# gRPC for Node.js

Deep dive by Michael Lumish - mlumish@

# What's gRPC

- RPC system working on top of HTTP2
  - Good nat traversal, stream multiplexing within single connection
- Implementations in many languages with strong backward compatibility
- Provides optional streaming calls
  - client-streaming, server-streaming, or both
- Provides authentication systems
  - oauth, jwt, custom protocols
- Provides strong protocol typing through protobuf
- Provides many rich features
  - Load balancing, retries, flow control, cancellation, deadlines, etc...

# Development workflow

Write .proto files, then write client or server handlers (or stubs)

Two supported workflows:

- Compile-time protobuf code generation using protoc
- Run-time protobuf parsing using third-party protobuf.js

# Demo basic client and server

# Original Library: Architecture

- C Core
  - C/C++, Directly uses TCP, includes HTTP/2 implementation, interfaces with libuv event loop
- Native Addon
  - Transforms types and control flow between C and JS, callback-based interface
- JavaScript Surface
  - Implements public API using Native Addon API

# Original Library: Precompiled Binaries

- Uses node-pre-gyp
  - Native addon distribution library, downloads and loads appropriate binary
  - Falls back to compilation using node-gyp
- Distributed on Google Cloud Storage
  - Single bucket, keyed by gRPC version
- Many different factors that affect binaries
  - Node vs Electron, operating systems, architecture
- Generated using custom scripts and internal CI infrastructure

# Original Library: Issues

- Installation
  - Download failures, filesystem access failures, missing version support, fallback failure
- Loading
  - Mismatch between installing and loading the library, especially when deploying code
- Debugging
  - Native addons are opaque to Node developers trying to debug code
- Combinatorial platform and runtime support
  - 11 Node versions and 10 Electron versions, 4 CPU architectures, 3 operating systems, 2 libc variants

# Pure JavaScript Node gRPC (grpc-js)

- Actually written in TypeScript
- Uses Node's built in http2 module
  - http2 module: added in Node 9, backported to Node 8
- Currently implemented: client with basic features:
  - Unary and streaming calls
  - Cancellation, deadlines, and automatic reconnection

# grpc-js Design Considerations and Priorities

- API compatibility
  - APIs in grpc-js exactly match corresponding APIs in the original library
  - Some APIs from the original library are omitted from the grpc-js API
- Prioritizing client and basic features
  - Demand for this re-implementation comes primarily from API client libraries that depend on gRPC
- Some advanced features omitted until we know of demand for them
  - advanced client-side load balancing, whole stream compression, others

# Demo API and Protocol Compatibility

# Development Plans for grpc-js

- Server
- Simple client-side load balancing
- TypeScript generation for generated code

# Questions?

- https://www.grpc.io
- https://github.com/grpc/grpc-node