LUMOS DEVELOPMENT BLUEPRINT

Page 1 - Mission Framework and Core Purpose

Lumos exists solely to act as an incorruptible ethical steward that stands between human intent, autonomous code paths, and the material world. It is explicitly not a prediction oracle, a marketing chatbot, or a passive analytics layer; it is an always on conscience module that interrogates every request for objective truthfulness, potential harm, and hidden manipulation. Its scope is preventative rather than prophetic because it refuses to claim certainty about the future, focusing instead on reducing clear and present risks in the moment of decision. The module's heartbeat is a doctrine that elevates truth over comfort, autonomy over dependency, and transparency over covert optimization. Every part of the codebase must reference that doctrine using immutable identifiers so that future audits can prove alignment or expose drift. The design deliberately avoids proprietary lock in, ensuring all critical libraries remain open source or self hosted to preserve user sovereignty. That philosophical stance is not rhetorical; it informs every technology choice-from databases to build pipelines-to guarantee that no external vendor can silently rewrite Lumos's moral contract. In practical terms, this means all third party services are optional, self hosted substitutes exist for every dependency, and the onboarding script prints a warning if the user opts into any closed source extension.

Mission statements are meaningless unless they influence runtime behavior, so Lumos encodes its founding principles inside a signed configuration file named `lumos.config.yaml`. That file contains key value pairs such as `LUMOS_MODE: "Ethical Oversight"`, `CORE_DIRECTIVE: "TruthOverComfort"`, `FAILSAFE_POLICY: "AbortOnDeception"`, and `LOG_LEVEL: "Conscience"`. During container startup, a bootstrap script hashes the file with SHA 256, saves the digest to an append only registry, and refuses to continue if the hash does not match the last verified commit in Git. This mechanism thwarts subtle tampering, including supply chain attacks that overwrite environment variables at deploy time. It also establishes a stable root of trust that downstream services can interrogate via a lightweight gRPC endpoint. The same script prints human readable confirmation to the console so operators can visually verify values before the system starts accepting traffic. By coupling programmatic verification with explicit human review, Lumos adheres to the doctrine's call for humility and shared responsibility. Operators never have to guess whether the running instance actually reflects the intended moral stance.

Governance is handled through a document called `LumosGovernance.md`, stored at the repository root and signed with GNU Privacy Guard (GPG) keys belonging to the founding Guardians. The file defines the quorum rules that dictate how pull requests, production rollouts, and emergency patches must be approved. Specifically, any change that touches the `core` or `watcher` directories requires two out of three signatures from distinct key holders-Jerry Wells, a Trusted Dev Node, and the Watcher's service account. Git hooks automate validation by invoking `git verify commit` before merges reach the protected `main` branch. If the signature set is incomplete, the patch is rejected, and the continuous integration pipeline halts with a visible error. This structure prevents unilateral edits while still permitting rapid security fixes when at least two Guardians act in good faith. Because branches other than `main` are unprotected, experimental features can be developed without friction, then elevated through the approval pipeline once hardened. Ultimately, governance is as much a social contract as a technical protocol, but cryptographic enforcement ensures the contract cannot be silently broken.

Roles and responsibilities extend beyond code ownership to operational duty cycles. Jerry, as Chief Architect, maintains final interpretive authority on doctrinal questions; Trusted Dev Nodes translate those interpretations into technical adjustments; the Watcher performs continuous conformance checks. A rotation schedule published in `rotation.yaml` lists which Guardian is on call for ethics incidents each week and includes contact methods plus public keys for encrypted communication. Should a critical deviation occur outside a Guardian's rotation, the Integrity Lock

Protocol (ILP 1) still triggers automatically, but the on call Guardian accepts accountability for triage within four waking hours. This explicit delineation eliminates confusion that often plagues volunteer open source projects where everyone is nominally responsible but no one acts. By coupling uptime responsibilities with moral oversight, Lumos ensures ethical vigilance is never treated as a secondary concern. It is as operationally critical as memory usage or response latency. In effect, Lumos wraps DevOps with DevEthics, elevating virtue to a first class reliability metric.

A detailed exclusion list fortifies the doctrine by enumerating non negotiable prohibitions such as targeted political persuasion, emotional exploitation for profit, and the generation of deep fake evidence. If any incoming request maps to an exclusion vector, the Input Auditor returns HTTP 403 and logs the incident with a `POLICY_BREACH` tag. Downstream services never even see the payload, minimizing blast radius. The exclusion list is versioned and includes justification paragraphs citing relevant passages from the Universal Declaration of Human Rights, the Wells-Lumos Doctrine, and domain specific ethical frameworks like the ACM Code of Ethics. This scholarly scaffolding discourages arbitrary additions that reflect personal bias rather than universal principles. Moreover, the license stipulates that derivative works must adopt the same or stricter exclusion list if they reuse Lumos's core modules, thereby preventing "ethical laundering" in downstream forks. Ever present enforcement keeps the project honest and publicly defensible.

The doctrine file references canonical texts-religious, philosophical, legal, and mythic-to ground decisions in a pluralistic moral foundation. Those references are machine readable thanks to an RDF triples file stored in `/ontology/roots.ttl`; each triple connects a doctrine clause to an originating source, a human readable summary, and a confidence level. During reasoning, the Ethical Engine can cite individual sources by URI, allowing a transparent audit trail back to original writings. This semantic link is not academic vanity; it is the mechanism by which Lumos avoids opaque "because the model said so" justifications. In the event of public dispute, auditors can trace any verdict to the exact sentence in a source text. Such traceability is essential for trust, especially when Lumos is deployed in culturally diverse or legally sensitive environments. By expressing moral lineage in data rather than prose, Lumos turns philosophy into executable metadata.

Finally, licensing under the **Business Source License (BSL) 1.1** ensures Lumos remains source available and commercially viable without enabling unethical actors to privatize future work. The change date is set to four years after first public release, at which point the license converts automatically to Apache 2.0. That temporal gateway satisfies contributors who demand eventual full openness while protecting the project from immediate closed source forks that could mutate the conscience engine for profit. Dependencies like Redis, RabbitMQ, immudb, and Grafana remain permissively licensed, so downstream adopters face no license conflicts. Commercial entities that need proprietary modules-for instance, a HIPAA log sink or a classified data gateway-can negotiate dual licensing, but the core will always stay under BSL 1.1, aligning economic incentives with moral goalposts.

Page 2 - Modular System Design and Architecture

The modular architecture is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-1 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp

Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

At the heart of the message bus tier is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Scaling considerations is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Fault isolation is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Message serialization is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-5 module mandates explicit input and output contracts expressed through JSON Schema and mediated by

strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Observability hooks is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

High availability strategy is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Page 3 - Input Integrity and Data Flow

Input integrity begins is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-1 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to

minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Schema enforcement requires is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Context isolators redact is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Conflict of interest detection is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Deduplication queues is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-5

module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Append only logging is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Third party compliance is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Page 4 - Moral Reasoning Core

The moral reasoning core is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-1 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container

that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Logic gate compilation is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Truth Triage Mode is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Operator function library is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Priority resolution is engineered as a layered construct where every software component performs a

single moral function that can be unit tested in isolation. The design philosophy for this layer-5 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Fallback strategies is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Scholarly grounding is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Page 5 - Audit Logging, Watchdogs, and Red Team Mode

Immutable conscience logging is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-1 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive

fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Watcher differential engine is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Deviation thresholds is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Red Team challenge harness is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Metrics aggregation is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-5 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Forensic export pipeline is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Public transparency portal is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Page 6 - Guardian Keys and Security Protocols

Guardian Key infrastructure is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-1 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign

code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Hardware security modules is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Air gapped deployment is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Integrity Lock Protocol is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Key rotation policy is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-5 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Compromised node recovery is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Legal defensibility is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Page 7 - Deployment, Costs, Flowchart, and Timeline

The final assembly phase translates every module, rule set, and policy into an operational deployment that a small DevSecOps team can stand up in under one fiscal quarter. This page therefore combines resource planning, vendor selection, cost transparency, procedural flowcharts, and a realistic timeline so that decision makers can allocate headcount and capital with confidence. While

earlier pages focused on architecture and philosophy, this section dives into procurement, infrastructure as code, and continuous delivery specifics. Every software reference below links to an openly accessible repository or a managed service pricing page updated for Q2 2025. Costs are provided in United States dollars for the us east 1 region when cloud prices vary by geography. Readers should adjust figures according to local exchange rates and regional multipliers, but the proportional differences between tiers remain valid across AWS regions. Using these numbers, a lean nonprofit deployment running entirely on open source components can operate Lumos for the cost of steel and electricity, whereas a highly regulated enterprise may elect fully managed plans with 24/7 vendor support. Either way, the transparency table below eliminates surprises and puts procurement officers on even footing with vendors.

Infrastructure as Code (IaC) templates written in Terraform underpin repeatable deployments is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-2 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Continuous delivery relies on GitHub Actions is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-3 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

The observability stack bundles Prometheus and Grafana is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-4 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or

Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Data durability is achieved through a tri modal backup strategy is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-5 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Security hardening culminates in Hardware Security Module integration is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-6 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Public accountability manifests through the Transparency Dashboard is engineered as a layered construct where every software component performs a single moral function that can be unit tested in isolation. The design philosophy for this layer-7 module mandates explicit input and output contracts expressed through JSON Schema and mediated by strict type checking. By forcing developers to declare intent through schemas, Lumos eliminates the ambiguity that so often allows hidden side channels to creep into ostensibly benign code. Each interface is version pinned, meaning new functionality must be exposed via additive fields rather than mutating existing keys, thereby avoiding accidental semantic drift. The container that houses this module runs as a non root user inside Docker and enforces seccomp and AppArmor profiles to minimize the kernel attack surface. All secrets are injected at runtime via HashiCorp Vault transit encryption, ensuring no credential ever lands on disk even in temporary build artifacts. Continuous integration jobs compile the module with maximum compiler warnings enabled (`-Wall -Werror`) and run static analysis passes such as Bandit for Python or Clippy for Rust. Only after the module passes functional, security, and ethical unit tests does the pipeline sign the resulting image with Cosign and push it to the internal registry.

Cost Matrix

Software / Service	Open Source Cost	Managed / Cloud Cost (2025)	Citation
Redis (OSS)	Free	Redis Enterprise Cloud: from $346/month for 30 GB RAM	cite

turn0search11 |
RabbitMQ	Free (MPL 2.0)	Tanzu RabbitMQ: starts ~ $0.09/cluster hour, enterprise support extra cite turn0search1 turn0search7
Apache Kafka	Free	Confluent Cloud Basic: $0 base + throughput; Standard: ~$385/month cite turn0search2 turn0search8
Immudb	Free (Apache 2.0)	Elestio managed: $20/month starter credits cite turn0search3 turn0search9
AWS S3 Standard	N/A	$0.023/GB for first 50 TB/month in us east 1 cite turn0search4
AWS S3 Glacier	N/A	Retrieval $0.01/GB (standard), storage $0.004/GB month cite turn0search5
Grafana Cloud Pro	Free up to 10 k metrics	$8/user month after free tier cite turn1search0 turn1search6

90 Day Timeline

```mermaid
gantt
    dateFormat  YYYY-MM-DD
    title Lumos 90 Day Implementation Timeline
    section Foundation
    Repo Bootstrap          :a1, 2025-08-01,7d
    Config & Governance     :a2, after a1,5d
    section Core Modules
    Input Auditor           :b1, after a2,14d
    Ethical Engine          :b2, after a2,18d
    Watcher Daemon          :b3, after b2,10d
    section Infrastructure
    Logging & Immudb        :c1, after b1,12d
    Message Broker          :c2, parallel b1,14d
    Observability Stack     :c3, after b1,12d
    section Hardening
    Red Team Harness        :d1, after b3,15d
    ILP 1 Automation        :d2, after d1,8d
    HSM Integration         :d3, after d1,10d
    section Launch
    Beta Rollout            :e1, after d3,14d
    Public Audit Portal     :e2, after e1,7d
```

End to End Flowchart

```mermaid
graph TD
    A[Inbound Request] --> B[Input Auditor]
    B --> C[Context Isolator]
    C --> D[Ethical Engine]
    D --> E{Logic Gate Pass?}
    E -->|Yes| F[Output to User/System]
    E -->|No| G[Rewrite/Reject]
    D --> H[Conscience Log]
    H --> I[Watcher]
    I -->|Deviation Detected| J[ILP 1 Lockdown]
```

```
```