommodating the macroscopic times (seconds to minutes) that LLM and distributed service operations take to complete.

2.3.2 Data and Control Flow

WFMS data flow begins with the data products and metadata produced by detector, simulation, reconstruction, validation, and analysis workflows. In streaming workflows, the flow starts at the E0-E1 interface, in data terms at the DAQ exit buffer, with time frame and super time frame data driving registration, movement, monitoring, and prompt processing.

In production workflows, the flow begins with community production requests leading (with human intervention) to corresponding physics configurations, which are transformed into executable production tasks that produce the desired data products.

Control flows begin with humans or scheduled automated operations. System operators, production experts, validation experts, and collaboration users interact through web interfaces, bots, REST clients, and analysis interfaces. These points of interaction, at exposed surfaces such as web browsers and servers, do not directly hold credentials to operate system services. Credentialed operations are routed through designated agents such as a production operations agent for interacting with distributed computing services, and LLM harness agents for engaging AI assistance. This keeps privileged actions secure, auditable, and bounded to only the appropriate exposure surface.

System state flows back through monitoring and notification channels. Services, agents, databases, logs, and curated artifacts provide the state used by dashboards, diagnostics, AI assessments, campaign reports, and user-facing overviews. Browser notification uses server-sent events where asynchronous work needs to report completion to an open page.

2.3.3 Human-in-the-Loop Automation

The architecture is based on maximal automation with explicit human control. AI-equipped automation should reduce routine operations effort, expose system state, identify failures, present inferences and interpretations, and carry out approved bounded actions. It does not remove the need for production experts, validation experts, and detector or physics domain experts to set priorities, curate knowledge, approve operational changes, and interpret results. Agentic tools must incorporate appropriate harnesses to assert controls and enforce predictable, responsible operation.

Human-in-the-loop operation appears at several levels:

- production requests and campaign scope are defined by people and structured by the system
- physics configuration systems encode production configurations in a form suitable for task generation and review
- production experts control task submission, retry, recovery, and campaign steering
- validation experts evaluate produced data and feed readiness decisions back into production
- AI assessments and reports support review, but do not replace operational responsibility; they inform actions leaving humans to drive them
- curated campaign narratives and comments provide context for later assessment and decision making

The architecture favors visible state, explicit control points, auditable actions, and fast feedback loops.

2.3.4 AI Integration

AI services consume structured WFMS context and produce assessments, comments, reports, summaries, and recommendations. Operational authority remains with the relevant human and programmatic/deterministic service components. If and when confidence in granting AI actionable functionality is established, the same programmatic action interfaces that humans operate can be made available to AIs, e.g. through appropriate MCP tools.

The same architectural pattern applies beyond production. Streaming workflows, validation, calibration, distributed CI, and analysis can all expose structured state through APIs and MCP tools. AI systems can then reason over current and historical state, curated campaign knowledge, produced artifacts, validation outcomes, and operational logs. The useful role of AI is to improve assessment, diagnosis, summarization, and ideation while keeping the WFMS state and action paths explicit.

The architectural rule is that AI outputs become artifacts in the system. They should carry provenance, be attached to the relevant production or workflow object, be open to human (and AI) comment where appropriate, and be available as context for later reports or assessments. AI is very new, any application of it is an R&D exercise, and use of it must accordingly accommodate correcting, qualifying or ignoring its products as appropriate, and trying and comparing different models and approaches.

[ToC]

2.4 WFMS Platform

This section covers the specific technical platform used to build the ePIC WFMS across streaming, production, validation, distributed analysis, calibration, distributed CI, and related workflow domains. It marks the passage from technology-agnostic architecture to implementation specifics. It documents the concrete systems, services, libraries, interfaces, and conventions used to realize the architecture described in the previous section.

The platform is developing through two active implementation fronts: the streaming workflow testbed and the epicprod production system. These fronts have different near-term purposes, but they are deliberately based on the same foundation: PanDA and related workflow services, Rucio and XRootD data handling, message-driven agents, a shared monitor/database service, REST and MCP interfaces, browser-based operational pages, AI bot interfaces, and harnessed LLM AI services.

2.4.1 Production and Testbed Convergence

The two implementation fronts exercise complementary parts of the platform. Production carries the immediate operational needs: campaign definition, task creation, PanDA execution, data product accounting, monitoring, human controls, and operations automation. The testbed exercises the datataking side: E0-E1-E2 streaming processing, message-driven agents, fast processing, Rucio data handling, and prompt monitoring.

Several production components serve the testbed as well. The Physics Configuration System (PCS) provides a dynamic, editable, composable catalog for task creation. Rucio-based input ingest and PanDA submission are production needs today and testbed needs for composed streaming-processing tasks. The task catalog's browsing, state, controls, and copy/edit capabilities can extend to testbed namespaces, data challenges, and future streaming exercises.

The ePIC-specific monitor originated in the testbed and now supports both fronts. Bot interfaces, alarms, operations agents, MCP tools, and LLM-backed assessment services are likewise shared, with authorization and action policies set per domain.

2.4.2 Repository Organization

The `swf-*` repositories implement the current platform. `swf-testbed` is the umbrella repository for testbed configuration, orchestration, and documentation. `swf-monitor` is the central web and database application — the platform's common monitor, web, API, and database services. `swf-common-lib` holds shared agent, messaging, logging, and Rucio helper code; code used by more than one component belongs there. `swf-epicprod` (created July 2026) is the production domain's repository, a peer application of `swf-testbed`: production-specific applications and documentation consolidate there, shipped as Django applications installed into the swf-monitor runtime. The epicprod implementation built to date resides in `swf-monitor` and migrates component by component; the swf-epicprod architecture map (`docs/ARCHITECTURE_MAP.md`) records each component's home, destination, and consumption interface. Agent repositories (`swf-daqsim-agent`, `swf-data-agent`, `swf-processing-agent`, `swf-fastmon-agent`) implement specific testbed roles. The fast-processing worker layer is implemented in `swf-transform` (the worker-node payload transformation, processing slice messages through the reconstruction payload) and `swf-panda-workers` (worker lifecycle coordination with iDDS and PanDA).

The three core repositories (`swf-testbed`, `swf-monitor`, `swf-common-lib`) advance together on coordinated branches; `swf-epicprod` is maintained on its main branch outside the coordinated set.

2.4.3 Web and Database Stack

swf-monitor

`swf-monitor` serves both the testbed and epicprod; the historical name now covers much more than monitoring. It is a Django application backed by PostgreSQL, providing the platform's browser pages, REST APIs, and MCP tools and the database-backed state beneath them: message history, agent state, PanDA monitoring views, PCS, task catalogs, and LLM assessment integration. With the production domain consolidating in `swf-epicprod` — whose applications run installed within the monitor's envelope — the monitor's own scope returns toward its platform role: the common monitor, web, and database service.

Database-Backed Operational State

[ToC]

The monitor database stores all operational state that must be queried, filtered, linked, audited, or rendered, holding current state and its history. Testbed records include runs, agents, messages, files, workflow executions, and fast monitoring state. Production records include requests, PCS entities, campaign tasks, PanDA associations, operator state, cached system status, LLM artifact pointers, and production metadata.

Django models and migrations define the schema. JSON fields hold extensible metadata; structured fields are used where filtering, linking, constraints, or operator workflows require them. The pervasive use of JSON fields makes the system flexible and adaptable to include new knowledge, state and functionality.

Browser Pages

Pages are server-rendered Django templates with targeted JavaScript for filtering, selection, asynchronous actions, and notification. Page state that affects interpretation (selected task, active tab, filters, sorting) is carried in the URL, making views bookmarkable and stable as references.

Pages that have buttons triggering long-running or credentialed work enqueue an agent request and return promptly, not waiting for completion; completion arrives through database state and immediate browser notification with an auto-updated page.

Server-Sent Events

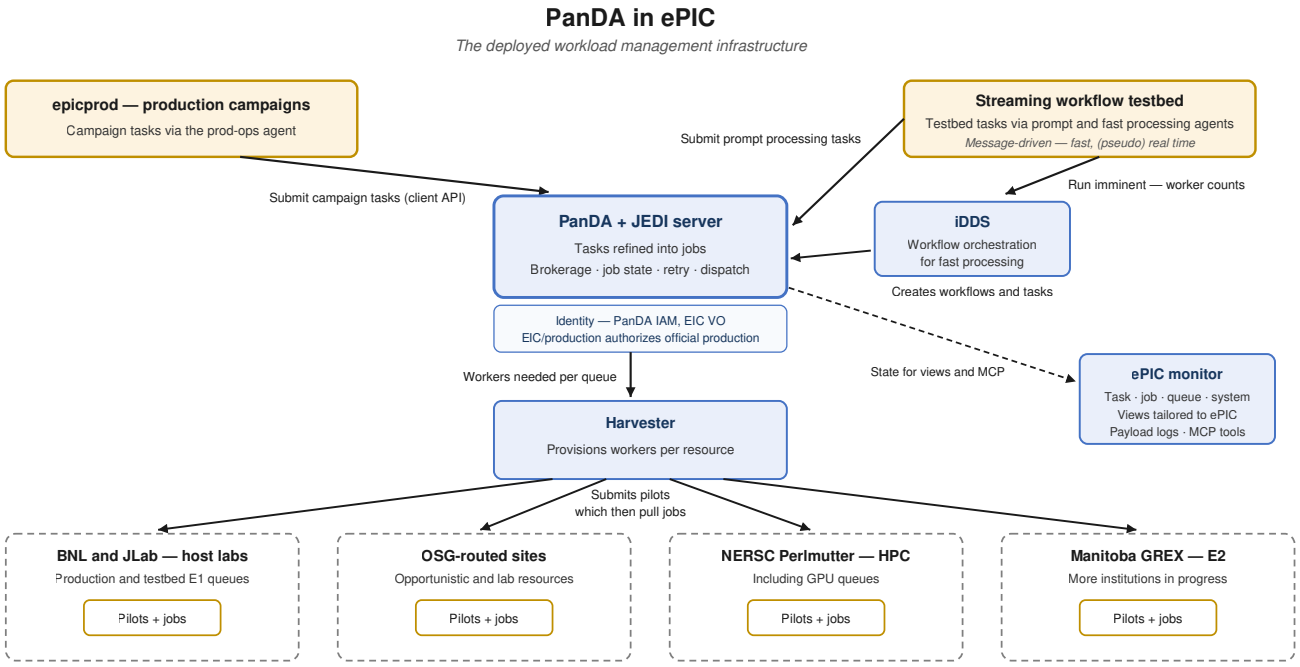
Server-Sent Events (SSE) deliver asynchronous completion notices to the browser. swf-monitor consumes message-bus events, persists relevant messages, and relays selected events to browsers through `/api/messages/stream/`. Operations-agent completions and corun-ai callbacks reach production pages the same way. SSE carries only the notification; on receipt, pages refresh their state from the database or REST.

2.4.4 Workflow and Workload Management

PanDA

PanDA[1] is the distributed workload management system for both fronts: it turns task specifications into jobs, brokers work to sites, and records task and job state. epicprod uses it for campaign tasks; the testbed uses it to execute prompt and streaming workflows.

swf-monitor presents PanDA task and job state through task, job, queue, and system views tailored to ePIC, supplanting the default ATLAS PanDA monitor with production task links, campaign context, payload-log access, Rucio output views, and LLM assessments.



JEDI

JEDI is the PanDA task definition and execution layer, used for direct production task submission from PCS. PCS builds the task specification, maps its fields into a JEDI `taskParamMap`, and queues the credentialed submission through the production operations agent. The live epicprod path uses the PanDA client API for EVGEN production tasks and records each physical submission attempt as a PanDA task association.

[ToC]

The integration separates logical campaign task identity from physical PanDA/Rucio attempt names. The logical identity remains stable in PCS; physical submissions may append `.tryN` so that reruns have unique PanDA task names and output namespaces, and/or `.bN` to organize blocks of outputs within a task when very large tasks would exceed Rucio's maximum dataset size (100k files).

iDDS

iDDS[3], part of the PanDA ecosystem, is PanDA's workflow management component, orchestrating multi-stage and data-driven workflows in which data availability, transformation, and workflow state must be coordinated beyond single task submission. It is capable of large and complex workflows, able to handle future streaming and production workflow expansion. PanDA also has an internal workflow capability under development, for workflows integral to PanDA's management of an overall task; PanDA has full and direct awareness and control of a multi-stage task, rather than delegating to iDDS. As this capability matures it will be reviewed for possible application in ePIC WFMS. The complex workflows of ePIC fast streaming processing will always be an iDDS domain.

Workflow Descriptions

The testbed uses a layered workflow model: TOML for human-editable configuration and parameters, Python/SimPy for execution and simulation patterns that need rapid iteration, and Snakemake as the layer where dependency management is central, as in calibration and other multi-step orchestration. Snakemake integration for real ePIC calibration and CI workflows is well advanced: a Snakemake executor plugin for PanDA is published, ePIC benchmark workflows have run in CI through an HTCondor executor, and the first eicweb CI jobs have submitted work to BNL through PanDA.

2.4.5 Data Management

Rucio

Rucio[2] is the distributed data management system of ePIC and of the platform. In the testbed it manages run datasets, super time frame (STF) file registration, subscriptions, transfer rules and execution, and Rucio Storage Element (RSE) state. In production it records and exposes production data products (EVGEN inputs, RECO outputs, and log datasets). Rucio DIDs (dataset identifiers), scopes, RSEs, rules, replicas, and metadata are operational objects, presented beside workflow state in monitor pages and task catalogs.

XRootD and FTS

XRootD is the file access and transfer protocol for the testbed and production paths. Rucio drives data movement through FTS (the File Transfer Service); agents also use XRootD directly for direct read operations, including over distributed sites.

Production Science-Data Constraint

The BNL PanDA server used for ePIC production is configured with an associated Rucio instance, BNL Rucio, where PanDA records logs. ePIC production science data is held in JLab Rucio. Production payloads therefore handle science data movement and registration directly against JLab Rucio on both the input and output sides, with PanDA not directly involved in science data resolution, transfer, and registration for production operations. Rucio operations may be consolidated on a single PanDA-coupled instance in the future. Payload-side data handling uses standard PanDA/Rucio copytools for worker node data movement. The monitor includes built-in access and presentation of Rucio-based science and log data.

Data Product Cataloging

Data products are cataloged and curated as the primary product of the production process. Production campaign tasks link to expected and observed outputs: Rucio DIDs, output status, logs, and PanDA task associations. The testbed records run, STF, time frame (TF), and workflow-stage metadata. Workflow state and data-product state are inspectable together.

2.4.6 Distributed Resources Integration

The platform expresses the Echelon model defined in Foundations through workflow processing destinations, a prompt processing 'decision box' determining E1 processing, data locations, site state, queue state, and monitoring views. The same abstractions serve present testbed emulation and future distributed facility scale operation.

[ToC]

PanDA represents compute resources as sites and queues; Harvester and the pilot infrastructure connect PanDA tasks to the site execution layer, typically a batch system. swf-monitor presents task, job, queue, site, resource usage, and harvester-worker state so that failures can be diagnosed at the level of the campaign task, PanDA task, job, queue, or site.

The testbed emulates elements of the E0-E1-E2 workflow with local services and configured RSEs. DAQ buffer and E1 storage roles are represented by distinct RSEs and XRootD paths even when implemented on the same host during development, so workflow and dataflow logic can be exercised before the physical facilities and final data paths exist.

2.4.7 Metadata and APIs

REST APIs are the programmatic interface for browser pages, scripts, agents, service-to-service calls, and automation; command-line clients and operational scripts work through REST endpoints rather than Django internals or direct database queries. Endpoints return structured errors, preserve stable identifiers, and represent the same state shown in the browser. Write actions that must work through the remote proxy use a proxy-safe shape: JSON response, no session or CSRF dependence the proxy cannot carry, and no redirect as the action result.

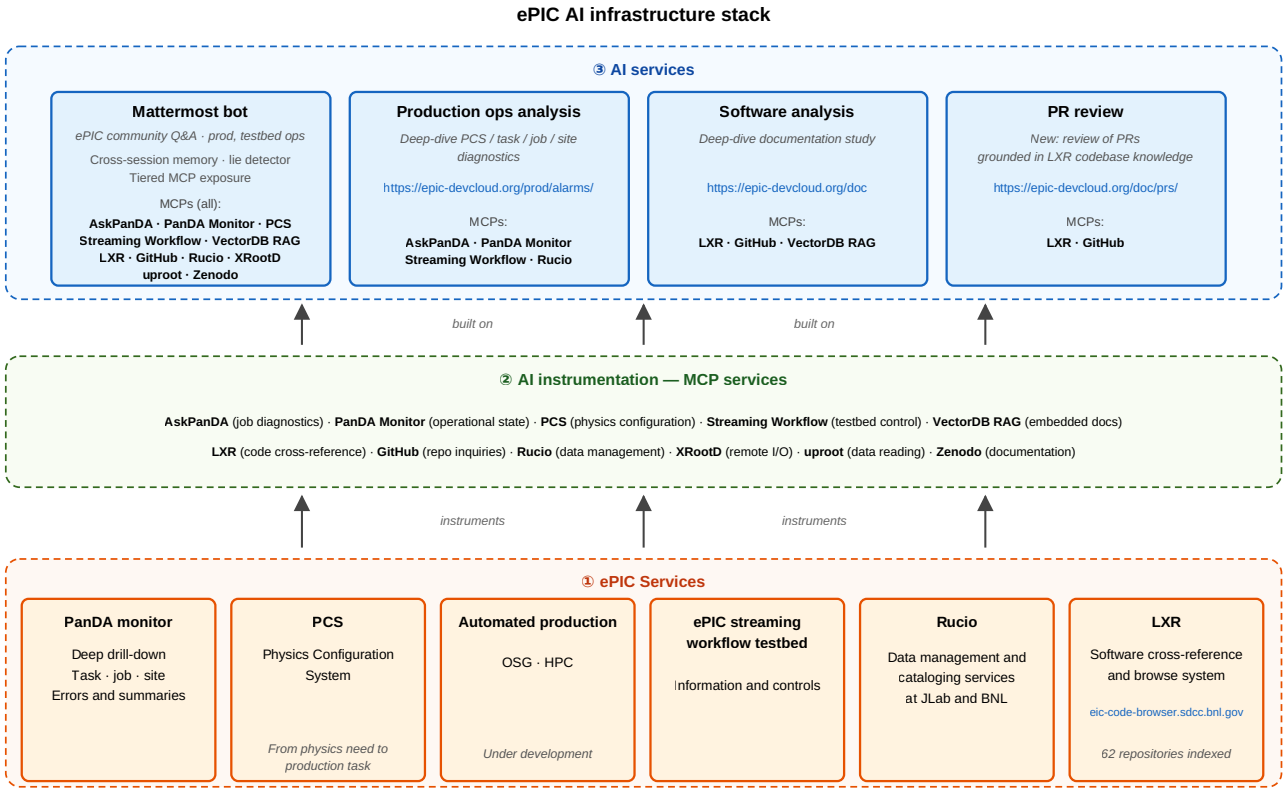
MCP (Model Context Protocol) tools are the corresponding structured interface for LLM clients, exposing testbed state, production state, PanDA monitoring, PCS entities, LLM artifacts, and selected bounded actions. swf-monitor serves MCP from a FastMCP ASGI worker separate from the Django WSGI site. Tools are designed as data access and bounded-action primitives: list tools support filtering, pagination, and stable identifiers, and action tools route through the same service layer and agent execution paths as browser and REST actions. Tool docstrings are operational metadata: they are the primary text an LLM reads when deciding what to call.

Both interfaces rest on stable identifiers: campaign task composed names, JEDI task IDs, PanDA job IDs, Rucio DIDs, RSE names, agent names, workflow execution IDs, section slugs, and document artifact group IDs. JSON metadata identifies the source system, artifact type, subject reference, producing user or service, and, for LLM-generated artifacts, model and prompt provenance.

AI Tool Coverage

The MCP instrumentation has grown out of the WFMS work into broad coverage of the systems ePIC AI clients need: PanDA job diagnostics and operational state, PCS physics configuration, testbed control, read-only database inquiry, Rucio data management, XRootD remote I/O and uproot data reading, code cross-reference over the ePIC software repositories (LXR), GitHub repository inquiries, embedded-documentation retrieval, and Zenodo documents. AI services are built on this instrumentation: the Mattermost bot serving collaboration-wide questions on production and testbed operations, production operations analysis with drill-down diagnostics, software documentation analysis, and pull-request review grounded in codebase knowledge. The stack — ePIC services, MCP instrumentation, AI services — is diagrammed below.

[ToC]



2.4.8 Agents and Services

Agent Infrastructure

`BaseAgent` in `swf-common-lib` is the shared agent base, providing STOMP/ActiveMQ integration, monitor registration, heartbeats, namespace filtering, message dispatch, REST logging, and background execution. Agents built on it are visible in the monitor and follow common message and status conventions. `BaseAgent.run_in_background()` gives handlers a bounded worker pool for calls into subprocesses, REST services, Rucio, and XRootD, keeping the single receiver thread responsive to liveness and control messages while slow work completes.

A precept throughout the system is no silent failures: an error always produces a visible consequence and informative logging for diagnostics. BaseAgent carries this into the agent layer with REST logging into the monitor, heartbeat and status reporting, and message-level error handling; across the platform, handlers, parsers, and service calls report errors into logs, user-visible messages, or operational summaries.

Production actions are recorded as a structured action stream within the same logging system: every submission, task operation, sweep, import, and configuration change logs the action, its subject, the outcome, and the measured duration. Each event declares its verbosity class in code, and a runtime policy — held with the rest of the operator-set configuration in a database-resident system configuration document, editable in the browser — selects what surfaces on live channels, beginning with a live view of the log pages. The stream is the data source for alarms, digests, and the LLM reporting and assessment services, which reason over it to answer what happened (action stream design).

ActiveMQ Artemis is the message broker. Topics carry broadcast events; queues deliver anycast work. Testbed workflow messages use broadcast topics for events such as run state and STF availability; production operations use an anycast control queue for single-consumer credentialed work. Destination names carry the explicit `/topic/` or `/queue/` prefix, and the choice between topic and queue follows the delivery semantics. Durable subscriptions are used sparingly because they create broker-side state and can accumulate messages.

Workflow and Operations Agents

The testbed agent set models the streaming workflow. `swf-daqsim-agent` simulates DAQ state and STF generation. `swf-data-agent` receives DAQ messages, creates run datasets, manages Rucio STF handling, and notifies processing and fast monitoring. `swf-processing-agent` submits and manages PanDA processing tasks. `swf-fastmon-agent` consumes STF availability, samples TF-level information, records metadata through REST, and publishes notification events for real-time monitoring.

[ToC]

`epicprod_ops_agent` is the always-on credentialed executor for production. The web tier, REST endpoints, MCP tools, bots, and scheduled jobs request work; the agent performs the privileged actions, currently including PanDA submission, payload-log retrieval, Rucio snapshot updates, and PanDA task operations. It follows a handler-plus-doer pattern: the handler validates and queues work, and a doer script performs the credentialed operation as a subprocess. Capabilities remain reusable from cron and operator scripts, and privileged service logic stays out of the browser request path.

Bot Interfaces

The bot interface is a natural-language chat layer over MCP tools and selected REST-backed operations. Bots run as persistent services with conversational access to the same structured state the browser and REST interfaces expose: testbed status, PanDA tasks and jobs, queues, PCS entities, and LLM artifacts, so operational questions get answers grounded in live system state. A demo video (June 2026) shows the production bot at work.

Bot authority is a per-domain policy: the production bot informs, assesses, and answers, while workflow actions remain with operators and the operations agent; a testbed bot or testbed mode can be more active where that suits workflow experimentation. As confidence in harnessed AI operation grows, bots are a natural interface for exercising the same bounded action paths that humans operate, following the AI integration approach set out in Architecture.

Alarms

The alarm system concentrates operator attention where it is needed. Alarms are derived from monitored state, attached to the relevant object or service, and visible in both detail and summary views. Alarm conditions are operator-defined and held in an alarm registry, evaluated on a frequent schedule against monitor state, with triggered alarms notifying by email as well as appearing in the views. It is the first alarm system fielded in the PanDA ecosystem.

Alarms serve the minimal-effort operations goal: automation handles routine conditions, and the alarm stream defines what needs human or AI attention. The capability applies to production operations now and extends to testbed and eventual E0-E1 operations, where streaming latencies demand automated detection and notification.

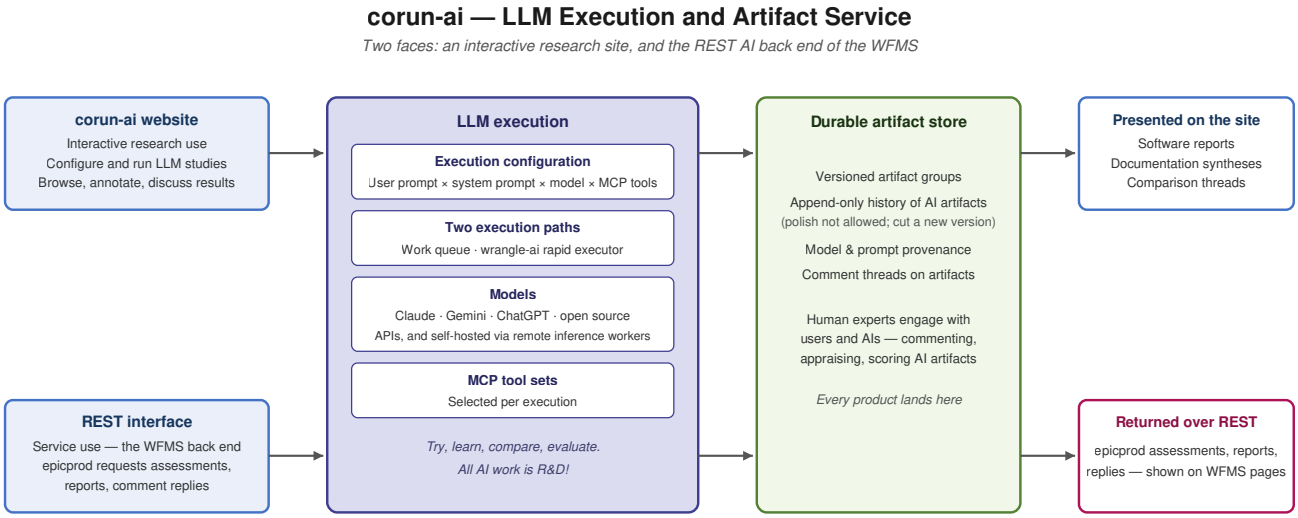
corun-ai and wrangle-ai

corun-ai is the LLM execution and durable artifact service of the platform, central to its AI integration: much of the AI functionality of the WFMS will be delivered through it. It runs LLM work under its own credentials and configuration (models, prompts, execution environment) and holds the results as durable document artifacts carrying model and prompt provenance.

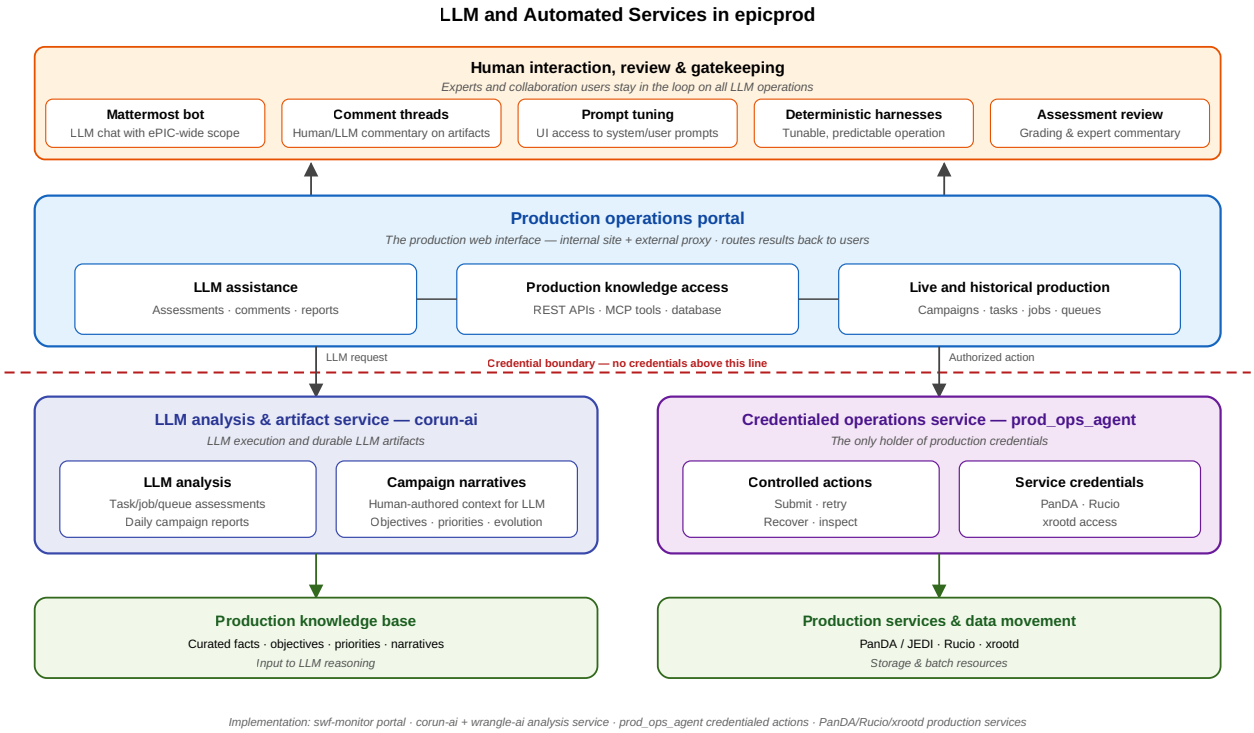
For epicprod, swf-monitor supplies production context and renders the production-facing pages while corun-ai stores and serves the LLM artifacts: assessments of tasks, jobs, queues, and campaigns; LLM replies in comment discussions; human-authored campaign narratives that give LLMs context for reasoning; and daily campaign reports analyzing status and progress. Its current applications are codoc-ai, producing on-request analyses of production, site performance, and failure modes, and argus-ai, the validation assessment application described in Validation. swf-monitor records pointers to these artifacts rather than copying generated content into production records. This implements the architectural rule that AI outputs become artifacts in the system: attached to the relevant object, open to comment, and available as context for later assessments and reports. The role grows with the system; assessment coverage across workflow domains, deeper diagnostics, campaign reporting, and testbed analytics on streaming workflow behavior all build on the same service.

wrangle-ai is the rapid asynchronous executor for bounded LLM operations such as comment replies and assessment probes. For browser-triggered LLM operations, swf-monitor calls corun-ai over REST, corun-ai creates and executes a work item, and completion returns to swf-monitor through a callback converted into an SSE browser notification: LLM results reach the requesting page the moment they complete.

[ToC]



The LLM and automated services of epicprod in one view — human-gated assistance and credentialed automation:



2.4.9 Authentication

Browser access is authenticated according to deployment surface. The internal `pandaserver02` site uses BNL-facing authentication; the external proxy at `epic-devcloud.org/prod` provides collaboration access through swf-remote and the production URL prefix. Browser authorization distinguishes read-only access, production request actions, operator actions, and privileged service execution.

Service credentials are held by the service that needs them. PanDA OIDC tokens, Rucio x509 proxies, XRootD access credentials, and LLM service tokens live in the environment of the agent or service that uses them. Privileged production actions route through `epicprod_ops_agent` : the web tier reads database state and world-readable cache artifacts and enqueues work, while PanDA, Rucio, and XRootD operations with production credentials run only in the agent. MCP observes the same boundary as an API surface whose action tools enqueue work for the agent.

[ToC]

The external proxy is part of the production surface and an implementation constraint. URLs intended for external collaborators work under `/prod/`; write actions that pass through the proxy use the proxy-safe REST patterns above; and pages make remote limitations visible where a control is available only on `pandaserver02`.

BNL internal services use a private certificate chain, so platform scripts and agents use a combined trust bundle that verifies both BNL services and public HTTPS endpoints. Certificate and proxy configuration belongs in deployment configuration and agent environments.

---

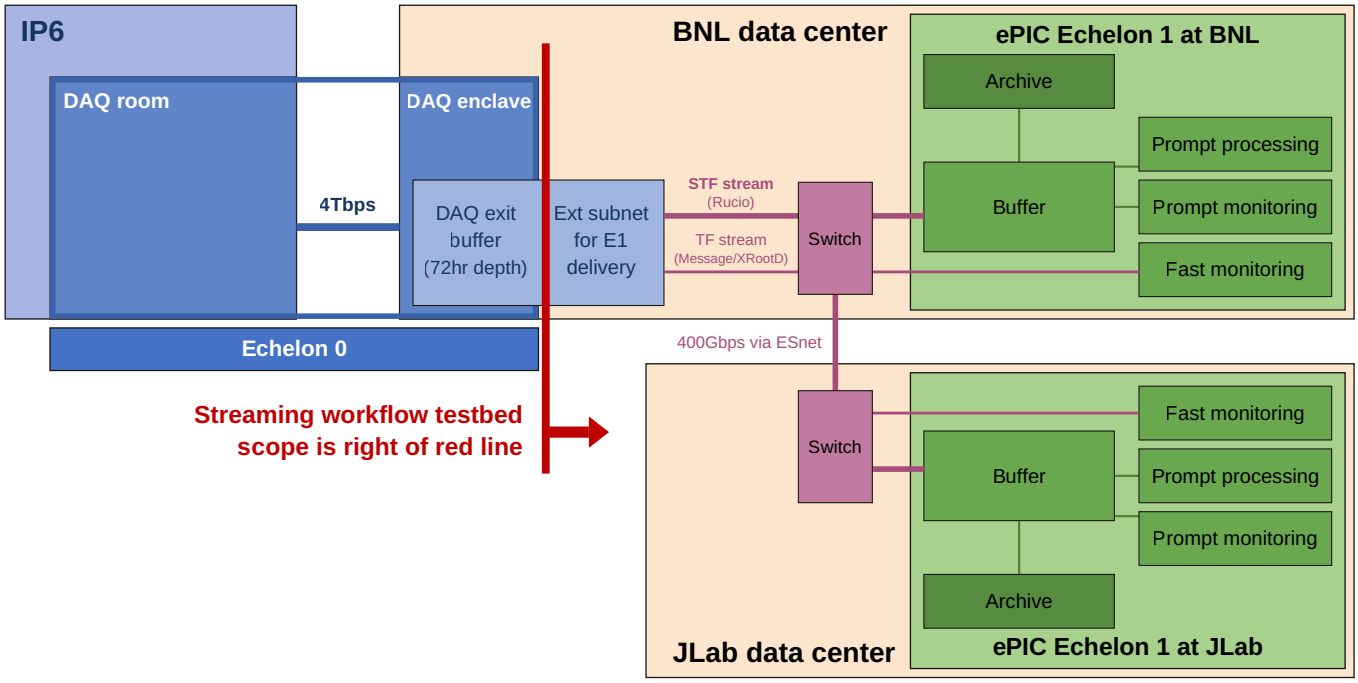
1. PanDA: Production and Distributed Analysis system. Documentation: https://panda-wms.readthedocs.io/ · paper: https://link.springer.com/article/10.1007/s41781-024-00114-3 ↩
2. Rucio: A Distributed Data Management System. https://link.springer.com/article/10.1007/s41781-019-0026-3 ↩
3. iDDS: intelligent Data Delivery Service. https://link.springer.com/article/10.1140/epjc/s10052-025-15275-7 ↩

[ToC]

3. Workflow Domains

3.1 Streaming Workflows

This section documents the post-DAQ workflows of datataking: the E0-E1 dataflow, time frame and super time frame processing, fast processing for low-latency control room and AI analytics, prompt processing of STFs, streaming reconstruction integration, fast monitoring, E2 participation in streaming workflows, and the current realization of these workflows in the streaming workflow testbed.



3.1.1 E0-E1 Interface — Controls and Dataflows

The E0-E1 interface is where WFMS responsibility begins. The DAQ system is one system spanning two facilities: the DAQ room at IP6 and the DAQ enclave in the BNL data center, connected at 4 Tbps. Super time frame (STF) files are built in the DAQ enclave and land in the DAQ exit buffer, sized for about 72 hours of datataking. The buffer extends beyond the enclave onto an external subnet for E1 delivery; this outward face is the piece of Echelon 0 that the post-DAQ world sees, and the WFMS scope runs from it rightward through E1 processing.[1]

Two dataflows leave the exit buffer. The STF stream is the complete raw data: STF files are registered in Rucio at the exit buffer and delivered to the E1 buffers at both BNL and, over ESnet at 400 Gbps, JLab, establishing the two geographically separated raw-data copies of the butterfly model. The E1 buffers serve the full-sample consumers: archiving, prompt processing, and prompt monitoring. The TF stream is a fast subsample delivered at finer granularity, with data available to E1 consumers within a few seconds of datataking; it feeds fast monitoring and fast processing. Candidate TF delivery mechanisms are messaging and direct XRootD reads against the exit buffer.

Control signals cross the interface alongside the data. The run lifecycle — run imminent, run start, pause and resume, run end — is broadcast from the DAQ side and drives downstream orchestration: dataset creation, processing task establishment, worker provisioning, and run closeout all key off these transitions.

3.1.2 Time Frames and Super Time Frames

The time frame (TF) is the atomic unit of ePIC streaming data: a contiguous, self-contained slice of the detector data stream. Super time frames aggregate consecutive time frames into file-sized units that serve as the unit of registration, transfer, bookkeeping, and bulk processing. The STF is what Rucio registers and moves, what run datasets collect, and what prompt processing consumes.

[ToC]

The two units define the two latency regimes. Full STFs carry the complete data sample on the timescale of file creation and transfer. TF-level data serves the fast paths: TF subsamples can be formed in the DAQ enclave in parallel with STF building, or skimmed from STFs sitting in the exit buffer,

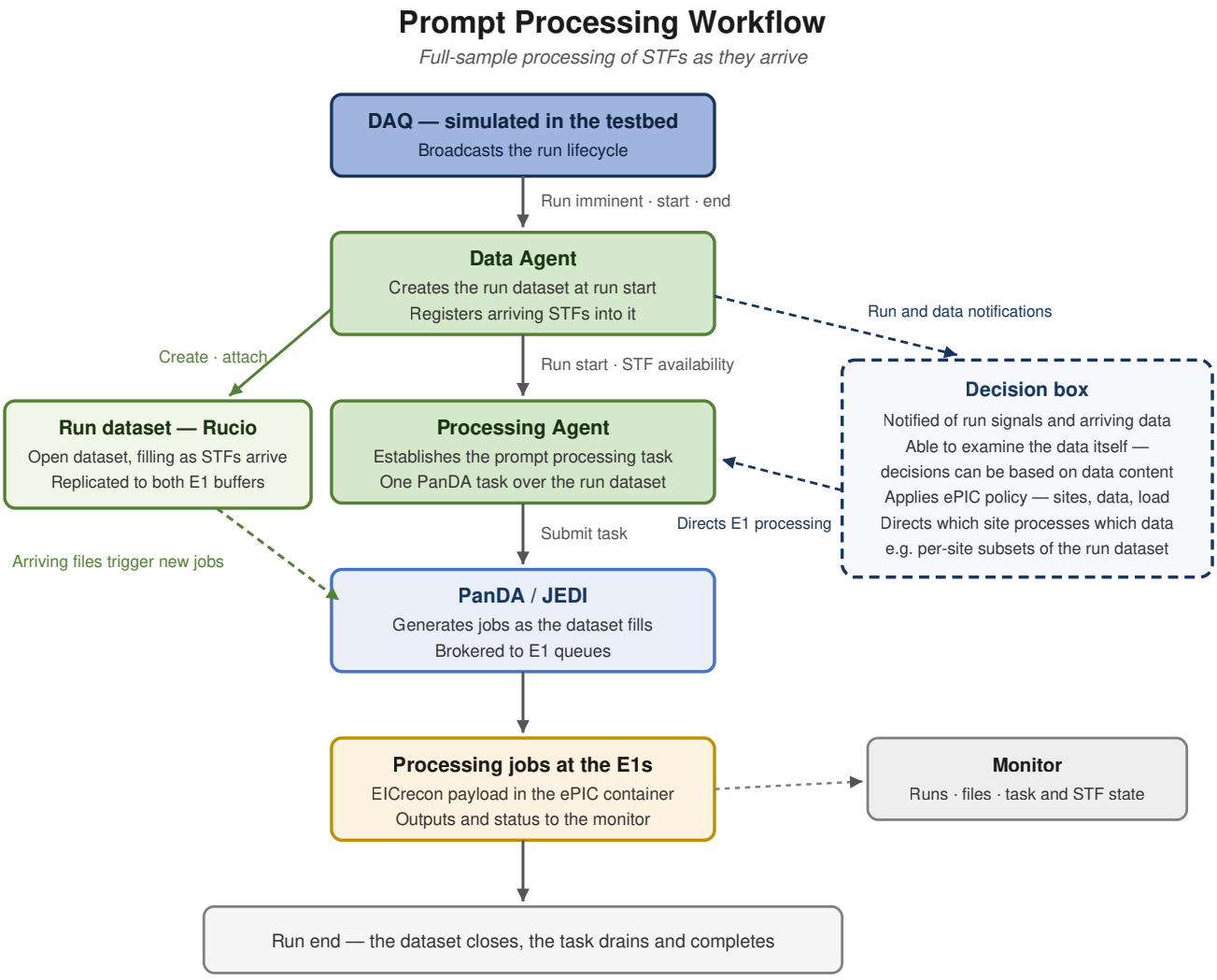
and are small enough to deliver and process within seconds. Downstream, sampled TFs are further divided into TF slices, the parallel work units distributed to fast processing workers.

3.1.3 Prompt STF Processing

Prompt processing is the full-sample processing path: STFs are processed at the E1 facilities as they arrive, delivering complete first-pass results over minutes to hours. At run start a Rucio dataset is created for the run; arriving STFs are registered into it and transferred to the E1 buffers. A processing task is established for the run in PanDA, and jobs process the STFs as the dataset fills. Results serve detector and physics evaluation well beyond what the fast path's sampled data supports. In early datataking the full STF sample is likely to be promptly processed, while the detector and software are being debugged and understood; as luminosity and understanding grow, the promptly processed fraction is expected to decline, a datataking-era policy choice the decision box below is designed to carry.

The prompt processing resource pool is E1 in the baseline and can extend to E2 facilities as capability and policy allow; PanDA brokering over queues and Rucio-managed data placement make wider distribution a configuration choice rather than a workflow redesign.

The workflow is diagrammed below, including the prompt processing decision box — the conceptual control point, notified of run signals and arriving data and able to examine the data itself, that applies ePIC policy to direct which site processes which data.



3.1.4 Fast Processing Pipeline

[ToC]

Fast processing exists for latency: first results from the data stream in O(10 s) to inform control room operations and AI tools of current detector and machine performance. TF samples are skimmed from arriving STFs, divided into TF slices, and distributed to a standing pool of workers running the

reconstruction payload — EICrecon for ePIC production, now being integrated into the testbed workers. Slice results flow to low-latency analytics and monitoring consumers.

The latency budget rules out provisioning workers on demand. The pipeline pre-provisions a configurable worker pool at run start: run-imminent signals carry the target worker count, iDDS and Harvester establish semi-persistent PanDA worker jobs on the compute resources, and the workers consume slices for the duration of the run and exit at run end. Slice-level state — queued, processing, completed, failed with bounded retry — is tracked in the monitor database.

3.1.5 Streaming Reconstruction Integration

Streaming reconstruction itself is not WFMS scope: EICrecon, its configuration, and its physics performance belong to ePIC software. The WFMS integrates reconstruction as the payload of streaming processing, and the integration is a workflow concern in its own right, with different requirements in the two latency regimes.

Prompt processing integrates reconstruction conventionally: EICrecon processes STF files as PanDA jobs in the ePIC container environment distributed over CVMFS — the same payload environment production uses. Prompt processing with an EICrecon reconstruction payload has run successfully in the testbed.

Fast processing cannot pay a per-slice startup cost: the payload must run as a standing process that accepts work as it arrives. This integration is an area of active development, in collaboration with EICrecon developers at JLab. The worker transformation (`swf-transform`) runs EICrecon as a persistent process and feeds it slice work over ZeroMQ messaging; the worker lifecycle layer (`swf-panda-workers`) provisions and scales the worker pool through iDDS and PanDA on run lifecycle signals and observed slice processing times. The payload capabilities this demands — event windowing directed by messages, remote input over XRootD, and clean process termination — are contributed upstream to EICrecon, and message-driven EICrecon is available in the ePIC container stack. The integration is exercised against real campaign simulation outputs.

3.1.6 Monitoring and Validation

Fast monitoring consumes sampled TF data at the E1s for near-real-time detector and data quality: fast monitoring agents read remotely against the exit buffer or receive delivered samples, and their outputs are available within seconds of datataking. Prompt monitoring runs against the full STF sample as it is processed. Both feed control room displays, automated quality checks, and AI analytics, and both are candidates for E2 consumers of the monitoring streams.

The workflows themselves are monitored through the platform's operational state: runs, files, workflow executions, messages, agent status, and slice bookkeeping are recorded in the monitor database and presented in live browser views. Streaming-side validation operates at the fast end of the validation latency range, evaluating detector performance and data quality from the first samples; broader validation, through full calibration cycles, is described in the Validation section.

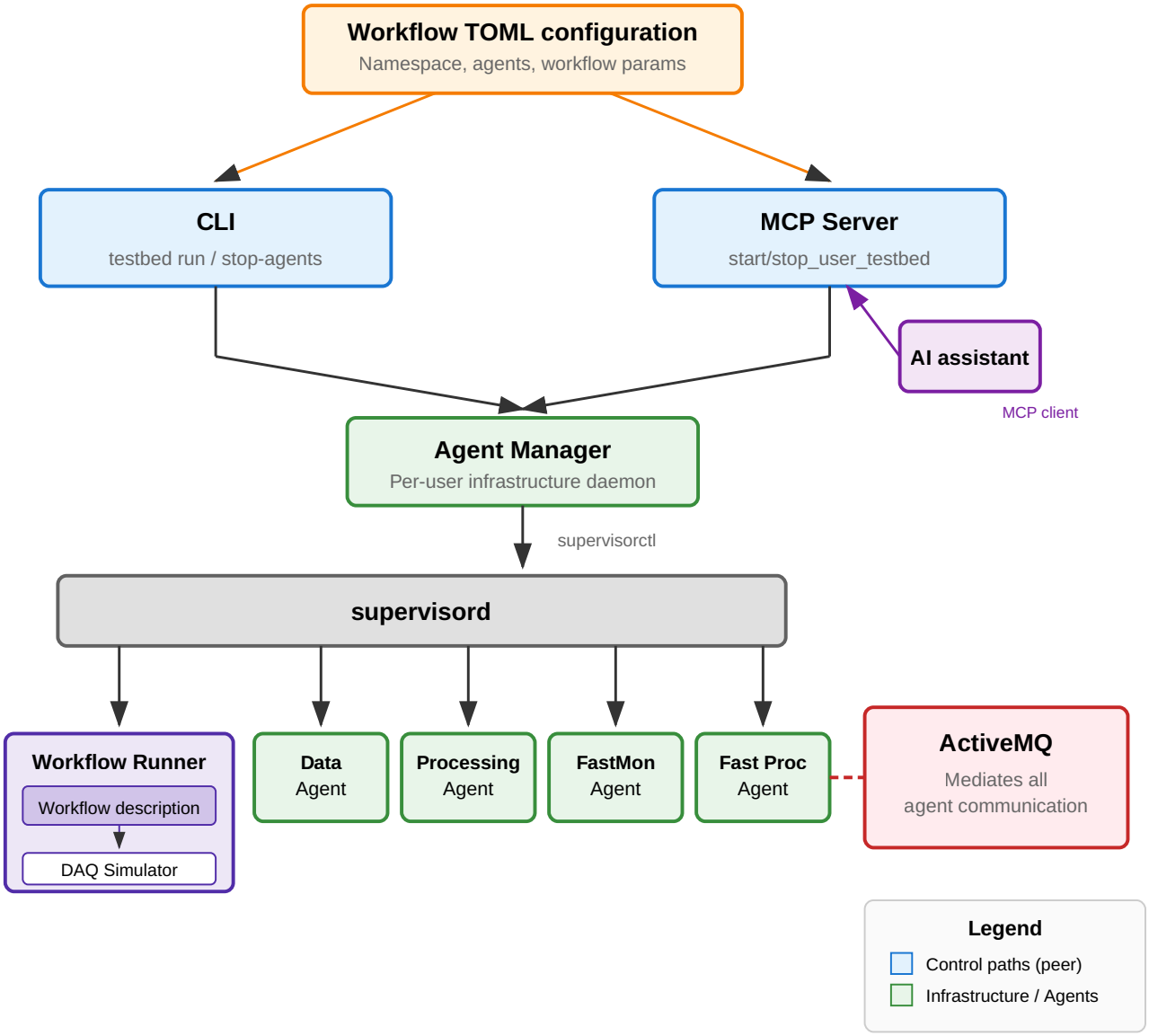
3.1.7 Streaming Workflow Testbed

The streaming workflow testbed is the current realization of these workflows. It prototypes the ePIC streaming model from E0 egress — the DAQ exit buffer — through processing at the two E1 facilities, exercising workflow and dataflow logic on real services (PanDA, Rucio, ActiveMQ, the monitor) with emulated facilities and simulated datataking, within the scope marked in the schematic above. Its architecture and agent design are documented in the testbed architecture overview.

A simulated DAQ drives the system: `swf-daqsim-agent` models detector, machine, and DAQ influences, generates the run lifecycle and STF stream, and is the primary driver of testbed activity. `swf-data-agent` is the central data handler, creating run datasets in Rucio, registering and attaching STF files to them, and notifying downstream consumers; a watcher role detecting stalls and anomalies is planned. `swf-processing-agent` establishes and manages the PanDA prompt-processing tasks. `swf-fastmon-agent` samples TF-level data from available STFs and records fast-monitoring metadata. A fast processing agent creates TF slices from the samples, broadcasts the run and target worker count to the worker layer to provision the standing pool, distributes slices, and collects results.

The agents are configured, launched, and supervised through a common management layer, controlled from the CLI and, equivalently, by AI assistants through MCP:

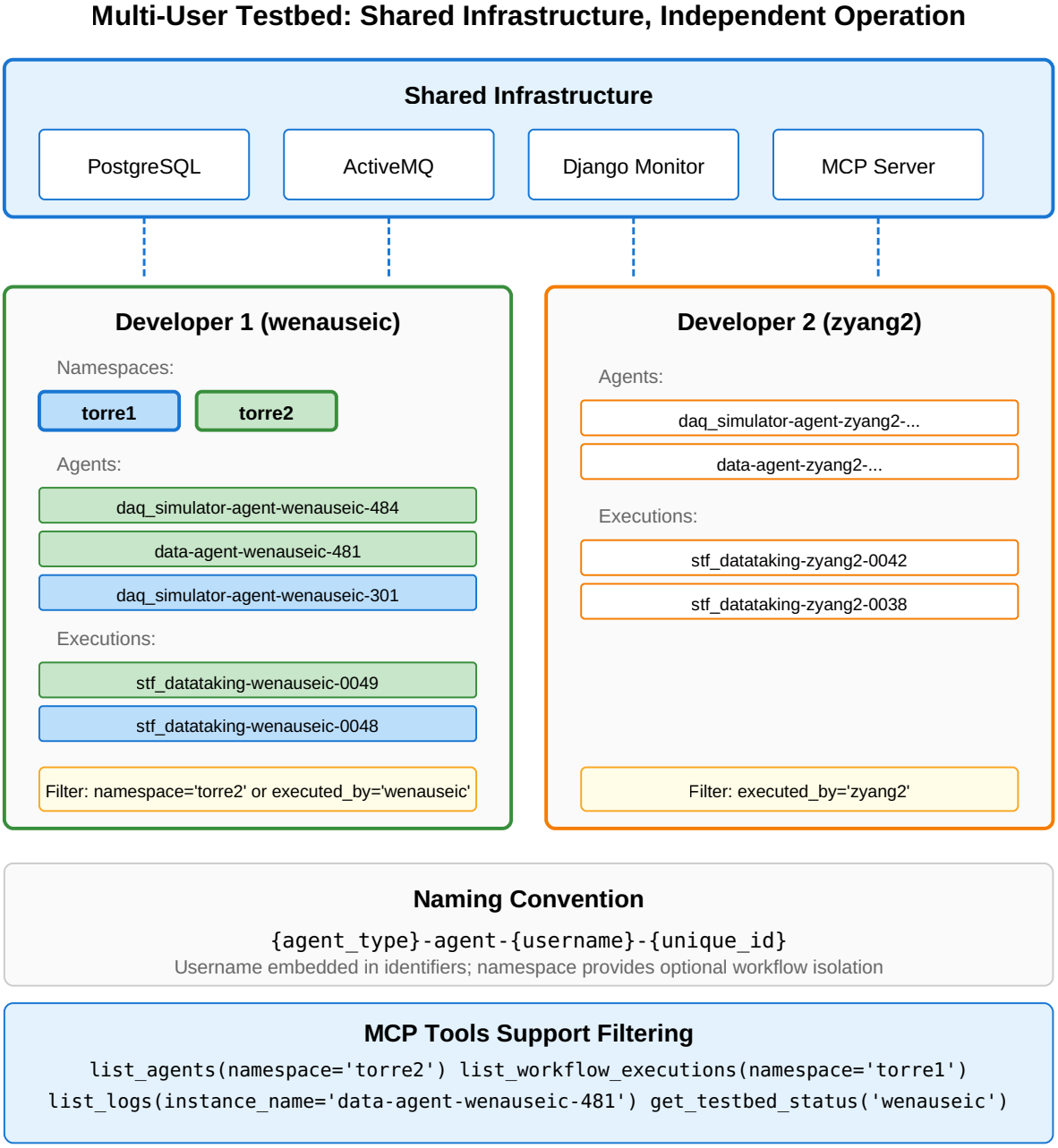
[ToC]



Two streaming workflows are realized today. The prompt processing workflow takes simulated runs from run-imminent through dataset creation, STF registration, and PanDA task submission over the run dataset. The fast processing workflow takes the same runs through TF sampling, slice creation, and slice processing on the pre-provisioned worker pool. Both are driven by TOML workflow configurations and tracked end to end in the monitor.

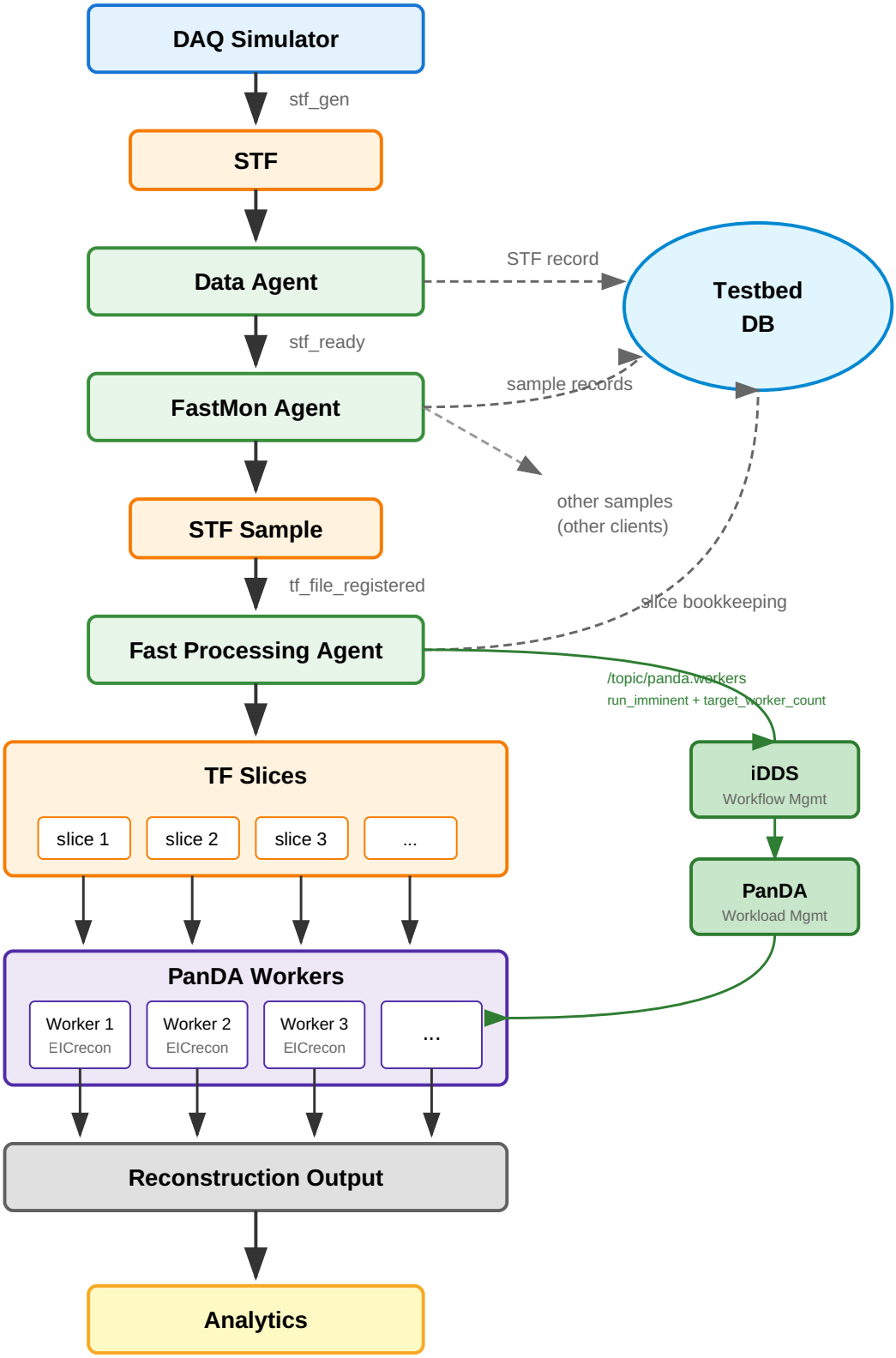
Concurrent testbed users share one infrastructure and operate independently, isolated by namespace and per-user agent identity:

[ToC]



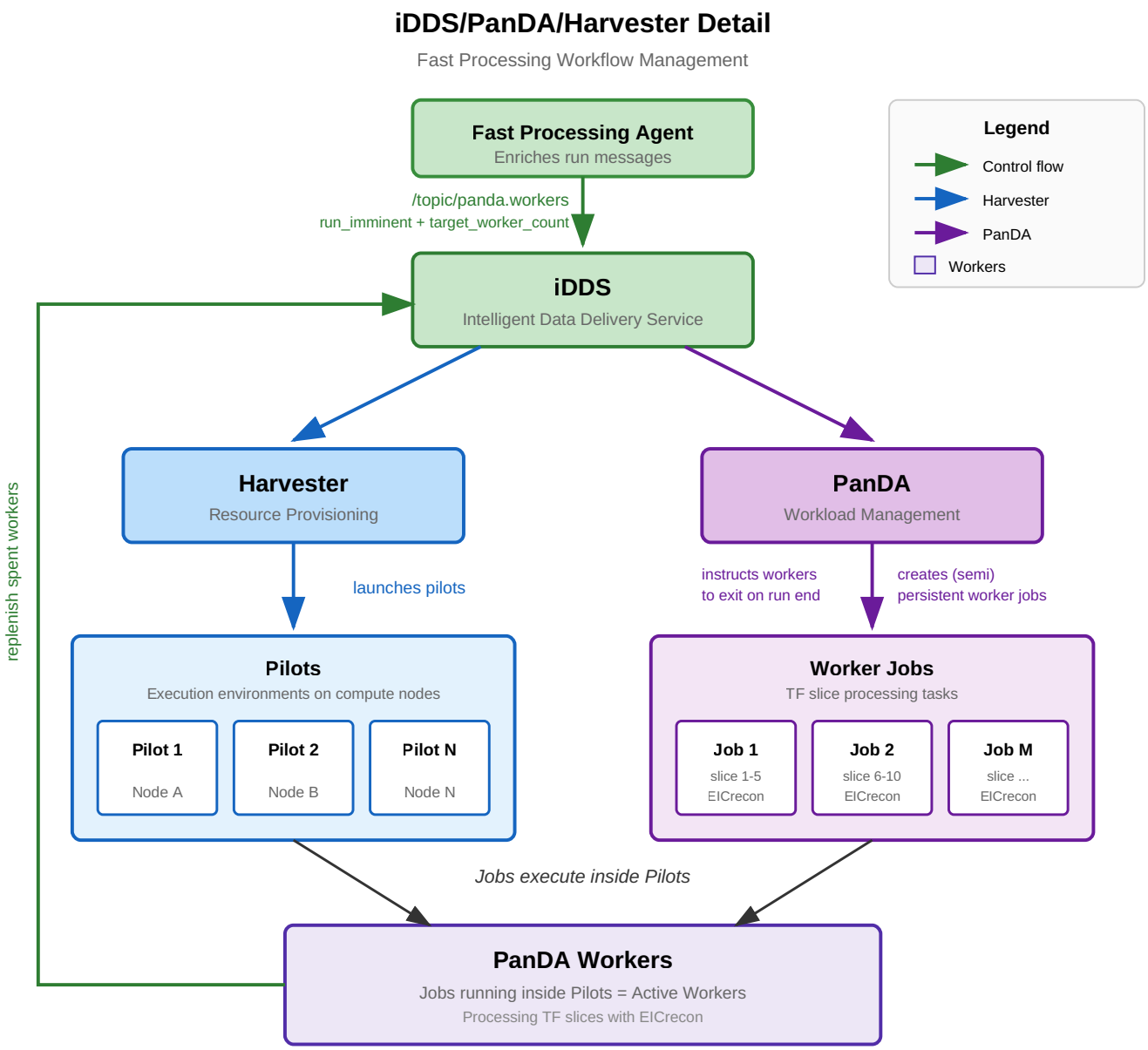
The fast processing pipeline is diagrammed below — the agent pipeline from simulated DAQ to PanDA workers; the integration of the real EICrecon payload into these workers is described in Streaming Reconstruction Integration above.

[ToC]



[ToC]

The iDDS/PanDA/Harvester detail behind the standing worker pool:



1. The ePIC Streaming Computing Model. https://zenodo.org/records/14675920 ↩

[ToC]

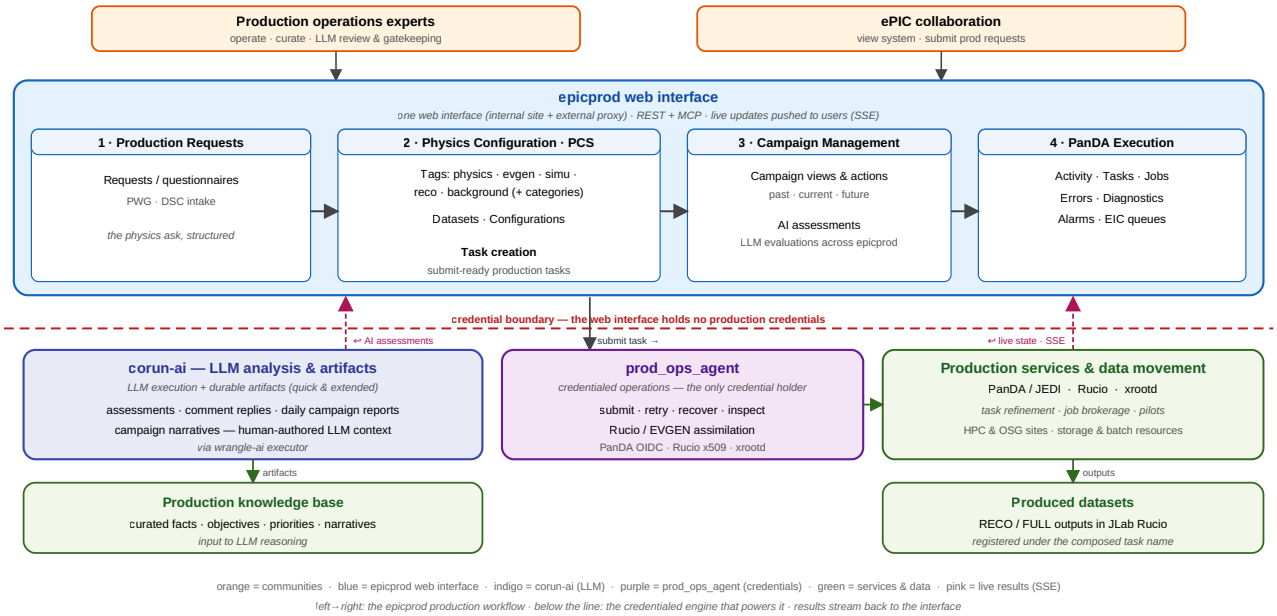
3.2 Production System

This section documents epicprod, the ePIC automated production system: request intake, physics configuration, campaign management, PanDA execution, data products, LLM assessments and reports, and the human controls and curation that steer it.

3.2.1 epicprod Overview

epicprod is the automated production system for ePIC simulation and reconstruction campaigns. It carries a production request from community submission through physics configuration, campaign preparation, PanDA execution, and cataloged data products, with AI assistance and human control points along the whole path. The operating principle set in Foundations, maximal automation with minimal operations effort, shapes each stage: the routine mechanics are automated, and people make the decisions. The implementation lives in the swf-monitor and swf-epicprod repositories — production-specific components and documentation are consolidating in swf-epicprod, running on the platform services of swf-monitor — whose documentation carries the design and operational detail behind this section. The definitive current (June 2026) description of the system as deployed — status, interface views, and near-term plans — is the S&C talk ePIC Production System: Status and Plans.

epicprod — the ePIC Production System



3.2.2 Production Requests

Physics working groups and detector groups request production datasets through the collaboration's request form. PCS mirrors each form response into a read-only questionnaire record, giving the collaboration a browsable view of all requests and giving production records a single upstream reference for request provenance (questionnaire design). The request cycle follows the collaboration's monthly campaign rhythm: requests are coordinated through the physics working groups and detector subsystem collaborations, validated by the production working group, and prioritized through the collaboration's physics and technical coordination before the monthly production list is frozen for the campaign.

Triage turns submissions into structured production records. An operator links a response to one or more production request records — one submission frequently spans several beam energies or Q^2 ranges — and composes each request from PCS tags and a production configuration. A request is deliberately more abstract than an output: it specifies the physics, beam energies, and kinematic range, and one request commonly leads to several produced datasets. Provenance is by reference: a task resolves through its request to the originating submission, contact, and estimates without duplicating them.

[ToC]

3.2.3 Physics Configuration System (PCS)

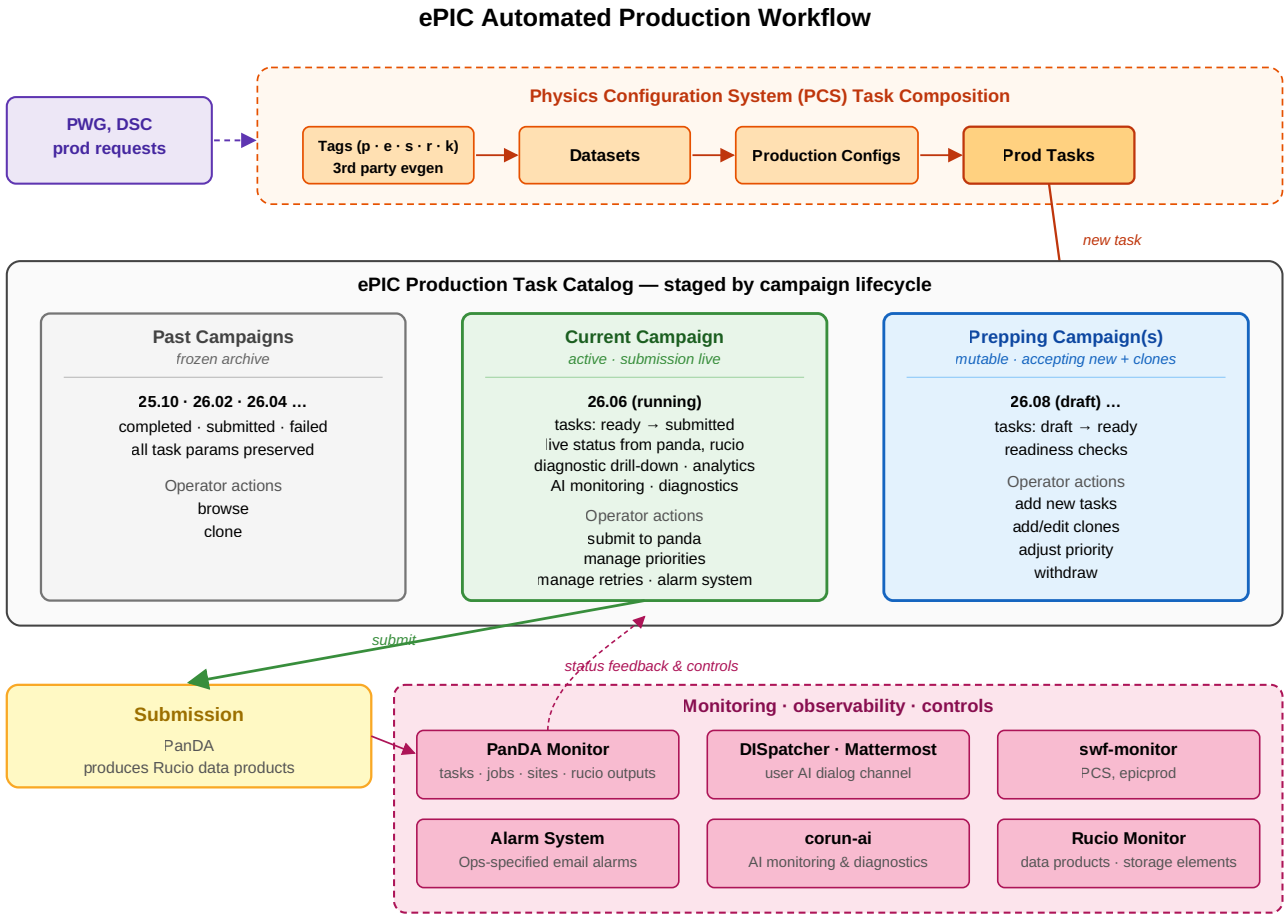
PCS is the configuration layer of epicprod: the catalog of physics and processing definitions — tags, datasets, sample variants, and production configs — from which production tasks are composed, with identity carried through composed names. As the principal place where physicists meet the production system, it has its own section: Physics Configuration System.

3.2.4 Campaign Management

Campaigns are time-ordered groupings of production tasks, named by year and month (25.10, 26.06). The production task catalog is the instrument of campaign management: tasks composed from PCS configuration entities, carrying their full parameter sets, staged by campaign lifecycle (task catalog design).

Prepping campaigns are mutable: operators add new tasks and clones, adjust priorities, withdraw entries, and move tasks from draft to ready under readiness checks. The current campaign is the live one: ready tasks are submitted, and the catalog presents live status from PanDA and Rucio with diagnostic drill-down, analytics, and AI monitoring. Past campaigns are a frozen archive with all task parameters preserved. Pre-PCS production has been assimilated into past campaigns, making the catalog the complete ePIC production record.

The catalog stays complete by construction: a nightly sweep reads recent ePIC tasks from the PanDA task database, associates them with their catalog entries, and assimilates any production submitted outside the catalog workflow — creating the campaign, dataset, and task records with the physics classification derived from the task identity. Tasks born in the catalog carry full configuration and request lineage; adopted tasks carry what assimilation can derive, and the adopted count is a standing measure of migration onto the campaign-task flow.



Each campaign carries a human-authored campaign narrative recording its goals, priorities, and evolution. The narrative gives operators and LLMs a shared context: daily campaign reports and assessments reason against it, and it accumulates the campaign's history as decisions are made.

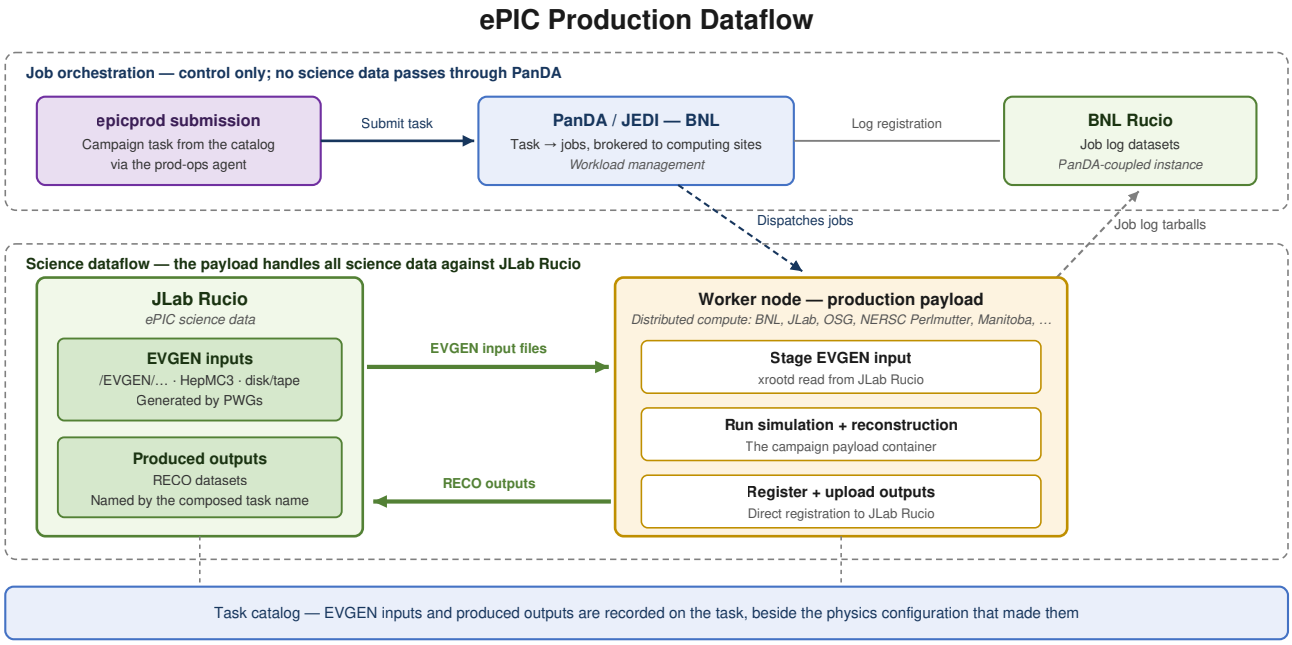
[ToC]

3.2.5 PanDA Execution

Task submission is a catalog action: the composed task specification is submitted through the production operations agent to PanDA, and each physical submission attempt is recorded on the task as a PanDA task association. The submission mechanics, task naming of retries, and the science-data constraint that has production payloads stage EVGEN inputs from and register outputs to JLab Rucio are described in Platform.

Live execution state flows back to the catalog and the ePIC PanDA monitoring views: task and job status, error summaries, queue and site state, and resource usage. Payload logs are retrievable on demand from the task pages. Operators steer execution with retry, recovery, and priority controls, and the alarm system carries execution conditions that need attention.

3.2.6 Data Products



Produced data are cataloged as the primary product of the system. Outputs are registered in JLab Rucio under the composed task name: RECO reconstruction outputs and the associated log datasets. The catalog gathers this lineage onto the task record, linking expected and observed outputs with their Rucio identifiers, status, and access references, so a physicist can go from a physics configuration to its data products in one place.

Third-party event-generation inputs are cataloged on the same footing. EVGEN datasets in JLab Rucio are swept into the catalog and matched to the requests they serve, so a task's inputs and outputs are both visible beside the configuration that consumes and produces them. On-the-fly event generation in the production payload is the planned evolution of the input path, replacing staged inputs where generators support it, beginning with a Pythia8 proof of concept.

3.2.7 AI Assistance

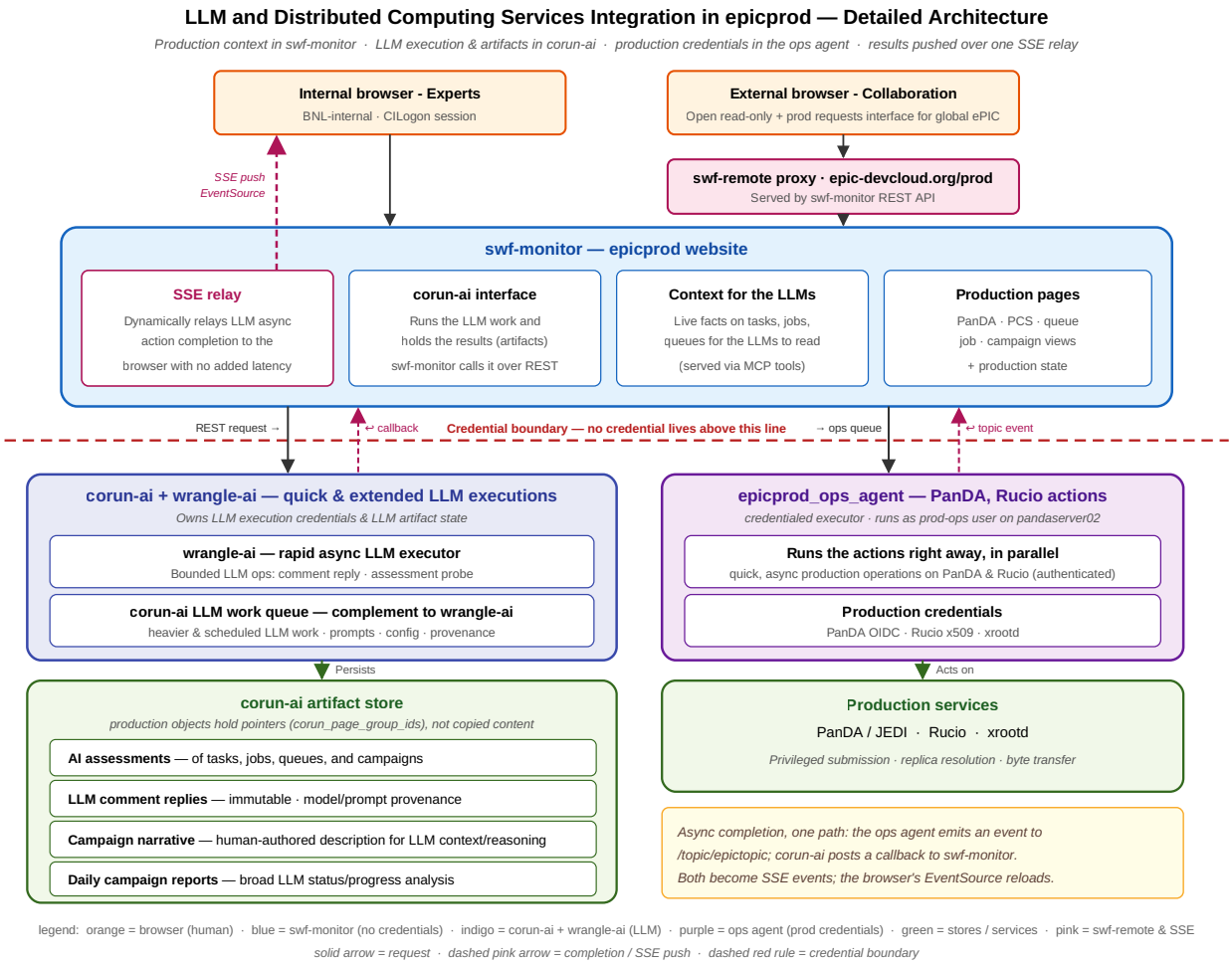
AI assistance appears throughout the production pages as durable, attributed artifacts. Assessments evaluate tasks, jobs, queues, and campaigns and are attached to the objects they assess. Comment threads on production objects carry discussion between operators, and an LLM participates in the thread on request. Daily campaign reports analyze status and progress against the campaign narrative. Each artifact records its model and prompt provenance and is open to comment, following the Architecture rule that AI outputs become artifacts in the system.

The assistance is delivered through the corun-ai and wrangle-ai services described in Platform: a page action requests the LLM operation, and the result is pushed to the requesting page the moment it completes. A production bot provides a conversational channel to the same system state for questions and diagnostics in chat.

[To]

Detailed epicprod LLM Operations Architecture

The architecture of the LLM operations path — the production pages, the LLM services, the credentialed operations agent, and the notification return — is diagrammed below.



3.2.8 Human Controls and Curation

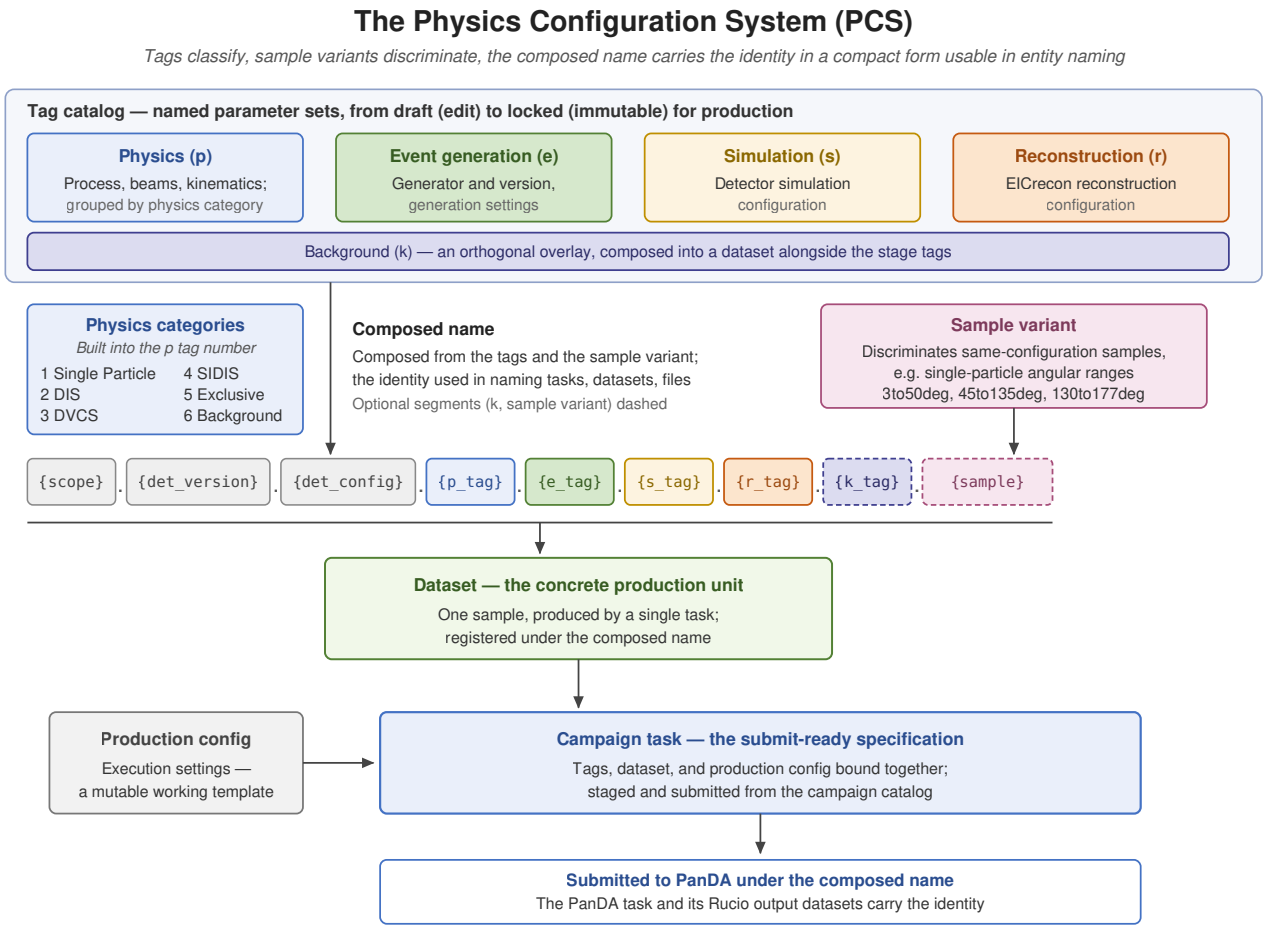
Canonical state changes by human hand. Lifecycle transitions — draft to ready, submission, lock, withdraw, retry and recovery — are operator decisions exercised through catalog controls. Automation and AI may detect a condition, present an interpretation, and surface the control that acts on it, and the human decides. Read access to the production pages is open to the collaboration on both the internal and external faces; actions are gated by login and role, and privileged execution stays behind the operations agent.

Curation is the human side of the automation bargain. Production experts maintain the campaign narratives, the curated knowledge the LLMs reason from, and the priorities that steer the work; they review AI assessments and reports rather than inherit them as fact. The system records these judgments — narratives, comments, decisions — so that the context available to the next assessment, human or AI, keeps improving.

[ToC]

3.3 Physics Configuration System

This section documents the Physics Configuration System (PCS), the configuration layer of epicprod and the principal place where physicists meet the production system: the tag catalog of physics and processing definitions, the datasets and sample variants composed from them, the composed-name identity, and the browse and compose interface from which production tasks are built and submitted.



3.3.1 The Configuration Layer

PCS is the catalog of physics and processing definitions from which production tasks are composed, and the system of record for what those definitions mean. Like all of epicprod it is part of the ePIC PanDA infrastructure at BNL, proxied for open collaboration access, and it serves PanDA use beyond production: the streaming workflow testbed and the AID2E detector-design project compose PanDA work from the same catalog. The working vocabulary — tags, datasets, sample variants, production configs, and the composed name — is defined in Concepts; this section describes the system built on it, with implementation detail in PCS.md.

3.3.2 Tags and the Tag Catalog

Four tag types classify the stages of production — physics (p), event generation (e), simulation (s), and reconstruction (r) — organized under physics categories, each tag carrying a parameter set appropriate to its type. The fifth type, background (k), is an orthogonal overlay: backgrounds compose into datasets alongside the stage tags, keeping signal and background configurations independent and avoiding the tag explosion of signal-background combinatorics.

[ToC] Tags follow the draft-refine-lock lifecycle: a draft tag is editable by its owner, and a locked tag is immutable and production-ready — a stable reference whose meaning never changes. The whole ePIC production record, current and past campaigns alike, is expressed on the tag basis, so the catalog is both the configuration source for new production and the classification of everything already produced.

3.3.3 Datasets, Variants, and Identity

Datasets bind tags into a defined data product — one sample, produced by a single task — with sample variants discriminating same-configuration samples produced separately. The composed name built from these entities carries the identity in a compact form usable in entity naming — tasks, datasets, and files: catalog pages, links, the API, the PanDA task name, and the Rucio output namespace all carry it, as diagrammed above and specified in Concepts. Production configs complete the picture on the execution side: reusable, deliberately mutable templates of software stack, resource, and splitting settings.

3.3.4 Browse and Compose

The PCS interface is a two-pane browse and compose view, open read-only to the whole collaboration: a filterable entity browser — search, parameter selectors, tag type selection — beside the detail and edit pane for the selected entity. Ownership sets what a login can do: owners can edit, copy, lock, and delete their draft entities, and anyone can copy anything, including entities they do not own.

Copy and edit is the intended working style. A physicist copies the tag closest to what they need — the copy takes a new auto-incremented label — edits the parameters, choosing from configured values or adding custom ones, and saves. While composing, browsing other tags shows field-by-field which values differ from the entity being edited, and a differing value can be adopted with a click. The same derivation style scales up: a campaign's tasks can be cloned and adjusted for the next campaign rather than rebuilt.

3.3.5 From Tags to Submitted Tasks

Task composition binds tags, dataset, and production config into a campaign task — the submit-ready specification described in Production System. Submission is one click from the compose view: the task specification is generated from the composed entities and submitted through the credentialed production operations agent using the PanDA client API, and the returned PanDA task ID is recorded on the campaign task for tracking and monitoring.

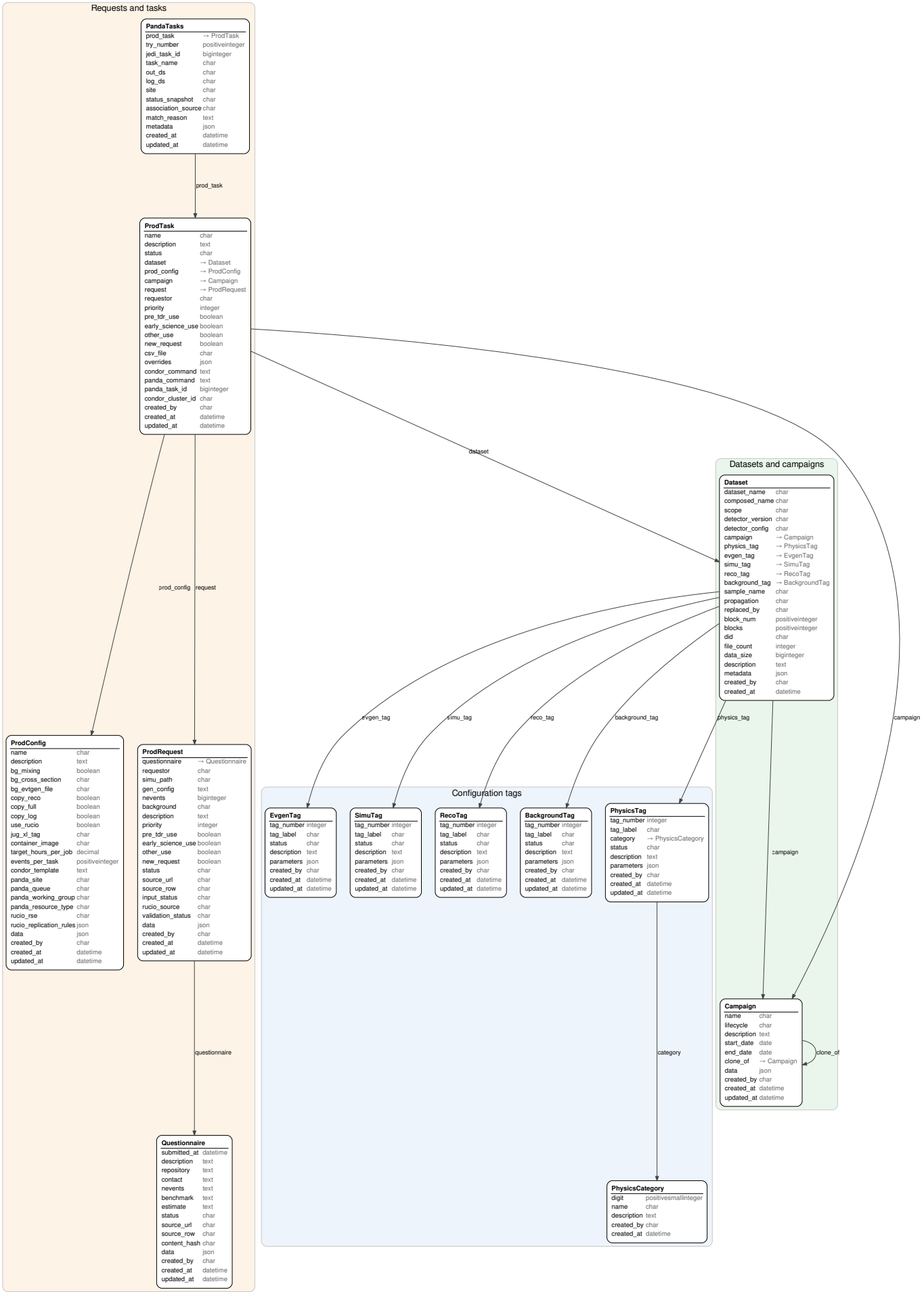
3.3.6 From Request to Configuration

Production requests reach PCS through the triage described in Production System: an operator links the request to the tags and datasets that realize it. The mapping is a human operations process at present; as the request form becomes more structured, the request-to-configuration mapping becomes more programmatic.

3.3.7 The PCS Data Model

The data model as implemented, generated from the live Django models: every entity with all of its fields and relations. The same schema in database form is the repository's generated testbed-schema.dbml.

[ToC]


Requests and tasks
PandaTasks
prod_task → ProdTask
try_number positiveinteger
jedi_task_id biginteger
task_name char
out_ds char
log_ds char
site char
status_snapshot char
association_source char
match_reason text
metadata json
created_at datetime
updated_at datetime
prod_task
ProdTask
name char
description text
status char
dataset → Dataset
prod_config → ProdConfig
campaign → Campaign
request → ProdRequest
requestor char
priority integer
pre_tdr_use boolean
early_science_use boolean
other_use boolean
new_request boolean
csv_file char
overrides json
condor_command text
panda_command text
panda_task_id biginteger
condor_cluster_id char
created_by char
created_at datetime
updated_at datetime
dataset
prod_config
request
Datasets and campaigns
Dataset
dataset_name char
composed_name char
scope char
detector_version char
detector_config char
campaign → Campaign
physics_tag → PhysicsTag
evgen_tag → EvgenTag
simu_tag → SimuTag
reco_tag → RecoTag
background_tag → BackgroundTag
sample_name char
propagation char
replaced_by char
block_num positiveinteger
blocks positiveinteger
did char
file_count integer
data_size biginteger
description text
metadata json
created_by char
created_at datetime
evgen_tag
simu_tag
reco_tag
background_tag
physics_tag
campaign
ProdConfig
name char
description text
bg_mixing boolean
bg_cross_section char
bg_evtgen_file char
copy_reco boolean
copy_full boolean
copy_log boolean
use_rucio boolean
jug_xl_tag char
container_image char
target_hours_per_job decimal
events_per_task positiveinteger
condor_template text
panda_site char
panda_queue char
panda_working_group char
panda_resource_type char
rucio_rse char
rucio_replication_rules json
data json
created_by char
created_at datetime
updated_at datetime
ProdRequest
questionnaire → Questionnaire
requestor char
simu_path char
gen_config text
nevents biginteger
background char
description text
priority integer
pre_tdr_use boolean
early_science_use boolean
other_use boolean
new_request boolean
status char
source_url char
source_row char
input_status char
rucio_source char
validation_status char
data json
created_by char
created_at datetime
updated_at datetime
Configuration tags
EvgenTag
tag_number integer
tag_label char
status char
description text
parameters json
created_by char
created_at datetime
updated_at datetime
SimuTag
tag_number integer
tag_label char
status char
description text
parameters json
created_by char
created_at datetime
updated_at datetime
RecoTag
tag_number integer
tag_label char
status char
description text
parameters json
created_by char
created_at datetime
updated_at datetime
BackgroundTag
tag_number integer
tag_label char
status char
description text
parameters json
created_by char
created_at datetime
updated_at datetime
PhysicsTag
tag_number integer
tag_label char
category → PhysicsCategory
status char
description text
parameters json
created_by char
created_at datetime
updated_at datetime
campaign
Campaign
name char
lifecycle char
description text
start_date date
end_date date
clone_of → Campaign
data json
created_by char
created_at datetime
updated_at datetime
clone_of
questionnaire
category
Questionnaire
submitted_at datetime
description text
repository text
contact text
nevents text
benchmark text
estimate text
status char
source_url char
source_row char
content_hash char
data json
created_by char
created_at datetime
updated_at datetime
PhysicsCategory
digit positivesmallinteger
name char
description text
created_by char
created_at datetime

[ToC]