Performance (Fastify)

By default, Nest makes use of the Express framework. As mentioned earlier, Nest also provides compatibility with other libraries such as, for example, Fastify. Nest achieves this framework independence by implementing a framework adapter whose primary function is to proxy middleware and handlers to appropriate library-specific implementations.

> info **Hint** Note that in order for a framework adapter to be implemented, the target library has to provide similar request/response pipeline processing as found in Express.

Fastify provides a good alternative framework for Nest because it solves design issues in a similar manner to Express. However, fastify is much **faster** than Express, achieving almost two times better benchmarks results. A fair question is why does Nest use Express as the default HTTP provider? The reason is that Express is widely-used, well-known, and has an enormous set of compatible middleware, which is available to Nest users out-of-the-box.

But since Nest provides framework-independence, you can easily migrate between them. Fastify can be a better choice when you place high value on very fast performance. To utilize Fastify, simply choose the built-in `FastifyAdapter` as shown in this chapter.

Installation

First, we need to install the required package:

```
$ npm i --save @nestjs/platform-fastify
```

Adapter

Once the Fastify platform is installed, we can use the `FastifyAdapter`.

```
@@filename(main)
import { NestFactory } from '@nestjs/core';
import {
  FastifyAdapter,
  NestFastifyApplication,
} from '@nestjs/platform-fastify';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create<NestFastifyApplication>(
    AppModule,
    new FastifyAdapter()
  );
  await app.listen(3000);
}
bootstrap();
```

By default, Fastify listens only on the `localhost 127.0.0.1` interface (read more). If you want to accept connections on other hosts, you should specify `'0.0.0.0'` in the `listen()` call:

```
async function bootstrap() {
  const app = await NestFactory.create<NestFastifyApplication>(
    AppModule,
    new FastifyAdapter(),
  );
  await app.listen(3000, '0.0.0.0');
}
```

Platform specific packages

Keep in mind that when you use the `FastifyAdapter`, Nest uses Fastify as the **HTTP provider**. This means that each recipe that relies on Express may no longer work. You should, instead, use Fastify equivalent packages.

Redirect response

Fastify handles redirect responses slightly differently than Express. To do a proper redirect with Fastify, return both the status code and the URL, as follows:

```
@Get()
index(@Res() res) {
  res.status(302).redirect('/login');
}
```

Fastify options

You can pass options into the Fastify constructor through the `FastifyAdapter` constructor. For example:

```
new FastifyAdapter({ logger: true });
```

Middleware

Middleware functions retrieve the raw `req` and `res` objects instead of Fastify's wrappers. This is how the `middie` package works (that's used under the hood) and `fastify` - check out this page for more information,

```
@@filename(logger.middleware)
import { Injectable, NestMiddleware } from '@nestjs/common';
import { FastifyRequest, FastifyReply } from 'fastify';

@Injectable()
```

```
export class LoggerMiddleware implements NestMiddleware {
  use(req: FastifyRequest['raw'], res: FastifyReply['raw'], next: () =>
void) {
    console.log('Request...');
    next();
  }
}
@@switch
import { Injectable } from '@nestjs/common';

@Injectable()
export class LoggerMiddleware {
  use(req, res, next) {
    console.log('Request...');
    next();
  }
}
```

Route Config

You can use the route config feature of Fastify with the `@RouteConfig()` decorator.

```
@RouteConfig({ output: 'hello world' })
@Get()
index(@Req() req) {
  return req.routeConfig.output;
}
```

> info **Hint** `@RouteConfig()` is imported from `@nestjs/platform-fastify`.

Example

A working example is available here.