What is gRPC?
gRPC is a modern, open source remote procedure call (RPC) framework that can run anywhere. It enables client and server applications to communicate transparently, and makes it easier to build connected systems.

Read the longer Motivation & Design Principles post for background on why we created gRPC.

What does gRPC stand for?
gRPC Remote Procedure Calls, of course!

Why would I want to use gRPC?
The main usage scenarios:

Low latency, highly scalable, distributed systems.
Developing mobile clients which are communicating to a cloud server.
Designing a new protocol that needs to be accurate, efficient and language independent.
Layered design to enable extension eg. authentication, load balancing, logging and monitoring etc.
Who's using this and why?
gRPC is a Cloud Native Computing Foundation (CNCF) project.

Google has been using a lot of the underlying technologies and concepts in gRPC for a long time. The current implementation is being used in several of Google's cloud products and Google externally facing APIs. It is also being used by Square, Netflix, CoreOS, Docker, CockroachDB, Cisco, Juniper Networks and many other organizations and individuals.

Which programming languages are supported?
See Officially supported languages and platforms.

How do I get started using gRPC?
You can start with installation of gRPC by following instructions here. Or head over to the gRPC GitHub org page, pick the runtime or language you are interested in, and follow the README instructions.

Which license is gRPC under?
All implementations are licensed under Apache 2.0.

How can I contribute?
Contributors are highly welcome and the repositories are hosted on GitHub. We look forward to community feedback, additions and bugs. Both individual contributors and corporate contributors need to sign our CLA. If you have ideas for a project around gRPC, read guidelines and submit here. We have a growing list of projects under the gRPC Ecosystem organization on GitHub.

Where is the documentation?
Check out the documentation right here on grpc.io.

What is the road map?
The gRPC project has an RFC process, through which new features are designed and approved for implementation. They are tracked in this repository.

How long are gRPC releases supported for?
The gRPC project does not do LTS releases. Given the rolling release model above, we support the current, latest release and the release prior to that. Support here means bug fixes and security fixes.

What is the gRPC versioning policy?
See the gRPC versioning policy here.

What is the latest gRPC Version?
The latest release tag is v1.56.0.

When do gRPC releases happen?
The gRPC project works in a model where the tip of the master branch is stable at all times. The project (across the various runtimes) targets to ship checkpoint releases every 6 weeks on a best effort basis. See the release schedule here.

How can I report a security vulnerability in gRPC?
To report a security vulnerability in gRPC, please follow the process documented here.

Can I use it in the browser?
The gRPC-Web project is Generally Available.

Can I use gRPC with my favorite data format (JSON, Protobuf, Thrift, XML) ?
Yes. gRPC is designed to be extensible to support multiple content types. The initial release contains support for Protobuf and with external support for other content types such as FlatBuffers and Thrift, at varying levels of maturity.

Can I use gRPC in a service mesh?
Yes. gRPC applications can be deployed in a service mesh like any other application. gRPC also supports xDS APIs which enables deploying gRPC applications in a service mesh without sidecar proxies. The proxyless service mesh features supported in gRPC are listed here.

How does gRPC help in mobile application development?
gRPC and Protobuf provide an easy way to precisely define a service and auto generate reliable client libraries for iOS, Android and the servers providing the back end. The clients can take advantage of advanced streaming and connection features which help save bandwidth, do more over fewer TCP connections and save CPU usage and battery life.

Why is gRPC better than any binary blob over HTTP/2?
This is largely what gRPC is on the wire. However gRPC is also a set of libraries that will provide higher-level features consistently across platforms that common HTTP libraries typically do not. Examples of such features include:

interaction with flow-control at the application layer
cascading call-cancellation
load balancing & failover

Why is gRPC better/worse than REST?

gRPC largely follows HTTP semantics over HTTP/2 but we explicitly allow for full-duplex streaming. We diverge from typical REST conventions as we use static paths for performance reasons during call dispatch as parsing call parameters from paths, query parameters and payload body adds latency and complexity. We have also formalized a set of errors that we believe are more directly applicable to API use cases than the HTTP status codes.

How do you pronounce gRPC?

Jee-Arr-Pee-See.