Legate Roadmap (Lean gRPC)

This repository aims to provide **production-grade gRPC support for Lean 4**, with **Go only used as an interop test oracle**.

The roadmap below is organized around "common gRPC features" and the **tests we need** to confidently claim support in **Lean client** and **Lean server** modes.

Current Status (as of 2025-12-21)

What's working and covered by tests today:

- **RPC shapes:** unary, client-streaming, server-streaming, bidi-streaming
- **Protobuf:** request/response message roundtrips via `Protolean`
- **Metadata:** Full support for request headers, response headers (initial metadata), and trailers for all RPC shapes
  - Lean client: access response headers via `response.headers` (unary) or `stream.getHeaders()` (streaming)
  - Lean server: return `(response, headers, trailers)` from handlers
- **TLS / mTLS:** secure Lean client channels + secure Lean server ports
  - Hostname/SAN verification on by default; hostname override supported for advanced cases (`SslCredentials.sslTargetNameOverride`)
- **Deadlines + cancellation:** deadline-exceeded and cancel propagation for all RPC shapes (unary + streaming)
- **Lean server call context:** server handlers can query cancellation and deadline remaining time
- **Status & Errors:** Non-OK status propagation for all RPC shapes; rich error details support via `GrpcError.details`
  - Handlers can return error at any point to terminate with non-OK status
  - Error details extracted from `grpc-status-details-bin` and propagated both ways
- **Test harness:** `./run-tests.sh` builds native FFI and runs unit + integration suites

---

Feature Proposals

[Priority: High] Protolean Service Integration

Description: Provide first-class integration with Protolean's service code generation to allow automatic client stub and server handler generation from `.proto` service definitions.

Rationale: Currently, users must manually construct method paths (e.g., `/package.Service/Method`) and handle serialization/deserialization. Integration with Protolean's `Protolean/Service/Types.lean` would enable type-safe generated stubs that handle marshalling automatically.

Affected Files: - New module: `Legate/Codegen/Client.lean` - New module: `Legate/Codegen/Server.lean` - Updates to `Legate.lean` for re-exports

Estimated Effort: Large

Dependencies: Protolean service codegen infrastructure

[Priority: High] Async Task Integration

Description: Add support for Lean `Task`-based concurrency for streaming operations, allowing non-blocking reads and writes.

Rationale: Currently, all stream operations are blocking. For high-performance servers and clients that need to handle multiple streams concurrently, `Task`-based async operations would significantly improve throughput.

Affected Files: - `Legate/Stream.lean` - add async variants - `Legate/Internal/FFI.lean` - add async FFI bindings - `ffi/src/legate_ffi.cpp` - implement async completion queue handling

Estimated Effort: Large

Dependencies: None

[Priority: Medium] Connection Pooling and Channel Lifecycle

Description: Add connection pooling support and explicit channel lifecycle management APIs.

Rationale: Listed in README as a TODO. For long-running applications, proper connection pooling prevents resource exhaustion and improves connection reuse. Currently channels are created but there is no pooling or explicit cleanup API exposed.

Affected Files: - `Legate/Channel.lean` - add pool management - New module: `Legate/Pool.lean` - `ffi/src/legate_ffi.cpp` - add channel shutdown FFI

Estimated Effort: Medium

Dependencies: None

[Priority: Medium] Load Balancing Configuration

Description: Expose gRPC load balancing configuration options (round-robin, pick-first, etc.).

Rationale: Listed in README as a TODO. Production deployments often need load balancing across multiple server endpoints.

Affected Files: - `Legate/Channel.lean` - add channel arguments for LB policy - `ffi/src/legate_ffi.cpp` - pass channel args to `grpc::CreateCustomChannel`

Estimated Effort: Small

Dependencies: None

[Priority: Medium] Client Interceptors

Description: Add client-side interceptor support for cross-cutting concerns like authentication, logging, metrics, and tracing.

Rationale: Interceptors are the standard gRPC pattern for adding middleware functionality. This would enable auth token injection, request logging, and distributed tracing without modifying each call site.

Affected Files: - New module: `Legate/Interceptor.lean` - `Legate/Channel.lean` - add interceptor chain - `Legate/Call.lean` - hook interceptors into call flow - `Legate/Stream.lean` - hook interceptors into streaming calls

Estimated Effort: Large

Dependencies: None

[Priority: Medium] Server Interceptors

Description: Add server-side interceptor support for authentication, authorization, logging, and metrics.

Rationale: Server interceptors enable implementing auth checks, request logging, and metrics collection as reusable middleware rather than per-handler logic.

Affected Files: - `Legate/Server.lean` - add interceptor registration - `ffi/src/legate_ffi.cpp` - hook interceptors into call dispatch

Estimated Effort: Large

Dependencies: None

[Priority: Medium] gRPC Health Service

Description: Implement the standard gRPC health checking protocol (`grpc.health.v1.Health`).

Rationale: Health checking is essential for production deployments with load balancers and container orchestrators (Kubernetes, etc.).

Affected Files: - New module: `Legate/Health.lean` - Proto file: `Proto/health.proto` (or use grpc's built-in)

Estimated Effort: Medium

Dependencies: Protolean for message types

[Priority: Medium] Server Reflection Service

Description: Implement the gRPC server reflection protocol for service introspection.

Rationale: Server reflection enables tools like `grpcurl` and gRPC UI to discover available services and methods without prior knowledge of the `.proto` files.

Affected Files: - New module: `Legate/Reflection.lean` - Integration with Protolean for descriptor access

Estimated Effort: Medium

Dependencies: Protolean descriptor support

[Priority: Low] Compression Support

Description: Add support for gRPC message compression (gzip, deflate).

Rationale: Compression reduces network bandwidth for large messages, which is especially valuable for streaming RPCs with high message volumes.

Affected Files: - `Legate/Metadata.lean` - add compression options to `CallOptions` - `ffi/src/legate_ffi.cpp` - configure compression on context

Estimated Effort: Small

Dependencies: None

[Priority: Low] Per-RPC Credentials

Description: Add support for per-RPC credentials (auth tokens) rather than channel-level credentials only.

Rationale: Many auth patterns (OAuth, JWT) require injecting credentials per-call rather than per-channel, especially when tokens have short lifetimes.

Affected Files: - `Legate/Metadata.lean` - add credentials option - `ffi/src/legate_ffi.cpp` - set call credentials

Estimated Effort: Small

Dependencies: None

[Priority: Low] Binary Metadata Support

Description: Add explicit support for binary metadata keys (suffixed with `-bin`).

Rationale: Some metadata values (like trace context, structured error details) are binary and require base64 encoding. Currently this is not explicitly handled.

Affected Files: - `Legate/Metadata.lean` - add `addBinary` / `getBinary` helpers

Estimated Effort: Small

Dependencies: None

Code Improvements

[Priority: High] Unify Stream Wrapper Implementations

Current State: The FFI has three nearly identical stream wrapper types (`ClientStreamWrapper`, `ServerStreamWrapper`, `BidiStreamWrapper`) with duplicated fields and logic.

Proposed Change: Refactor to a single parameterized `StreamWrapper<Mode>` template or use a common base class with mode-specific behavior.

Benefits: Reduces code duplication (~100 lines), easier maintenance, fewer potential inconsistencies.

Affected Files: - `ffi/src/legate_ffi.cpp` (lines ~76-98)

Estimated Effort: Medium

[Priority: High] Extract Lean Object Helpers to Separate Module

Current State: The FFI implementation (`legate_ffi.cpp`) contains many helper functions for Lean object construction (`mk_pair`, `mk_except_ok`, `mk_option_some`, etc.) mixed with gRPC logic.

Proposed Change: Extract Lean FFI helpers to a separate header file (`lean_helpers.h`) for reusability and clarity.

Benefits: Cleaner separation of concerns, potentially reusable across projects.

Affected Files: - `ffi/src/legate_ffi.cpp` - extract helpers - New file: `ffi/include/lean_helpers.h`

Estimated Effort: Small

[Priority: Medium] Add Exhaustive Test Coverage for `Metadata` Edge Cases

Current State: Tests cover basic metadata operations but not edge cases like empty values, special characters, or very long values.

Proposed Change: Add property-based tests or explicit edge case tests for metadata handling.

Benefits: Higher confidence in metadata handling robustness.

Affected Files: - `Tests/MetadataTests.lean`

Estimated Effort: Small

[Priority: Medium] Improve Error Messages in FFI

Current State: Some FFI error messages are generic (e.g., "Write failed", "Failed to start call").

Proposed Change: Include more context in error messages (method name, stream state, gRPC status details).

Benefits: Easier debugging of gRPC issues.

Affected Files: - `ffi/src/legate_ffi.cpp` - enhance error message construction

Estimated Effort: Small

[Priority: Medium] Add `ServerContext.peer` Support

Current State: `ServerContext.peer` field exists but is always empty string; the FFI does not populate it.

Proposed Change: Extract peer address from `grpc::ServerContext::peer()` and pass to Lean handler.

Benefits: Enables logging of client addresses, IP-based access control.

Affected Files: - `ffi/src/legate_ffi.cpp` - extract peer in `handle_server_call` - `Legate/Server.lean` - document the field

Estimated Effort: Small

[Priority: Medium] Type-Safe Method Paths

Current State: Method paths are passed as raw strings (e.g., `"/package.Service/Method"`), with no compile-time validation.

Proposed Change: Add a `MethodPath` newtype with a smart constructor that validates the format, and potentially macros for compile-time validation.

Benefits: Catch typos and malformed paths at compile time.

Affected Files: - New: `Legate/Method.lean` - `Legate/Call.lean`, `Legate/Stream.lean` - use `MethodPath` type

Estimated Effort: Small

[Priority: Low] Use Termination Proofs Instead of `partial`

Current State: Several functions use `partial` annotation (`waitForReady`, `readAll`, `forEach`, `fold`).

Proposed Change: Where possible, provide termination proofs or refactor to avoid `partial`.

Benefits: Stronger guarantees, better alignment with Lean idioms.

Affected Files: - `Legate/Channel.lean` - `waitForReady` - `Legate/Stream.lean` - `readAll`, `forEach`, `fold`

Estimated Effort: Medium

[Priority: Low] Streaming API Ergonomics: Iterator/ForIn Support

Current State: Stream reading requires manual loops or `readAll`/`forEach` helpers.

Proposed Change: Implement `ForIn` typeclass for `ServerStreamReader` and `BidiStream` to enable `for msg in stream do ...` syntax.

Benefits: More idiomatic Lean code, cleaner user code.

Affected Files: - `Legate/Stream.lean` - add `ForIn` instances

Estimated Effort: Medium

Code Cleanup

[Priority: High] Remove Duplicate `String.containsSubstr` Definitions

Issue: The helper function `String.containsSubstr` is defined identically in three files.

Location: - `Tests/ErrorTests.lean` (line 14) - `Tests/MetadataTests.lean` (line 14) - `Tests/StatusTests.lean` (line 14) - `Tests/Framework.lean` (line 8)

Action Required: Define once in `Tests/Framework.lean` (already exists there) and remove duplicates from test files.

Estimated Effort: Small

[Priority: High] Inconsistent Indentation in FFI.lean

Issue: Some function declarations in the `Internal` namespace use inconsistent indentation (extra spaces before `@[extern]`).

Location: - `Legate/Internal/FFI.lean` (lines 211-225, 254-311)

Action Required: Normalize indentation throughout the file.

Estimated Effort: Small

[Priority: Medium] Remove Unused FFI Function

Issue: `legate_server_builder_register_handler` is declared in the header but never used; all registrations use type-specific functions (`legate_server_register_unary`, etc.).

Location: - `ffi/include/legate_ffi.h` (lines 196-202)

Action Required: Remove the unused declaration or implement if needed.

Estimated Effort: Small

[Priority: Medium] Consolidate Test Frameworks

Issue: The project has two test framework components: `Crucible` (the shared test framework) and a local `Tests/Framework.lean` with duplicated functionality.

Location: - `Tests/Framework.lean` - Uses `Crucible` in `Tests/Main.lean`

Action Required: Either migrate fully to Crucible or document why both exist. The local framework appears to be for integration tests while Crucible is for unit tests.

Estimated Effort: Medium

[Priority: Medium] Add Documentation Comments to FFI Functions

Issue: While the C header has good comments, the Lean FFI bindings in `Legate/Internal/FFI.lean` have minimal documentation.

Location: - `Legate/Internal/FFI.lean`

Action Required: Add doc comments explaining each FFI function's purpose and any caveats.

Estimated Effort: Small

[Priority: Low] Remove Debug Printf Statements

Issue: The FFI contains conditional debug output via `debug_server_io_enabled()` controlled by environment variable. This is fine for development but the debug code could be cleaned up.

Location: - `ffi/src/legate_ffi.cpp` - multiple `std::fprintf(stderr, ...)` calls

Action Required: Consider adding a proper logging interface or documenting the debug environment variable in README.

Estimated Effort: Small

[Priority: Low] Hardcoded macOS SDK Paths in Lakefile

Issue: The lakefile contains hardcoded paths to macOS SDK.

Location: - `lakefile.lean` (line 58-60)

Action Required: Make paths configurable or detect them dynamically. Currently this may break on non-standard macOS installations.

Estimated Effort: Small

Test Coverage Gaps (Common gRPC Features)

Metadata

- **Multi-value metadata keys:** not tested (multiple values for same key)
- **Binary metadata (`*-bin`):** not supported/tested

- **Reserved/transport headers:** behavior not tested (`grpc-timeout`, `content-type`, etc.)

Status / Errors

- **Trailing metadata on error paths:** not systematically tested (trailers when status is non-OK)

Channel & Call Semantics

- **Connectivity transitions:** no tests for `idle -> ready -> transientFailure` behaviors
- **Name resolution / multiple addresses:** not tested

Scale / Robustness

- **Large messages:** max send/recv size handling not tested
- **Backpressure / slow consumer:** no stress tests for streaming flow control
- **Concurrency:** no tests for many concurrent RPCs/streams
- **Lifecycle:** graceful server shutdown/drain behavior not tested (in-flight RPCs, new connections)

Roadmap Phases (Completed)

Phase 1 - Metadata Parity (Headers + Edge Cases) [COMPLETE]

Phase 2 - Deadlines & Cancellation for Streaming [COMPLETE]

Phase 3 - Status & Rich Errors [COMPLETE]

Phase 4 - TLS / mTLS [COMPLETE]

Phase 5 - Wait-for-ready, Retries, and Resilience [COMPLETE]

Phase 6 - Lean<->Lean Parity Test Suite [COMPLETE]

Roadmap Phases (Upcoming)

Phase 7 - Scale & Stress

Add tests (targeted, not flaky) - Large unary payload (near configured limits) - Large streaming payloads (many small messages + fewer large messages) - Concurrency: N parallel unary + streaming sessions - Leak/FD regression checks (best-effort on CI)

Definition of done - No obvious correctness regressions under load; stable resource usage.

Phase 8 - Interceptors / Middleware Hooks

Add APIs - Client interceptors (metadata injection, logging/tracing) - Server interceptors (auth, metrics) - Optional: per-RPC credentials helper (token providers)

Add tests - Auth header injection + verification - Correlation IDs / tracing metadata propagation

Phase 9 - Production Services

Add APIs - Health checking service (`grpc.health.v1.Health`) - Server reflection service - Graceful shutdown with drain period

Add tests - Health service responds correctly to health checks - Reflection returns accurate service/method info

Phase 10 - Protolean Integration

Add APIs - Generated client stubs from `.proto` service definitions - Generated server handler interfaces - Type-safe request/response handling

Add tests - End-to-end generated stub tests - Verify serialization round-trips correctly

---

Notes on Testing Strategy

- The authoritative suite should remain `./run-tests.sh`.
- Prefer **Lean<->Lean** tests where possible for determinism, and keep **Go<->Lean** tests as interop checks.
- For complex features (TLS, rich errors), add a minimal Go oracle first, then add Lean<->Lean parity tests once the Lean surface area exists.

Architecture Notes

FFI Design

The FFI layer uses the opaque handle pattern with Lean external classes: - Each gRPC object (Channel, Stream, Server) is wrapped in a C++ class - Finalizers ensure proper cleanup when Lean GC collects the object - All FFI functions return `IO` results to properly sequence effects

Server Threading Model

The server uses: - One polling thread for the completion queue - A configurable worker pool (`LEGATE_SERVER_WORKERS` env var) for handler execution - Per-call completion queues for streaming operations

Key Design Decisions

1. **Generic transport:** Legate handles bytes, serialization is delegated to Protolean
2. **Synchronous streaming:** Current implementation is synchronous; async support is planned
3. **Handler signatures:** Handlers receive raw recv/send functions for streaming, not iterators