Apollo GraphQL

Version Server v4 / Client v3 · verified 2026-08-01
A compact guide to Apollo Server and Client, queries, mutations, and real-time updates using WebSockets.

Contents

Mental model 2
Apollo Server setup 2
Apollo Client setup 3
Queries and mutations 4
Server-side subscriptions 5
Client-side subscriptions 6
Further reading 7

Mental model

Apollo provides a complete GraphQL ecosystem with a server for Node.js and a client for web apps. The server resolves operations into data, while the client fetches and caches it declaratively. Subscriptions enable real-time, long-lasting server-to-client updates over WebSockets.

Apollo Server setup

Setup	Purpose
`ApolloServer`	The main server class
`startStandaloneServer`	Quick HTTP server startup
`typeDefs`	The GraphQL schema definition
`resolvers`	Functions returning data for schema

You define your schema and resolvers, then start the server. A standalone server runs immediately on a chosen port.

```
import { ApolloServer } from "@apollo/server";
import { startStandaloneServer }
  from "@apollo/server/standalone";

const server = new ApolloServer({
  typeDefs: `type Query { h: String }`,
  resolvers: { Query: { h: () => "w" } }
});
const { url } = await startStandaloneServer(
```

```
  server, { listen: { port: 4000 } }
);
```

Apollo Client setup

Setup	Purpose
`ApolloClient`	Core client manager
`InMemoryCache`	Normalizes and stores responses
`ApolloProvider`	Context provider for React
`HttpLink`	Link for network requests

The client manages fetching and caching. Provide it via context to make hooks available to the React component tree.

```
import { ApolloClient, InMemoryCache, HttpLink }
  from "@apollo/client/core/index.js";
import { ApolloProvider }
  from "@apollo/client/react/index.js";

const client = new ApolloClient({
  link: new HttpLink({ uri: "/g" }),
  cache: new InMemoryCache()
});

export function App({ children }: any) {
  return <ApolloProvider client={client}>
```

```
    {children}
  </ApolloProvider>;
}
```

Queries and mutations

Hook	Operation
`useQuery(gql)`	Fetches data and tracks state
`useMutation(gql)`	Modifies data, returns execute fn
`gql`	Template literal for GraphQL strings

Queries are executed upon render, providing `loading`, `error`, and `data` states. Mutations provide an execution function.

```
// @ts-expect-error
import { gql, useQuery, useMutation }
  from "@apollo/client/react";

const GET = gql`query { u { n } }`;
const SET = gql`mutation($n: String!) { s(n:$n) }`;

export function User() {
  const { data, loading } = useQuery<any>(GET);
  const [set] = useMutation(SET);
  if (loading) return <div>...</div>;
  return <button onClick={() => set({
```

```
    variables: { n: "A" }
  })}>{data.u.n}</button>;
}
```

> **Gotcha:** `useMutation` does not automatically execute upon render. You must invoke the returned function manually.

Server-side subscriptions

Setup	Purpose
`graphql-ws`	WebSocket protocol implementation
`PubSub`	Event bus (not for production)
`wsServer`	The WebSocket server instance

Subscriptions require an HTTP server and a WebSocket server sharing the same port. The `expressMiddleware` integration supports this.

```
import { WebSocketServer } from "ws";
declare const useServer: any;
import { makeExecutableSchema } //
  from "@graphql-tools/schema";
import { PubSub } from "graphql-subscriptions";

const schema = makeExecutableSchema({
  typeDefs: `type Subscription { t: String }`,
  resolvers: { Subscription: { t: { subscribe: () =>
```

```
    // @ts-expect-error
    new PubSub().asyncIterator("T") } } }
});

const w = new WebSocketServer({ port: 400 });
useServer({ schema }, w as any);
```

> **Warning:** The default `PubSub` class is in-memory only. Multiple server instances will not share events unless backed by Redis or Kafka.

Client-side subscriptions

Hook / Setup	Purpose
`GraphQLWsLink`	Link for WebSocket communication
`split`	Routes queries vs. subscriptions
`useSubscription`	Consumes streaming updates

Use `split` to route operations: WebSockets for subscriptions and HTTP for queries/mutations.

```
import { split, HttpLink, ApolloClient }
  from "@apollo/client/core/index.js";
import { GraphQLWsLink }
  from "@apollo/client/link/subscriptions/index.js";
import { createClient } from "graphql-ws";
```

```
const link = split(({ query }: any) => {
  return query.operation === "subscription";
}, new GraphQLWsLink(createClient({ url: "w" })),
   new HttpLink({ uri: "h" }));
const c = new ApolloClient({
  link, cache: {} as any
});
```

Further reading

- Apollo Server documentation
- Apollo Client documentation
- graphql-ws protocol