Quick Start Guide

Before reading this guide, please first read the Setting Up Environment. This guide will lead you step by step to learn how to use Module Federation. We will build two independent Single Page Applications (SPAs) that will share components using Module Federation. If you encounter unfamiliar terms in the following text, please refer to the Glossary.

Creating a Producer

1. Initializing the Project

Use Rsbuild to create a producer and call the following command:

npm yarn pnpm bun

```
npm create rsbuild@latest
```

> Complete the project creation according to the prompts.

```
? Input target folder -> federation_provider
? Select framework -> React
? Select language -> TypeScript
```

2. Installing Module Federation Build Plugin

Following the steps above for initializing the project, we created a `React` project named `federation_provider`. Execute the following commands in sequence:

npm yarn pnpm bun

```
cd federation_provider
npm add @module-federation/enhanced
```

3. Exporting Modules by the Producer

> Change the entry file to asynchronous

```
// Move src/index.tsx to a newly created src/bootstrap.tsx file
// src/index.tsx
import('./bootstrap');

// src/bootstrap.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root')!);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

> Add a Button Component

```
// src/button.tsx
export default function Button() {
  return <div>Provider button</div>;
}
```

> Export the Button component through Module Federation

```
// rsbuild.config.ts
import { defineConfig } from '@rsbuild/core';
import { pluginReact } from '@rsbuild/plugin-react';
import { ModuleFederationPlugin } from '@module-federation/enhanced/rspack';

export default defineConfig({
  server: {
      port: 3000,
    },
    dev: {
      // It is necessary to configure assetPrefix, and in the production environment, you need to configure o
      assetPrefix: 'http://localhost:3000',
    },
    tools: {
      rspack: (config, { appendPlugins }) => {
        // You need to set a unique value that is not equal to other applications
        config.output!.uniqueName = 'federation_provider';
        appendPlugins([
          new ModuleFederationPlugin({
            name: 'federation_provider',
            exposes: {
              './button': './src/button.tsx',
            },
            shared: ['react', 'react-dom'],
          }),
        ]);
      },
    },
    plugins: [pluginReact()],
  });
```

4. Starting the Producer

npm yarn pnpm bun

```
npm run dev
```

```
➜  federation_provider npm run dev

> federation_provider@1.0.0 dev
> rsbuild dev --open

  Rsbuild v0.5.1

  > Local:    http://localhost:3000/
  > Network:  http://10.94.55.204:3000/
  > Network:  http://10.4.255.21:3000/

start   Compiling...
[ Module Federation Manifest Plugin ] Manifest Link: http://localhost:3000/mf-manifest.json
```

After the project starts, the `Manifest Link: http://localhost:3000/mf-manifest.json` message appears, which is the manifest information link for Module Federation.

Creating a Consumer

1. Initializing the Project

Use Rsbuild to create a consumer and call the following command:

npm yarn pnpm bun

```
npm create rsbuild@latest
```

> Complete the project creation according to the prompts.

```
? Input target folder -> federation_consumer
? Select framework -> React
? Select language -> TypeScript
```

2. Installing Module Federation Build Plugin

Following the steps above for initializing the project, we created a `React` project named `federation_consumer`. Execute the following commands in sequence:

npm yarn pnpm bun

```
cd federation_consumer
npm add @module-federation/enhanced
```

3. Consuming the Producer

> Declare the `Module Federation` type path in `tsconfig.json`

```
{
  "compilerOptions": {
    "paths":{
      "*": ["./@mf-types/*"]
    }
  }
}
```

> Add Module Federation plugin to consume remote modules

```
// rsbuild.config.ts
import { defineConfig } from '@rsbuild/core';
import { pluginReact } from '@rsbuild/plugin-react';
import { ModuleFederationPlugin } from '@module-federation/enhanced/rspack';

export default defineConfig({
  server: {
    port: 2000,
  },
  tools: {
    rspack: (config, { appendPlugins }) => {
      appendPlugins([
        new ModuleFederationPlugin({
          name: 'federation_consumer',
          remotes: {
            federation_provider:
              'federation_provider@http://localhost:3000/mf-manifest.json',
          },
          shared: ['react', 'react-dom'],
        }),
      ]);
    },
  },
  plugins: [pluginReact()],
});
```

> Change the entry to be asynchronous

```
// Move src/index.tsx to a newly created src/bootstrap.tsx file
// src/index.tsx
import('./bootstrap');

// src/bootstrap.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root')!);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

> Reference the remote module

```
import './App.css';
// The remote component provided by federation_provider
import ProviderButton from 'federation_provider/button';

const App = () => {
  return (
    <div className="content">
      <h1>Rsbuild with React</h1>
      <p>Start building amazing things with Rsbuild.</p>
      <div>
        <ProviderButton />
      </div>
    </div>
  );
};

export default App;
```

Summary

Through the above process, you have completed the export of a component from a 'producer' for use by a 'consumer' based on Module Federation. Along the way, you have preliminarily used and understood the configurations of remotes, exposes, and shared in the Module Federation plugin.

If you wish to learn how to directly export and consume remote modules on Webpack and Rspack build tools, you can refer to: Rspack Plugin, Webpack Plugin

---

Next Steps

You may want to:

Feature Navigation

Learn about all the features provided by Module Federation

Review Configuration

Learn how to configure Module Federation

Glossary of Terms

Understand the concepts related to Module Federation