grpcurl curl for gRPC

CONCEPTS

Flags must come before the address and method; the address goes first, then list, describe or the method name.

grpcurl flags + JSON
↓ HTTP/2
gRPC server host:port
↓ schema from
Reflection or -proto / -protoset

Without -plaintext grpcurl uses TLS; a plain server then fails the handshake. Reflection must be enabled on the server, or give a schema file.

Requests and responses are JSON by default; -format text switches to protobuf text format.

DISCOVER

`grpcurl <host:port> list`
List services via reflection

`grpcurl <host:port> list <service>`
List a service's methods

`grpcurl <host:port> describe`
Describe every service

`grpcurl <host:port> describe <service>`
Describe a service

`grpcurl <host:port> describe <service>.<method>`
Describe one method

`grpcurl <host:port> describe <message>`
Describe a message type

`grpcurl -msg-template <host:port> describe <message>`
Show a JSON template for a message

`grpcurl -protoset <file> list`
List from a descriptor set

`grpcurl -import-path <dir> -proto <file> list`
List from a .proto file

CALL

`grpcurl <host:port> <service>/<method>`
Call with an empty request

`grpcurl -d '<json>' <host:port> <service>/<method>`
Call with a JSON body

`grpcurl -d @ <host:port> <service>/<method>`
Read the body from stdin

`grpcurl -format text -d '<text>' <host:port> <service>/<method>`
Use protobuf text format

`grpcurl -emit-defaults ...`
Print zero-valued fields

`grpcurl -allow-unknown-fields ...`
Ignore unknown request fields

`grpcurl -max-time <secs> ...`
Limit total time

`grpcurl -connect-timeout <secs> ...`
Limit connect time

`grpcurl -max-msg-sz <bytes> ...`
Raise response size limit

`grpcurl -version`
Print the version

CONNECTION AND TLS

`grpcurl -plaintext <host:port> ...`
Connect without TLS

`grpcurl -insecure <host:port> ...`
Skip server cert verification

`grpcurl -cacert <file> <host:port> ...`
Trust a custom CA

`grpcurl -cert <file> -key <file> <host:port> ...`
Present a client certificate (mTLS)

`grpcurl -authority <name> ...`
Set :authority and TLS name

`grpcurl -servername <name> ...`
Override the TLS server name

METADATA

`grpcurl -H '<name>: <value>' ...`
Send a header (repeatable)

`grpcurl -rpc-header '<name>: <value>' ...`
Header for the call only

`grpcurl -expand-headers -H 'authorization: Bearer ${TOKEN}' ...`
Expand env vars in headers

`grpcurl -v ...`
Show headers, trailers and details

SCHEMA SOURCES

`grpcurl -proto <file> -import-path <dir> ...`
Use .proto source (repeatable)

`grpcurl -protoset <file> ...`
Use a compiled descriptor set

`grpcurl -protoset-out <file> <host:port> describe <service>`
Save a descriptor set

`protoc --descriptor_set_out=<file> --include_imports <proto>`
Build a protoset with protoc

RECIPES

Explore an unknown plaintext server

```
grpcurl -plaintext
localhost:50051 list
grpcurl -plaintext
localhost:50051 list
my.pkg.Greeter
grpcurl -plaintext
localhost:50051 describe
my.pkg.HelloRequest
```

Authenticated call over TLS

```
grpcurl -H "authorization: Bearer
$TOKEN" \
  -d '{"name":"world"}' \
  api.example.com:443
my.pkg.Greeter/SayHello
```

Pipe a response through jq

```
grpcurl -plaintext -d '{"id":1}'
localhost:50051 \
  my.pkg.Items/Get | jq '.name'
```

Call from a local .proto with a JSON file

```
grpcurl -import-path ./protos -
proto items.proto \
  -d @ localhost:50051
my.pkg.Items/Create < req.json
```