Angular Micro Frontends Simplified: Implementing Dynamic Module Federation with Nx

Manohar Taneti Follow 3 min read · Jun 28, 2024

1)Installing Nx Globally

- “open cmd (Windows+R) to install Nx globally”
- “npm install -g nx”

2)Workspace Explanation

- “A workspace is a folder created using Nx.”
- “The folder consists of a single git repository.”

- “Folders for apps (applications) and libs (libraries)”
- “Scaffolding to help with building, linting, and testing”

3)To Create an Nx Workspace

- “**npx create-nx-workspace ng-mfe**”
- You’ll be prompted a few questions. Pick the `Angular` stack, `Integrated Monorepo` layout and the `webpack` bundler. You can use the default values for the rest of the prompts. We won't use the application that gets generated by default on this guide, so you can remove it or leave it as it is that is not a problem.

4)**Creating our applications**

We need to generate two applications that support Module Federation.

We’ll start with the Admin Dashboard application which will act as a host application for the Micro-Frontends (**MFEs**):

nx g @nx/angular:host apps/dashboard — prefix=ng-mf

The `host` generator will create and modify the files needed to set up the Angular application.

Now, let's generate the Login application as a remote application that will be consumed by the Dashboard host application.

nx g @nx/angular:remote apps/login — prefix=ng-mf — host=dashboard

Note how we provided the `--host=dashboard` option. This tells the generator that this remote application will be consumed by the Dashboard application. The generator performed the following changes to automatically link these two applications together:

- Added the remote to the `apps/dashboard/module-federation.config.ts` file
- Added a TypeScript path mapping to the root tsconfig file
- Added a new route to the `apps/dashboard/src/app/app.routes.ts` file

5)**What was generated?**

Let's take a closer look after generating each

application.

For both applications, the generators did the following:

- Created the standard Angular application files
- Added a `module-federation.config.ts` file
- Added a `webpack.config.ts` and `webpack.prod.config.ts`
- Added a `src/bootstrap.ts` file
- Moved the code that is normally in `src/main.ts` to `src/bootstrap.ts`
- Changed `src/main.ts` to dynamically import `src/bootstrap.ts` *(this is required for the Module Federation to load versions of shared libraries correctly)*
- Updated the `build` target in the `project.json` to use the `@nx/angular:webpack-browser` executor *(this is required to support passing a custom Webpack configuration to the Angular compiler)*
- Updated the `serve` target to use `@nx/angular:dev-server` *(this is required as we first need Webpack to build the application with our custom Webpack configuration)*

The key differences reside within the configuration of the Module Federation Plugin within each application's `module-federation.config.ts`.

6)We can now run both the Dashboard and Login applications:

nx serve dashboard — devRemotes=login

Navigating to `http://localhost:4200` should show the Dashboard application with the Login application embedded within it.

This concludes the setup required for a Micro Frontend approach using **Static Module Federation.**

Transitioning from Static to Dynamic Module Federation with Nx:

To convert a static module federation setup to dynamic module federation in your Nx workspace, follow these steps:

> **1.Define Remote Modules in a Manifest in Dashboard Application**
> *Create a `**module-federation.manifest.json**` file in **src/assests/** folder with the following content in our*

> *Dashboard :*

```
{
 "login": "http://localhost:4201"
}
```

2. Update `main.ts` in the Dashboard

Modify your `*main.ts*` file to fetch and set remote definitions dynamically:

```
import { setRemoteDefinitions } from '@nx/angular

fetch('/assets/module-federation.manifest.json')
  .then((res) => res.json())
  .then((definitions) => setRemoteDefinitions(def
  .then(() => import('./bootstrap').catch((err) =
```

3. Clear Static Remote Definitions

Ensure the `*remotes*` array in your module federation configuration is empty in Dashboard:

```
// module-federation.config.ts
module.exports = {
  remotes: []
};
```

4. Configure Remote Module Loading in Routes

Update your `*app.routes.ts*` to load remote modules dynamically:

```
import { loadRemoteModule } from '@nx/angular/mf'

const routes = [
  {
    path: 'login',
    loadChildren: () =>
      loadRemoteModule('login', './Routes').then(
  },
];
```

These changes are all you need to replace static module federation with dynamic module federation in your Nx workspace.

5. Serve the Application with Remote Modules

Finally, serve your application using the following command:.

```
nx serve dashboard --devRemotes=login
```

By following these steps, you can seamlessly transition from static to dynamic module federation, enhancing the flexibility and scalability of your micro frontend architecture

with Nx.

Module Federation Angular

Written by Manohar Taneti

9 followers · 7 following

Follow

No responses yet

László Kővári

What are your thoughts?

More from Manohar Taneti

Manohar Taneti

Revolutionize Your Angular Apps with...

Are you an Angular developer looking to...

Apr 8, 2024 3

Manohar Taneti

Microfrontends using NX in Angular

Microfrontends: An architectural style where...

Nov 22, 2023 4 1

See all from Manohar Taneti

Recommended from Medium

Vetriselvan Panneerselvam

Angular 21 Monorepo Micro Frontends wit...

Learn how to build a micro frontend architecture in an...

Jan 24 149

Aukevanoost

Migrating a stateful monolith to micro...

This article will dive into the challenges of integrating...

Aug 10, 2025 3

In Fronten... by Abhiman R...

Essential Considerations for...

When building micro frontends for a multi-tenan...

Aug 26, 2025

Satendra Sharma

Practical Module Federation with JS

Backend Services became scalable , manageable ,...

Oct 1, 2025

Tanvir Khan

Embracing Microfrontends: A...

I still remember the day our team's frontend monolith...

Dec 8, 2025

Piyali Das

Angular 19 SSE using EventSource

Server-Sent Events (SSE) is a web technology that...

Jan 28 3

See more recommendations