



OVERVIEW

Server-Sent Events API
style/protocol for service
communication
Key principles and design philosophy
Compare to REST, SOAP, GraphQL
alternatives
Typical request
`curl -X POST
https://api.example.com/graphql \
-H 'Content-Type: application/json' \
-d '{"query": "{ users { id name } }"}'`

CORE CONCEPTS

Schema: defines types, queries,
mutations
Operations: Query (read), Mutation
(write), Subscription (real-time)
Resolver: function that returns data
for a field
Schema example:
`type User { id: ID!, name: String!,
email: String! }
type Query { user(id: ID!): User, users:
[User!]! }`

SETUP & SERVER

```
npm install apollo-server graphql
const { ApolloServer } =
  require('@apollo/server')
const server = new ApolloServer({
  typeDefs, resolvers })
await server.listen({ port: 4000 })
# GraphQL Playground at
http://localhost:4000
```

QUERIES & MUTATIONS

```
# Query
query { user(id: "1") { name email } }
# Mutation
mutation { createUser(name: "Alice",
  email: "a@b.com") { id } }
# Variables
query GetUser($id: ID!) { user(id: $id)
  { name } }
{ "id": "123" }
```

RESOLVERS

```
const resolvers = {
  Query: {
    user: (_, { id }, ctx) =>
      ctx.db.users.findById(id),
    users: (_, __, ctx) =>
      ctx.db.users.findAll(),
  },
  Mutation: {
    createUser: (_, args, ctx) =>
      ctx.db.users.create(args),
  },
}
```

AUTH & SECURITY

Add auth in context or middleware
`const server = new ApolloServer({
 context: async ({ req }) => ({
 user: await
 authenticate(req.headers.authorization),
 db,
 }},
})`
Validate and sanitize all inputs
Rate limit and depth-limit queries

ERROR HANDLING

```
throw new UserInputError('Invalid  
input', { field: 'email' })  
throw new AuthenticationError('Not  
authenticated')  
throw new ForbiddenError('Not  
authorized')  
Errors returned in errors[] array  
alongside data  
Use formatError to sanitize error  
messages in production
```

PERFORMANCE

N+1 problem: use DataLoader for
batching
`const userLoader = new DataLoader(ids =>
 batchLoadUsers(ids))
Query.user = (_, { id }) =>
 userLoader.load(id)`
Persisted queries: cache query
document hash
Response caching: @cacheControl
directive
Field-level tracing for performance
analysis

TOOLING

GraphQL Playground / Apollo Studio
interactive IDE
graphql-codegen generate types from
schema
graphql-inspector schema
diff/validation
Apollo Client frontend integration
urql lightweight GraphQL
client
graphql-shield permission layer

TESTING

```
# Apollo Server integration test
const { query } =
  createTestClient(server)
const { data } = await query({ query:
  GET_USER, variables: { id: '1' } })
expect(data.user.name).toBe('Alice')
```

Test resolvers in isolation with
mock context
Use Jest or Vitest as test runner

COMMON GOTCHAS

N+1 queries: always use DataLoader
for relations
Query depth limiting: prevent DoS
attacks
Don't expose internal error details
in production
Schema-first vs code-first: choose
one approach
Subscriptions need separate
WebSocket transport setup