SETTING UP NESTJS WITH FIREBASE FUNCTIONS

Szymon Bartczak
11 February 2020 • #nodejs #api #development #web-development

What if we could deploy a backend application in just a few steps, without going through the overly complicated configuration process? Imagine doing so without worrying if it can keep up with incoming traffic and get all these domain certificates already on board. This is actually possible with Firebase Functions and any framework based on Express (including Express itself).

Firebase Functions is an alternative to AWS Lambda created with ease of use and fast deployment in mind. According to official documentation, “Functions let you automatically run backend code in response to events triggered by Firebase features and HTTPS requests.” You can write any JS code and run it with a simple request or by setting up a trigger. What is more, Functions require exactly the same parameters as ExpressJS, meaning we can just forward anything through Functions, and let Express take care of it. This enables the use of the service as a deployment server for the whole backend application.

“NestJS is a framework for building efficient, scalable Node.js server-side applications.” Nest philosophy is to provide good architecture, regardless of the size of the project. Since its architecture was inspired by Angular, it takes advantage of most modern JavaScript and TypeScript practises. Under the hood, it uses either Express or Fastify. We can use this to our advantage, using Express from Nest as a Firebase Function.

Table of content

WHY?

The whole deployment process is very hard to set up right and fast using Kubernetes or Docker. If you are creating a small side project or a startup with little traffic, you want to cheaply set up deployment in minutes. Firebase Functions are (to some extent) free and do all the necessary stuff like load balancing and SSL out of the box. It's a good match for such needs.

But why mix it with NestJS? Of course, you can simply create a small function to do the exact task you need, but sometimes a more organised structure with more possibilities is needed. NestJS is a good candidate when planning to write a micro service, but it's also a great tool for standard backend app.

PRE-SETUP

Before we begin to setup the whole application, there are some preparations we have to take care of:

1. **Download Firebase CLI** – it will be needed for deployment and initial configuration
   $ npm install -g firebase-tools

2. **Create Firebase project (optionally)** – this will come in handy when setting up Firebase Functions, because we can already select a project during this process. You can create a new project here https://console.firebase.google.com/. Projects can also be created with the firebase init command.

3. **Download Nestjs CLI** – this step could be considered optional, but will massively speed things up, so I've decided to keep it. If you really don't want this module, you can skip it and configure the nest project by yourself, or use some kind of boilerplate.
   $ npm install -g @nestjs/cli

SETUP

Now, once you are ready, go to the directory where you want your project to be stored. Please note that you don't have to create a project directory – it will be created automatically in the 1st step.

1. Create a Nest project as usual. I've chosen the standard way by executing:
   `$ nest new project-name`
   This will automatically generate directory, all startup files, configuration (including package.json) and initialize the Git version control system.

2. Login to Firebase:
   `$ firebase login`

3. Go to project directory and initialize Firebase Functions:
   `$ firebase init functions`
   There will be 4 steps:
   1. Associating a project.
      From there, you can select the project you created. This will be treated as the default project. This can be modified later if you want.

   2. Choosing language.
      Select TypeScript as Nest use TypeScript

   3. Lint configuration.
      It's up to you whether you want the Firebase Tslint config, but because it's deprecated, I don't recommend it. Configure Eslint for your project on your own.

   4. Installing dependencies with npm
      I wouldn't recommend doing it, because we will use package.json generated by Nest.

4. Modify package.json
   Go to the `functions` directory and open `package.json` . We will copy part of this file to `package.json` in the project directory.
   1. Copy scripts:

```
"serve": "npm run build && firebase serve --only functions",
"shell": "npm run build && firebase functions:shell",
"start": "npm run shell",
"deploy": "firebase deploy --only functions",
"logs": "firebase functions:log"
```

If you are using yarn you should also change all *npm run -> yarn*

2. Setup Node version.

Firebase supports Node 8 and Node 10 (still Beta when writing this article). Node 8 is what was generated by Firebase, so I left it as is.

```
"engines": {
   "node": "8"
 },
```

3. Copy the necessary dependencies

```
"dependencies": {
    "firebase-admin": "^8.6.0",
    "firebase-functions": "^3.3.0"
  },
  "devDependencies": {
    "typescript": "^3.2.2", – this is probably already added
    "firebase-functions-test": "^0.1.6" – this is optional
  },
```

4. Setup main:

```
"main": "dist/index.js"
```

The file should look like this:

```
{
  "name": "nest-functions",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "license": "MIT",
  "main": "dist/index.js",
  "engines": {
    "node": "8"
  },
  "scripts": {
    "build": "rimraf dist && tsc -p tsconfig.build.json",
    "serve": "npm run build && firebase serve --only functions",
    "shell": "npm run build && firebase functions:shell",
    "deploy": "firebase deploy --only functions",
    "logs": "firebase functions:log",
    "format": "prettier --write \"src/**/*.ts\"",
    "start": "ts-node -r tsconfig-paths/register src/main.ts",
    "start:func": "npm run shell",
    "start:dev": "tsc-watch -p tsconfig.build.json --onSuccess \"node
    "start:debug": "tsc-watch -p tsconfig.build.json --onSuccess \"n
    "start:prod": "node dist/main.js",
    "lint": "tslint -p tsconfig.json -c tslint.json",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^6.7.2",
    "@nestjs/core": "^6.7.2",
    "@nestjs/platform-express": "^6.7.2",
    "reflect-metadata": "^0.1.13",
    "rimraf": "^3.0.0",
    "rxjs": "^6.5.3",
    "firebase-admin": "^8.6.0",
    "firebase-functions": "^3.3.0"
  },
  "devDependencies": {
    "@nestjs/cli": "^6.8.1",
    "@nestjs/schematics": "^6.6.6",
    "@nestjs/testing": "^6.7.1",
    "@types/express": "^4.17.1",
    "@types/jest": "^24.0.18",
    "@types/node": "^12.7.5",
    "@types/supertest": "^2.0.8",
    "jest": "^24.9.0",
    "prettier": "^1.18.2",
    "supertest": "^4.0.2",
    "ts-jest": "^24.1.0",
    "ts-loader": "^6.1.1",
    "ts-node": "^8.4.1",
    "tsc-watch": "^2.4.0",
    "tsconfig-paths": "^3.9.0",
    "tslint": "^5.20.0",
    "typescript": "^3.6.3"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".spec.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "coverageDirectory": "./coverage",
    "testEnvironment": "node"
  }
}
```

5. Remove the functions directory, as it is not needed anymore

6. Modify firebase.json
   We need to tell Firebase where to look for functions. Firebase is looking for the index file in the specified directory. Since firebase.json is in the same directory as the Nest application we can just add:
   "source": "."
   Important! This should be added inside the functions property.

The whole file should look like this:

```
{
  "functions": {
    "predeploy": "npm --prefix \"$RESOURCE_DIR\" run build",
    "source": "."
  }
}
```

7. Run package installation
   $ yarn or $ npm install

8. Create the index file as an entry point. The file should be located in the same directory as the source of firebase.json.

```
import { NestFactory } from '@nestjs/core';
import { ExpressAdapter } from '@nestjs/platform-express';
import * as express from 'express';
import * as functions from 'firebase-functions';

import { AppModule } from './app.module';

const expressServer = express();

const createFunction = async (expressInstance): Promise<void> => {
  const app = await NestFactory.create(
    AppModule,
    new ExpressAdapter(expressInstance),
  );

  await app.init();
};

export const api = functions.region('europe-west1')
  .https.onRequest(async (request, response) => {
    await createFunction(expressServer);
    expressServer(request, response);
  });
```

The important thing is whatever you name that exported variable, that will be the name of the function in Firebase Functions. I called mine `api`, so the entry point to my application will be `{FIREBASE_ADRESS}/api`.
Another important thing is that if you don't specify a region, it's set by default to `us-central1`.

OK, that's it for now, I hope you like this article and you'll find it useful! If you'd like to talk more or you have any further questions - feel free to drop us a line at malgorzata.galinska@softwarebrothers.co

RELATED ARTICLES

BUILDING A BULLETPROOF JAVASCRIPT API CLIENTS WITH NOCK - JSCASTS EPISODE 15

In this episode, I will tell you how to write bulletproof API clients in javascript using Test Driven Development. The most interesting part will be unit tests, where we will use Nock - server mocking and expectation library.

READ MORE

SERVER SIDE RENDERING WITH REACT AND EXPRESS - JSCASTS EPISODE 14

Next episode of JSCast. From episode 14 of this series, you will learn how to add Server Side Rendering with React.

READ MORE

RENDER ONE FILE HTML PAGE WITH AWS API GATEWAY+LAMBDA USING NODE.JS WITH PUG

Last month, I decided to improve the way our clients monitor progress in their projects. To deal with that issue, our UX team created a design for a beautiful and usable report. A couple of hours after receiving the designs, I implemented them into a service serving single HTML files.

READ MORE

Our work was featured on

ALSO ON **HOWWEDOIT**

2 years ago • 1 comment
Setting up CI&CD for VPS-hosted apps …

2 years ago • 1 comment
Solution to the View Layer's Problems in …

a year ago
Hybric Progra Comp

2 Comments **howwedoit** 1 **Login**

Recommend Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Victor • 5 months ago
Ok... Cool! Nice write up. Is there a part 2? What can we do with this project? Can you provide a sample API request? A sample cloud function?
2 | • Reply • Share ›

Alexey Nagornov • 2 months ago • edited
Looks like this code will initialize nest app on each request. I've tested it and my suspicion has been confirmed.