Migrating to v11 from v10

This chapter provides a set of guidelines for migrating from `@nestjs/graphql` version 10 to version 11. As part of this major release, we updated the Apollo driver to be compatible with Apollo Server v4 (instead of v3). Note: there are several breaking changes in Apollo Server v4 (especially around plugins and ecosystem packages), so you'll have to update your codebase accordingly. For more information, see the Apollo Server v4 migration guide.

Apollo packages

Instead of installing the `apollo-server-express` package, you'll have to install `@apollo/server`:

```
$ npm uninstall apollo-server-express
$ npm install @apollo/server
```

If you use the Fastify adapter, you'll have to install the `@as-integrations/fastify` package instead:

```
$ npm uninstall apollo-server-fastify
$ npm install @apollo/server @as-integrations/fastify
```

Mercurius packages

Mercurius gateway is no longer a part of the `mercurius` package. Instead, you'll have to install the `@mercuriusjs/gateway` package separately:

```
$ npm install @mercuriusjs/gateway
```

Similarly, for creating federated schemas, you'll have to install the `@mercuriusjs/federation` package:

```
$ npm install @mercuriusjs/federation
```

Migrating to v10 from v9

This chapter provides a set of guidelines for migrating from `@nestjs/graphql` version 9 to version 10. The focus of this major-version release is to provide a lighter, platform-agnostic core library.

Introducing "driver" packages

In the latest version, we made a decision to break the `@nestjs/graphql` package up into a few separate libraries, letting you choose whether to use Apollo (`@nestjs/apollo`), Mercurius (`@nestjs/mercurius`), or another GraphQL library in your project.

This implies that now you have to explicitly specify what driver your application will use.

```
// Before
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';

@Module({
  imports: [
    GraphQLModule.forRoot({
      autoSchemaFile: 'schema.gql',
    }),
  ],
})
export class AppModule {}

// After
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';

@Module({
  imports: [
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      autoSchemaFile: 'schema.gql',
    }),
  ],
})
export class AppModule {}
```

Plugins

Apollo Server plugins let you perform custom operations in response to certain events. Since this is an exclusive Apollo feature, we moved it from the `@nestjs/graphql` to the newly created `@nestjs/apollo` package so you'll have to update imports in your application.

```
// Before
import { Plugin } from '@nestjs/graphql';

// After
import { Plugin } from '@nestjs/apollo';
```

Directives

`schemaDirectives` feature has been replaced with the new Schema directives API in v8 of `@graphql-tools/schema` package.

```
// Before
import { SchemaDirectiveVisitor } from '@graphql-tools/utils';
import { defaultFieldResolver, GraphQLField } from 'graphql';

export class UpperCaseDirective extends SchemaDirectiveVisitor {
  visitFieldDefinition(field: GraphQLField<any, any>) {
    const { resolve = defaultFieldResolver } = field;
    field.resolve = async function (...args) {
      const result = await resolve.apply(this, args);
      if (typeof result === 'string') {
        return result.toUpperCase();
      }
      return result;
    };
  }
}

// After
import { getDirective, MapperKind, mapSchema } from '@graphql-
tools/utils';
import { defaultFieldResolver, GraphQLSchema } from 'graphql';

export function upperDirectiveTransformer(
  schema: GraphQLSchema,
  directiveName: string,
) {
  return mapSchema(schema, {
    [MapperKind.OBJECT_FIELD]: (fieldConfig) => {
      const upperDirective = getDirective(
        schema,
        fieldConfig,
        directiveName,
      )?.[0];

      if (upperDirective) {
        const { resolve = defaultFieldResolver } = fieldConfig;

        // Replace the original resolver with a function that *first*
calls
        // the original resolver, then converts its result to upper case
        fieldConfig.resolve = async function (source, args, context, info)
{
          const result = await resolve(source, args, context, info);
          if (typeof result === 'string') {
            return result.toUpperCase();
          }
          return result;
        };
        return fieldConfig;
      }
    },
  });
}
```

To apply this directive implementation to a schema that contains `@upper` directives, use the `transformSchema` function:

```
GraphQLModule.forRoot<ApolloDriverConfig>({
  ...
  transformSchema: schema => upperDirectiveTransformer(schema, 'upper'),
})
```

Federation

`GraphQLFederationModule` has been removed and replaced with the corresponding driver class:

```
// Before
GraphQLFederationModule.forRoot({
  autoSchemaFile: true,
});

// After
GraphQLModule.forRoot<ApolloFederationDriverConfig>({
  driver: ApolloFederationDriver,
  autoSchemaFile: true,
});
```

> info **Hint** Both `ApolloFederationDriver` class and `ApolloFederationDriverConfig` are exported from the `@nestjs/apollo` package.

Likewise, instead of using a dedicated `GraphQLGatewayModule`, simply pass the appropriate `driver` class to your `GraphQLModule` settings:

```
// Before
GraphQLGatewayModule.forRoot({
  gateway: {
    supergraphSdl: new IntrospectAndCompose({
      subgraphs: [
        { name: 'users', url: 'http://localhost:3000/graphql' },
        { name: 'posts', url: 'http://localhost:3001/graphql' },
      ],
    }),
  },
});

// After
GraphQLModule.forRoot<ApolloGatewayDriverConfig>({
  driver: ApolloGatewayDriver,
  gateway: {
    supergraphSdl: new IntrospectAndCompose({
      subgraphs: [
        { name: 'users', url: 'http://localhost:3000/graphql' },
        { name: 'posts', url: 'http://localhost:3001/graphql' },
      ],
    }),
  },
});
```

> info **Hint** Both `ApolloGatewayDriver` class and `ApolloGatewayDriverConfig` are exported from the `@nestjs/apollo` package.