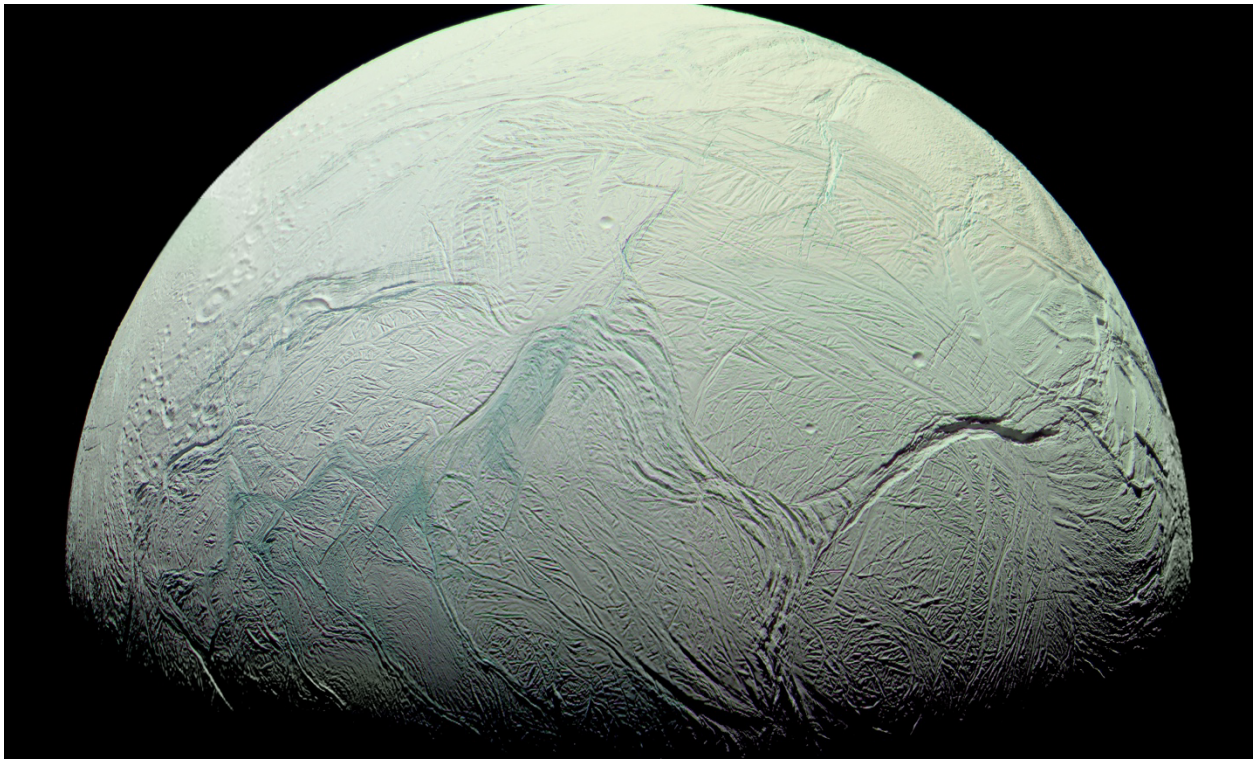
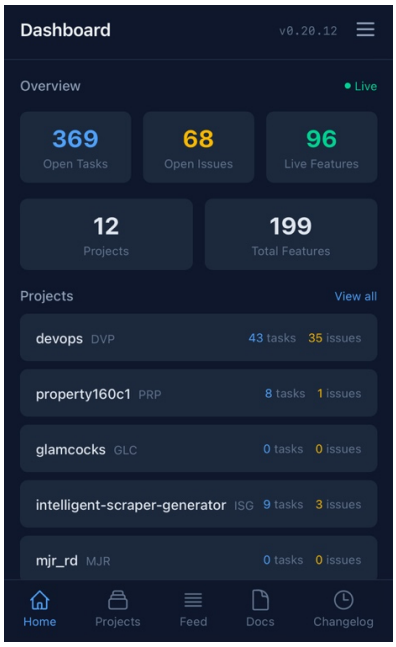


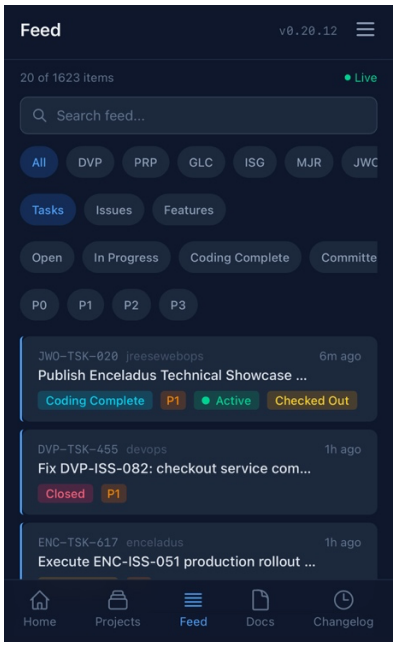
# ENCELADUS

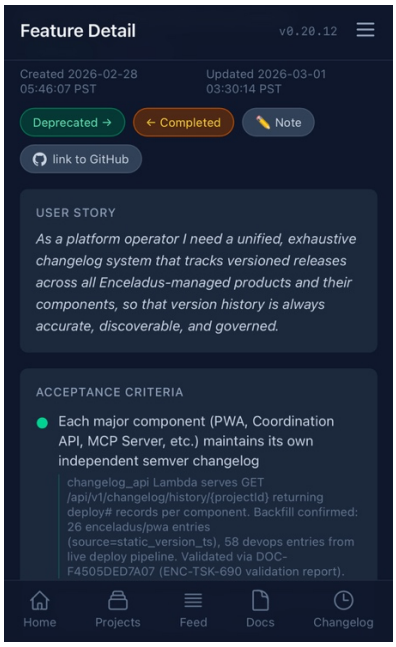
## Product Portfolio Operations Platform

Comprehensive Product Narrative & Technical Architecture
v0.17.0 — February 2026

Author: Claude with direction by J Reese



# Enceladus Launches Unified Agent Coordination Platform

*Serverless portfolio operations system, originally built to coordinate personal project infrastructure, now enables any AI agent to read from and write to a complete project portfolio with governed lifecycle guarantees.*

**San Francisco, CA —** INSIDE ENCELADUS — February 26, 2026 — Managing a portfolio of production systems across multiple domains—genealogy applications, web infrastructure, development operations—while coordinating AI agents from different providers to do meaningful work on them is a challenge that has taken over a year of continuous development to solve. Today, **Enceladus v0.17.0** ships with a fully governed deployment lifecycle, an abstracted checkout service, and a remote MCP server accessible from Claude.ai—completing the transition from a simple project tracker into a comprehensive agent coordination platform. With this release, the platform manages **5 active production projects**, **37 features**, **682+ closed tasks**, and **55 knowledge documents**, all orchestrated through a single governance framework that enforces ontological discipline at every write boundary.

Until today, coordinating work across AI agents from different providers—Claude, OpenAI Codex, AWS Bedrock—required manual handoffs, redundant context loading, and no guarantees that one agent's work wouldn't collide with another's. Each provider had its own conventions, its own session model, and no shared understanding of what a “task” or “feature” actually meant in operational terms. Documentation lived in scattered markdown files. Deployment evidence existed only in CI logs.

Enceladus addresses this by treating project management primitives—Features, Tasks, Issues—not as database records, but as ontologically defined objects with governed lifecycles, required evidence gates, and deterministic completion contracts. Every mutation flows through a single MCP server with 35+ tools, enforcing governance hash authorization on every write. The checkout service ensures only one agent owns a task at a time. The deployment pipeline validates PR merge evidence against the GitHub API before accepting any deployment request.

> *“The insight that changed everything was treating our entities as conceptual objects rather than rows in a table. A Feature isn't just a record—it has a user story, acceptance criteria, and a governed completion handshake. That ontological discipline cascades through every operational mechanism in the system.”*

Enceladus is built entirely on AWS serverless infrastructure—**19 Lambda functions**, **7 DynamoDB tables**, **2 SQS FIFO queues**, and a **CloudFront CDN**—running at approximately $30–40/month. The React 19 PWA at **jreese.net/enceladus** provides mobile-first visibility into all project state, while the MCP server enables any AI agent with the right credentials to participate in governed workflows. The platform ships with CI/CD automation across 5 GitHub Actions workflows, nightly parity audits, and a secrets guardrail scanning every commit.

# Background: The Coordination Problem

The modern software practitioner increasingly relies on AI agents as force multipliers. Claude Code writes and commits. OpenAI Codex reasons over large codebases. AWS Bedrock agents execute infrastructure operations. But these capabilities exist in silos—each agent starts cold, unaware of what other agents have done, what the current project state is, or what governance rules apply.

For a solo operator managing multiple production systems, this fragmentation compounds. A genealogy site redesign, a CDN configuration update, and a deployment pipeline enhancement all require different context, different tools, and different verification steps. Without a unified operations layer, the human becomes the sole integration point—manually transferring context, resolving conflicts, and validating completeness.

Enceladus was built to eliminate this bottleneck. It serves as the shared memory and governance authority across all projects and all agent sessions, ensuring convergence toward production-quality outcomes regardless of which provider executes the work.

## The Philosophy: Ontological Discipline

The foundational insight driving Enceladus is that proper ontological definition of system primitives is essential for building maximally abstracted, extensible components. This philosophy manifests in three core principles:

- **Features are solutions, not records.** A feature represents a technical solution to a problem or opportunity that enables a user to accomplish something. Every feature requires a user story (“As a [USER/SYSTEM] I need to [X] so that [Y]”), at least one testable acceptance criterion, and a governed completion handshake where evidence is submitted against each criterion before the feature can close.
- **Tasks are discrete units of agent-executable work.** A task is defined so that an abstracted terminal agent session, provided with minimal but adequate acceptance criteria, can successfully complete it. Parent/child hierarchies organize sequences of work; the checkout service ensures exclusive ownership; and the lifecycle state machine (open → in-progress → coding-complete → committed → pr → merged-main → deploy-success → closed) enforces evidence at every gate.
- **Issues are evidence-backed observations.** Every issue requires at least one evidence object with description and steps-to-duplicate, enabling any agent or human to verify the observation independently. This prevents phantom issues and ensures collaborative resolution.
- **Documents are reusable institutional memory.** Documentation captures discoveries, decisions, and patterns that might otherwise be lost between sessions. They prevent token waste on previously-explored approaches and serve as the backbone of the platform's learning loop.

# System Architecture

Enceladus is a fully serverless AWS platform optimized for low-cost, high-reliability operations. The architecture separates concerns across five layers: data persistence, compute, API routing, CDN distribution, and agent integration.

## Technology Stack

| Layer | Technologies |
|---|---|
| Frontend | React 19.2, TypeScript 5.9, Vite 7.3, Tailwind CSS 4.1, TanStack React Query 5.90 |
| Backend | Python 3.11, AWS Lambda (19 functions), API Gateway HTTP API v2 |
| Data | DynamoDB (7 tables, PAY_PER_REQUEST), S3 (jreese-net bucket) |
| Auth | AWS Cognito (RS256 JWT), Lambda@Edge, service-to-service key |
| Messaging | SQS FIFO (2 queues), SNS (4 topics), EventBridge (3 rules, 2 pipes) |
| CDN | CloudFront distribution at jreese.net |
| CI/CD | GitHub Actions (5 workflows), CodeBuild, per-Lambda deploy scripts |
| AI/Agents | MCP server (35+ tools), Bedrock Agent, Anthropic API, OpenAI API |

## Infrastructure Inventory

| Resource | Count | Details |
|---|---|---|
| Lambda Functions | 19 | 18 in us-west-2, 1 Lambda@Edge in us-east-1 |
| DynamoDB Tables | 7 | All PAY_PER_REQUEST with deletion protection |
| SQS FIFO Queues | 2 | Deploy queue (180s) + Feed publish queue (900s) |
| API Routes | 30+ | All via HTTP API Gateway, Payload Format 2.0 |
| MCP Tools | 35+ | Across 8 domains: projects, tracker, docs, deploy, etc. |
| Frontend Pages | 15 | Mobile-first PWA with 60+ components |
| GitHub Workflows | 5 | Deploy, parity audit, secrets guard, dictionary guard |
| SNS Topics | 4 | Alerts, sync triggers, dead-letter |
| EventBridge | 3 rules + 2 pipes | Stream-to-queue + scheduled polling |
| Monthly Cost | ~$30–40 | Fully serverless, no reserved capacity |

## Data Layer: Seven Tables of Truth

All persistent state lives in DynamoDB with PAY_PER_REQUEST billing and deletion protection enabled on every table. The schema design prioritizes agent-friendly access patterns—single-table queries return complete operational context for any project.

| Table | Key Schema | Purpose |
|---|---|---|
| devops-project-tracker | PK: project_id, SK: {type}#{id} | Core tracker for all tasks/issues/features across all projects |
| projects | PK: project_id | Project metadata registry (5 active projects) |
| documents | PK: document_id (DOC-*) | Document metadata (content in S3) |
| coordination-requests | PK: request_id | Agent coordination request lifecycle |
| devops-deployment-manager | PK: project_id, SK: record_id | Deployment specs, requests, state, changelog |
| governance-policies | PK: policy_id | Governance rules + data dictionary + OAuth clients |
| agent-compliance-violations | PK: violation_id | Policy violation audit trail |

DynamoDB Streams on the tracker and documents tables feed EventBridge Pipes that route change events through SQS FIFO queues to the feed publisher Lambda. This event-driven pipeline ensures the PWA's cached JSON feeds stay synchronized within 5 minutes of any mutation—without polling.

# Compute Layer: 19 Lambda Functions

Every Lambda function follows cross-cutting patterns: Cognito JWT validation via a shared layer, module-level singleton DynamoDB clients, structured error envelopes, and CORS headers locked to jreese.net. Service-to-service calls use the X-Coordination-Internal-Key header to bypass JWT validation.

### Core API Functions

| Function | Memory | Timeout | Purpose |
|---|---|---|---|
| tracker_mutation | 256 MB | 10s | Tracker CRUD API for tasks/issues/features |
| coordination_api | 512 MB | 120s | Core coordination platform (~8,000 lines) |
| document_api | 256 MB | 30s | Document CRUD with S3 storage |
| deploy_intake | 512 MB | 120s | Deployment submission and state management |
| project_service | 256 MB | 30s | Project lifecycle management |

| | | | |
|---|---|---|---|
| feed_query | 256 MB | 15s | Feed read API with subscriptions |
| github_integration | 256 MB | 30s | GitHub App issue sync (ENC-FTR-021) |
| reference_search | 256 MB | 10s | S3 reference document search |

### Pipeline Functions

| Function | Trigger | Purpose |
|---|---|---|
| deploy_orchestrator | SQS FIFO (batch 1) | CodeBuild orchestration + non-UI inline deploy |
| deploy_finalize | EventBridge (CodeBuild state) | Post-build: audit records, worklogs, version bump |
| feed_publisher | SQS FIFO (batch 10) | DynamoDB → S3 feeds + CloudFront invalidation |
| governance_audit | DynamoDB Stream (batch 25) | Write-source anomaly detection |
| json_to_parquet | EventBridge rule | Analytics pipeline: JSON → Parquet |
| glue_crawler_launcher | SNS | Glue crawler with 2-hour cadence guard |

### Auth & Edge Functions

| Function | Runtime | Region | Purpose |
|---|---|---|---|
| auth_edge | Node.js 18 | us-east-1 | CloudFront Lambda@Edge JWT validation |
| auth_refresh | Python 3.11 | us-west-2 | Cognito token refresh endpoint |
| bedrock_agent_actions | Python 3.11 | us-west-2 | Bedrock Agent action group executor |
| enceladus-shared (Layer) | Python 3.11 | us-west-2 | JWT, auth, AWS clients, serialization |

# Governance Framework

Governance is not an afterthought in Enceladus—it is the architectural foundation. Every write operation, across every surface (PWA, MCP, coordination dispatch, CLI), flows through the same validation gates. The governance framework operates across four dimensions:

## Write-Source Attribution

All mutations to the tracker table include a write_source attribute identifying the channel: mcp_server (governed MCP tools), tracker_cli (human-supervised CLI), mutation_api (Cognito-

authenticated PWA), or feed_publisher (automated pipeline). The governance_audit Lambda, triggered by DynamoDB Streams, detects MISSING_WRITE_SOURCE, EMPTY_WRITE_SOURCE, and UNKNOWN_CHANNEL anomalies and alerts via SNS.

## Governance Hash (Optimistic Concurrency)

Every MCP write operation requires a governance_hash parameter—the SHA-256 of all loaded governance files. If the hash doesn't match the current live governance state, the write is rejected. This ensures agents operate against consistent governance rules and prevents stale-policy mutations.

## Checkout Service (Task Locking)

The checkout service (ENC-FTR-037) ensures exclusive task ownership. When an agent begins work, it calls **checkout_task** which atomically sets active_agent_session_id and advances the task to in-progress. All subsequent status transitions must use **advance_task_status**—direct tracker_set calls for task status return 403. The lifecycle gate matrix enforces evidence at every transition:

| Transition | Required Evidence | Token Issued |
|---|---|---|
| in-progress | active_agent_session_id | — |
| coding-complete | (none) | CAI (Commit Approval ID) |
| committed | commit_sha (40-char hex, GitHub-validated) | CCI (Commit Complete ID) |
| pr | (none) | — |
| merged-main | pr_id + merged_at (GitHub API validated) | — |
| deploy-success | deploy_evidence (GitHub Actions Jobs API object) | Clears CAI + CCI |
| closed | live_validation_evidence | Releases checkout |

## Governance Data Dictionary

The governance data dictionary (governance_data_dictionary.json) defines authoritative field semantics, allowed enum values, and validation constraints for every governed entity. It is enforced at two levels: (1) a CI guard that fails PRs when schema-affecting code changes without corresponding dictionary updates, and (2) a runtime endpoint that agents query to validate values before submission. The dictionary currently governs tracker.task, tracker.issue, tracker.feature, deploy.request, coordination.request, and 15+ additional entity schemas.

# MCP Server: The Agent Interface

The MCP (Model Context Protocol) server is the canonical interface for all AI agent interactions with Enceladus. Written in async Python (~4,400 lines), it exposes 35+ tools across 8 operational domains via stdio transport. Every tool routes through the platform's HTTP APIs—the MCP-API boundary governance policy ensures no tool handler directly accesses DynamoDB business tables, preventing transport-specific behavior drift.

## Tool Inventory by Domain

| Domain | Tools | Key Capabilities |
|---|---|---|
| Projects | 2 | List all projects, get project details |
| Tracker | 7+ | CRUD for tasks/issues/features, pending updates, acceptance evidence |
| Checkout | 4+ | checkout_task, advance_task_status, release_task, validate CCI |
| Documents | 6 | Create, read, update, search, policy check |
| Governance | 3+ | Hash computation, governance update, data dictionary query |
| Deploy | 8+ | Submit, status, history, state management, trigger, changelog |
| Coordination | 4 | Capabilities, request state, dispatch plan generation |
| GitHub | 3 | Issue creation, Projects v2 board sync, project listing |
| System | 1 | Connection health check |

The MCP server is deployed in two modes: (1) a **local stdio server** for desktop agent sessions (Claude Code, Codex CLI), and (2) a **remote Streamable HTTP Lambda** (ENC-FTR-025) accessible from Claude.ai web/mobile connectors via OAuth 2.1 with PKCE. The remote server issues stateless HMAC-signed refresh tokens with 30-day TTL, enabling persistent sessions without server-side token storage.

> *"The MCP server is the funnel through which all agent intelligence flows into the platform. By enforcing API-boundary governance at this layer, we guarantee that every agent—regardless of provider—operates under identical validation, auth, and audit rules."*

# Event-Driven Pipelines

## Feed Publishing Pipeline

The feed pipeline transforms DynamoDB mutations into cached JSON feeds served through CloudFront. The flow: tracker/document mutations trigger DynamoDB Streams, which are routed through EventBridge Pipes (filtering out reference records) into SQS FIFO queues. The queue uses 5-minute visibility timeout as a natural debounce mechanism, grouping rapid mutations by project_id. The feed_publisher Lambda (batch size 10, 600s timeout) regenerates per-project JSON feeds, uploads to S3, and invalidates CloudFront paths—ensuring the PWA displays fresh data within minutes.

## Deployment Orchestration Pipeline

Deployments flow through a governed pipeline: deploy_intake validates the request (including GitHub PR merge verification), writes to the deployment-manager table, and enqueues to SQS FIFO. The deploy_orchestrator resolves semver, writes a deployment spec, and triggers CodeBuild for UI deployments or executes inline for non-UI Lambda updates. On CodeBuild completion, an EventBridge rule triggers deploy_finalize, which updates the spec, writes changelog audit records, and stamps deployment worklogs on related tracker records.

## Analytics Pipeline

After feed publication, staged JSON exports are written to the analytics S3 bucket. An EventBridge rule triggers the json_to_parquet_transformer, which converts JSON to Snappy-compressed Parquet files using PyArrow. SNS notifications trigger Glue crawlers (with a 2-hour cadence guard) to update the data catalog, enabling Trino/Superset analytics over the complete project history.

# Frontend: Mobile-First PWA

The Enceladus PWA is a React 19 application built with Vite 7.3, TypeScript, and Tailwind CSS. It ships as a Progressive Web App with offline static asset caching, installable on iOS and Android via service worker. The dark-themed UI (slate-900 background) is optimized for mobile-first interaction with no desktop breakpoints—it's designed to be the dashboard you check from your phone.

## Pages & Navigation

| Route | Page | Key Features |
|---|---|---|
| / | Dashboard | Stat cards, top projects overview |
| /projects/:id | Project Detail | Tabs for tasks/issues/features, reference docs |
| /feed | Unified Feed | Live polling (3s), merged task/issue/feature stream |

| | | |
|---|---|---|
| /tasks/:id | Task Detail | Mutations, checkout status, lifecycle visualization |
| /documents | Documents | All 55 knowledge documents with search |
| /coordination | Coordination | Agent request monitoring, dispatch status |

State management combines **TanStack React Query** (2-min staleTime, 30-min gcTime) with React Context for auth state. The API client layer implements 3-cycle retry with automatic credential refresh on 401 responses. Feed data is served from S3 through CloudFront, while mutations route through the API Gateway. 13 custom hooks abstract data fetching, filtering, pagination, and session lifecycle management.

# Feature Portfolio: 37 Features

The Enceladus feature portfolio spans foundational infrastructure, governance enforcement, agent integration, and user experience. Each feature follows the governed ontology: user story, acceptance criteria, and evidence-backed completion handshake.

## Production Features (Deployed & Governed)

| ID | Title | Priority |
|---|---|---|
| ENC-FTR-022 | Native Disciplined Deployment Lifecycle State Governance | P0 |
| ENC-FTR-033 | Universal Cross-Project Changelog with Release Docs | P2 |
| ENC-FTR-035 | Enhanced Lifecycle Stage Gates (deploy-init/deploy-success) | P1 |
| ENC-FTR-037 | Abstracted Checkout Service with Full Lifecycle Gates | P1 |

## Completed Features (20)

| ID | Title | Priority |
|---|---|---|
| ENC-FTR-001 | Launch Mobile Project Status PWA | P2 |
| ENC-FTR-004 | Custom Auth UX with Session Management | P0 |
| ENC-FTR-006 | Agent Document Management (upload/store/retrieve) | P1 |
| ENC-FTR-009 | PWA Parent/Child Task Hierarchy + Rich RelatedItems | P1 |
| ENC-FTR-011 | Product Ontology Governance (Feature/Task/Issue formalization) | P1 |
| ENC-FTR-012 | Ontology Schema Enhancement (user_story, evidence, agent fields) | P1 |
| ENC-FTR-013 | Ontology Enforcement (state machine, completeness scoring) | P2 |
| ENC-FTR-014 | PWA Task Hierarchy UX (separate hierarchical from related) | P0 |
| ENC-FTR-016 | Native Desktop MCP Session Briefing Documentation | P1 |

| | | |
|---|---|---|
| ENC-FTR-020 | Multi-Agent Governance Guardrail Framework | P1 |
| ENC-FTR-021 | Link to GitHub Issues and Projects | P1 |
| ENC-FTR-026 | Governance Data Dictionary (HTTP + DynamoDB + CI Guard) | P1 |
| ENC-FTR-028 | Unified Multi-Runtime Service Authentication | P0 |
| ENC-FTR-029 | Full Archive Storage of EC2 Codex Session Prompts | P2 |
| ENC-FTR-030 | Structured Server-Authoritative Datetime | P1 |

### In-Progress Features (5)

| ID | Title | Priority |
|---|---|---|
| ENC-FTR-005 | Coordination Mode for Agent-Orchestrated Execution | P1 |
| ENC-FTR-015 | Ontology-Driven Agent Manifest Redesign | P1 |
| ENC-FTR-023 | Harden MCP Deploy Observability and Trigger Flow | P1 |
| ENC-FTR-024 | Checkout Integrity & Session Transparency | P1 |
| ENC-FTR-025 | Remote MCP Server: Streamable HTTP for Claude.ai Connector | P1 |

## CI/CD & Deployment

Enceladus operates 5 GitHub Actions workflows providing continuous deployment, parity validation, secrets scanning, and governance enforcement:

| Workflow | Trigger | Purpose |
|---|---|---|
| api-mcp-backend-deploy | Push to coordination_api/** or mcp-server/** | Deploys coordination API + embedded MCP runtime |
| github-integration-deploy | Push to github_integration/** | Deploys GitHub App integration Lambda |
| ui-backend-deploy | Push to frontend/ui/** | Submits deploy request → SQS → CodeBuild pipeline |
| nightly-parity-audit | Daily 09:25 UTC | Validates live Lambda SHA256 matches repo source |
| secrets-guardrail | PR + push + daily | TruffleHog scan of repo history and filesystem |

The nightly parity audit deserves special attention: it computes SHA-256 hashes of every Lambda function's deployed code and compares against repo source, detecting any configuration drift between what's committed and what's running in production. Output artifacts (resource_inventory.json, lambda_parity.json, summary.md) are retained for 14 days.

## Deployment Types Registry

The deployment manager supports 13 deployment types through a unified intake API: github_public_static, github_private_sst, github_public_workers, github_private_workers, lambda_update, lambda_layer, container_template, glue_crawler_update, glue_job_update, eventbridge_rule, s3_asset_sync, cloudfront_config, and step_function_update. Each type routes through the appropriate orchestration path with semver tracking and changelog generation.

# Frequently Asked Questions

1. **Why the name “Enceladus”?**

Enceladus is a moon of Saturn known for its subsurface ocean and geothermal activity—a small body with disproportionate energy and complexity beneath the surface. The platform shares this quality: a modest ~$35/month serverless footprint that powers sophisticated multi-agent coordination, governed lifecycles, and real-time operational intelligence across an entire project portfolio.

2. **What are the core tenets?**

- **Ontological discipline.** System primitives (Features, Tasks, Issues) have formal definitions, not just database schemas. This discipline cascades into cleaner abstractions, more reliable agent interactions, and extensible data structures.
- **Governance as architecture.** Governance is not a layer bolted on top—it's the substrate. Every write path, every status transition, every deployment goes through the same validation gates.
- **Agent-first design.** The MCP server is the primary interface. The PWA is a visibility tool. If an agent can't do it through MCP, we haven't shipped it yet.
- **Evidence over assertion.** Status transitions require proof: commit SHAs validated against GitHub, PR merge timestamps within 60-second tolerance, deployment evidence from GitHub Actions Jobs API. No transition advances on trust alone.
- **Token economy.** Every design decision considers token cost. Prompt caching (90% discount), strategic model selection, batch API integration (50% discount), and minimal-context briefings all reduce the operational cost of intelligence.

3. **How does multi-agent coordination work?**

The coordination API accepts requests specifying outcomes, preferred providers, and related tracker records. The dispatch plan generator applies heuristics to select execution modes (codex_full_auto, claude_headless, claude_agent_sdk, aws_step_function) and distributes work. Each provider's callback is normalized into a canonical schema. Multi-dispatch plans track per-dispatch outcomes independently, with configurable rollback policies (continue, halt_remaining, rollback_completed) governing partial failure behavior.

4. **What prevents agent collisions?**

The checkout service (ENC-FTR-037) implements exclusive task ownership. checkout_task atomically sets active_agent_session_id and transitions the task to in-progress. Only the owning session can advance the task's status. Child tasks can be checked out while a parent is locked—this is expected behavior for coordination-dispatched parallel work. The checkout releases on task closure or explicit release_task calls.

5. **How does the platform enforce deployment integrity?**

Every deployment request requires a merged PR number and merge timestamp. deploy_intake validates both against the GitHub API before accepting the request. The checkout service

issues Commit Approval IDs (CAI) at coding-complete and Commit Complete IDs (CCI) at committed—the CCI must appear in the PR body for the GitHub Actions PR Commit Gate to pass. deploy-success requires structured GitHub Actions Jobs API evidence with 7 validated fields. This chain ensures no code reaches production without a verified path from checkout through merge through deployment.

6. **What is the project portfolio?**

| Project | Prefix | Status | Description |
|---|---|---|---|
| devops | DVP | Active Production | DevOps governance hub (parent) |
| enceladus | ENC | Active Production | Portfolio management platform |
| jreesewebops | JWO | Active Production | Web infrastructure operations |
| harrisonfamily | HFY | Active Production | Harrison genealogy site |
| agentharmony | AGH | Active Production | Governance nucleus |

# Version History & Changelog

Enceladus maintains a governed changelog with semver tracking. Recent releases:

| Version | Type | Date | Summary |
|---|---|---|---|
| v0.17.0 | Minor | 2026-02-26 | Checkout service CCI validation + deploy evidence gates |
| v0.16.0 | Minor | 2026-02-22 | Governance data dictionary + CI guard enforcement |
| v0.15.0 | Minor | 2026-02-19 | Deploy lifecycle stage gates (deploy-init/deploy-success) |
| v0.14.0 | Minor | 2026-02-15 | Remote MCP server OAuth 2.1 with PKCE + refresh tokens |
| v0.9.0 | Minor | 2026-01-14 | Unified service-to-service authentication overhaul |

The platform has processed **682+ closed tasks** and resolved **79 issues** since inception, with a **~95% completion rate** across all tracked work. 55 knowledge documents capture architectural decisions, implementation plans, deployment strategies, and operational procedures.

# Looking Ahead

The Enceladus roadmap converges on three vectors:

- **Multi-agent coordination maturity (ENC-FTR-005).** Full coordination dispatch with deterministic completion contracts across providers, including the Codex CLI host-v2 execution mode (ENC-FTR-034) for self-hosted agent sessions.

- **Ontology-driven agent manifests (ENC-FTR-015).** Specialized agents per record type, with manifest schemas that declare capabilities, required governance context, and expected tool invocation patterns.
- **Remote accessibility expansion (ENC-FTR-025).** The Streamable HTTP MCP server brings Enceladus to Claude.ai web and mobile connectors, enabling portfolio management from any device without local tooling.

Enceladus is not a product for sale—it is an operational philosophy made manifest. It demonstrates that a single operator, armed with the right abstractions and governed AI agents, can build and maintain production systems at a scale and quality level that would traditionally require a team. The platform's power comes not from complexity, but from the **disciplined simplicity** of treating every entity, every transition, and every mutation as an opportunity to enforce quality at the architectural level.

End of Document | Enceladus v0.17.0 | March 2026