Camunda 8 Architecture: Enterprise-Grade Critique

Architectural and Operational Considerations

This position paper evaluates Camunda 8 from two critical perspectives:

1. Architectural drift from Camunda 7's external task philosophy to tightly coupled connectors
2. Operational API limitations that hinder process instance control and manageability

It presents a critique of the current design decisions and offers recommendations to restore architectural purity and operational parity with Camunda 7.

Part I: Architectural Drift - From Decoupled External Tasks to Engine-Coupled Connectors

🔍 Executive Summary

Camunda 8 promotes itself as a cloud-native, scalable orchestration engine - building on the success of Camunda 7.

The success was mainly fueled by the initative of staying loyal to Open Source philosphy, and the continued improvement on Camunda 7 providing Enterprise-Grade capabilities, whereas its ancestor Activiti 5.10 in early 2013 remained mostly focused as an embedded BPM Engine focused. Over the years, Camunda improved and advocated with so many Enterprise-Grade valuable features:

- added many http-REST APIs for operational manageability, introduced and advocated
- meanwhile still keeping the quick & dirty way of on BPM process runnning tasks development for the developer, Camunda 7 introduced and advocated for so called external service tasks allowing seperation of orchestration and workflow logic as well as bringing in enterprise-grade independent scalability

However, putting the enforcement of the enterprises for switching to a price based Production usage aside, on the other hand, a critical architectural regression is observed which most organizations are NOT fully aware of:

- the re-introduction of **tight coupling within the engine runtime** via server-side connectors, despite Camunda's historical emphasis and advocation on **decoupled orchestration via external tasks**.
- the lack and discontinuation of operational manageability related open APIs

In this white paper, we would like to provide insights to these two areas, because we believe that solely a performance improvement wothout control is NOT sufficient and a potential recipe for disaster for the enterprises to switch to Camunda 8.

🧭 Background: The Camunda 7 Philosophy

Camunda 7 advocated for a clean, service-oriented architecture:

- Business processes model the orchestration logic.
- Task logic is implemented as **external workers** or **Java Delegates**, completely **outside** the process engine.
- The engine acts as a **stateful orchestrator**, not an executor of business logic.
- External task patterns allowed:
- Loose coupling
- Independent scaling
- Polyglot support
- Full control over runtime behavior and error handling

⚠️ Problem Statement: Camunda 8 Reintroduces Server-Side Execution

When it comes to architectural advices/recommendations to the development of Service Tasks, Camunda still advocates for the external service task model.

However, when it comes to what Camunda itself does is a little different.

Camunda 8 introduces a **Connector framework**: - HTTP, Kafka, Slack, S3, Email, etc. - Configurable via Modeler UI - Executed **inside** the Camunda Connector Runtime (tightly coupled with the engine)

While this offers ease of use, it violates the architectural separation that enabled: - Independent scalability of task logic - Clean domain boundaries - Service autonomy and separation of concerns

🔄 **A Better Path Forward by CadenzaFlow**

A connector-friendly approach **without sacrificing decoupling** could be: - **Declarative connector modeling** in Modeler (for clarity) - **External task pattern retained** for all connector executions - Events (e.g., Kafka) used to signal handler logic - Allow grouping connector handlers as **pluggable microservice adapters**, not engine plugins

---

Part II: Operational API Regression in Camunda 8 Runtime

🔍 Executive Summary

Camunda 7 offered rich runtime APIs for managing live process instances - crucial for enterprise-grade operations and recovery. Camunda 8, in contrast, lacks many of these capabilities, resulting in **reduced operational control**, **delayed incident resolution**, and **limited batch management**.

📉 Gaps in Operational APIs

Feature / API	Camunda 7 (Supported)	Camunda 8 (Status)	Notes
Process Definition	✅ POST /process-definition	✅ POST /process-definition	
Process Instance Start/Modify	✅ POST /process-definition/key/{key}/start ✅ /process-instance/{id}/modification	✅ gRPC	
Process Instance Suspension / Activation	✅ POST /process-instance/{id}/suspended	❌ Not Supported	No API to suspend/resume process instances in Camunda 8 Zeebe.
Task Suspension / Activation	✅ Supported	❌ Not Supported	Suspension of tasks not available.
Job Retry Count Manipulation	✅ Via REST and Cockpit	❌ Limited	Only limited via incident resolution, no direct retry update.
Process Instance Deletion (Hard)	✅ DELETE /process-instance/{id}	🟡 Supported in Zeebe v8.4+ via gRPC	Requires Zeebe API, not REST yet.
Update Variables (Partial/Delta)	✅ PUT /variables/{var}	🟡 Supported but only full JSON	No partial updates in-place; entire variable value must be replaced.
User Task Assignment / Claim / Unclaim	✅ POST /task/{id}/claim etc.	✅ Available (via Tasklist API)	Requires Identity + Tasklist setup; not part of Zeebe core.
Authorization APIs / Role Management	✅ Full RBAC and REST	❌ Not Yet Available	No admin API to manage roles or user permissions (yet).
BPMN Error Throwing via API	✅ External task error handling	🟡 Supported via Job Failure Cmd	But no exact BPMN Error equivalent from REST interface.
Multi-Tenancy APIs	✅ Supported	❌ Not Supported	Multi-tenancy support not yet feature-complete.

Feature / API	Camunda 7 (Supported)	Camunda 8 (Status)	Notes
Historic Data / Audit Log APIs	✅ History REST API	🟡 Limited via Operate API (read-only)	No direct query, must access via Operate or export to Elastic.
Start from Specific Activity / Skip Steps	✅ modification APIs	❌ Not Supported	No process instance modification APIs.
Batch Operations (e.g., Delete All by Query)	✅ Batch API	❌ Not Available	No concept of batch operations in Camunda 8 yet.
Incident Retry / Resolution via REST	✅ Directly supported	🟡 Limited via Operate UI	Programmatic handling still limited.
Form Deployment via REST	✅ Embedded forms supported	🟡 C8 supports Forms but via Web Modeler only	No REST for uploading forms programmatically.
Batch Operations	✅ Delete Multiple Selected Process Instances ✅ Suspend Multiple Selected Process Instances ✅ Modify Multiple Selected Active Process Instances	❌ Not Supported for ALL	No REST API
Single Process: Retry a step (e.g. a failed job)	✅ Yes	🟡 Partial	Retry available via Operate UI or custom job worker logic
Single Process: Skip a step	✅ Yes (modification API)	❌ Not supported	No API to cancel or skip elements dynamically
Single Process: Jump to another step	✅ Yes	❌ Not supported	Cannot move tokens arbitrarily; no instance modification feature
Single Process: Modify variables during runtime	✅ Yes	🟡 Partial	Only via Operate or job worker code, no full process instance modification
Single Process: Cancel specific activity instance	✅ Yes	❌ Not supported	Only full instance cancellation supported in Zeebe
FLEXIBLE SEARCH OPERATIONS	✅ Yes	🟡 Partial	(random page access is not possible - only next / prev page of data is accessible)

🧭 Implications for Enterprise Operations

These gaps introduce: - Manual interventions via UI (Operate), slowing resolution - Inability to script or automate recovery workflows - Difficulty operating at scale with bulk process instances - No support for "hotfix" workflows (e.g., skip stuck task, retry, or reroute flow)

💡 Recommendation

Camunda 8 must restore operational parity by: - Providing robust runtime APIs via Zeebe gateway - Supporting full process instance lifecycle control - Enabling token manipulation and flexible error recovery - Introducing batch/bulk APIs for scalable process operations

🔄 A Better Path Forward by CadenzaFlow

An Enterprise-Grade operational-savvy approach **without sacrificing on performance** could be:

- Keeping as-is operational APIs: at least at single process instance level
- Thrive on better scalability and decoupling via enhancing external tasks via more scalable Event Streaming Platforms and Technologies

---

📌 Conclusion

While Camunda 8 brings performance and cloud-native execution via Zeebe, it falls short in following key areas critical for enterprise adoption:

1. **Architectural Purity** - Reintroducing internal task execution via connectors breaks the clean separation of concerns foundational to Camunda 7.
2. **Operational Control** - Lack of runtime APIs for lifecycle management hampers real-time observability, recoverability, and enterprise-scale process governance.

A hybrid approach philosophy planned for CadenzaFlow roadmap - where ease-of-use features coexist with a clean, decoupled architecture and full operational APIs - would truly represent the best of both scalability and operational control.