cognizant softvision

PROGRAMMERS' WEEK 2022

Rapid Micro Frontend Development with Module Federation

Doug Braxton

Agenda

- Micro Frontends Overview
- What is Module Federation?
- Using Module Federation to build Micro Frontends
- How Should I Divide My Application?
- Tools That Make It Easier
- Recent Implementation Lessons Learned

PROGRAMMERS´
WEEK 2022

Micro Frontends Overview

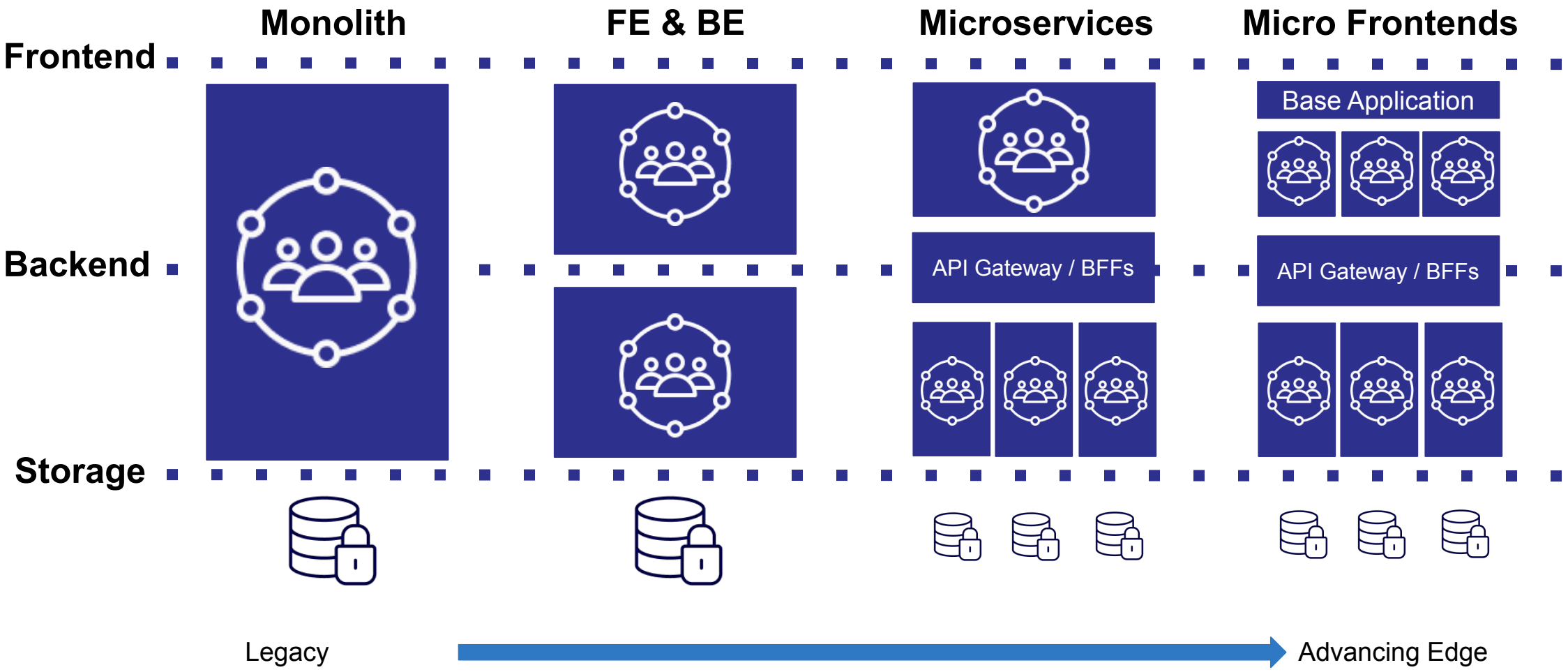
cognizant
softvision

What are Micro Frontends?

In the simplest terms, a Micro Frontend architecture is an application built out of smaller, domain specific applications.

- Micro frontends take the concept of micro services and extend it to the front-end.
- Micro frontends are a composition of features (apps) owned by independent teams.
- Each team is (ideally) responsible for the development of their features end-to-end, from the front-end all the way to the database.
- Each feature (app) is technology agnostic and not constrained by the development of other features.

What are Micro Frontends?



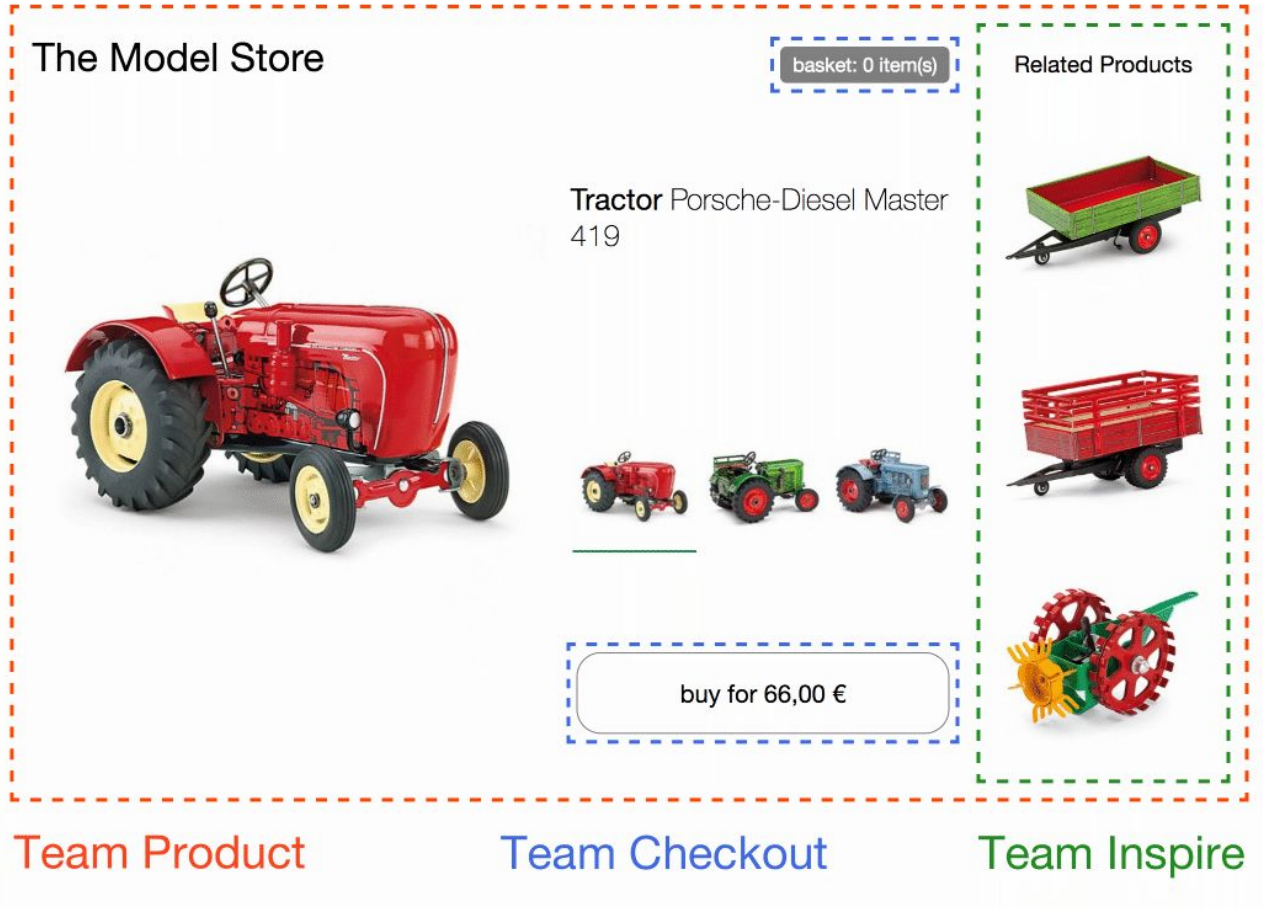
What are Micro Frontends?

In this example from https://micro-frontends.org/ we see a store application that is used as an example of how to migrate to micro front-ends:

Team Checkout (in blue): provides a seamless checkout experience app. When a user clicks on a selection, it updates the ‘buy’ button with the price. When the button is clicked, it adds the item to running total and updates the basket.

Team Inspire (in green): provides a list of related products based on a product selection.

Each one of these respective areas operate completely independent of one another. Each team’s components can be used on completely different websites or applications.



How are Micro Frontends Made?

Generally speaking, Micro Frontends are more of a **concept** than a strictly defined term. Technologically speaking, they have been around a very long time and there's no one way to make them. That being said, here are the main approaches a development team can take:

- **Server-side template composition**: uses server-side includes to plug in page-specific content from fragment HTML files
- **Build-time integration**: publish each micro frontend as a package, and have the container application include them
- **Run-time integration via iframes**: uses iframes to include content from other hosted pages
- **Run-time integration via JavaScript (most popular)**: micro frontends are included onto the page using a <script> tag, and upon load render in defined areas

We will be discussing the final method for the remainder of this presentation.

PROGRAMMERS´
WEEK 2022

Webpack & Module Federation Overview

cognizant
softvision

What is Webpack?

- A *static module bundler*
- Robust plugin ecosystem for various transformation and bundling tasks
- Readies development code for production consumption

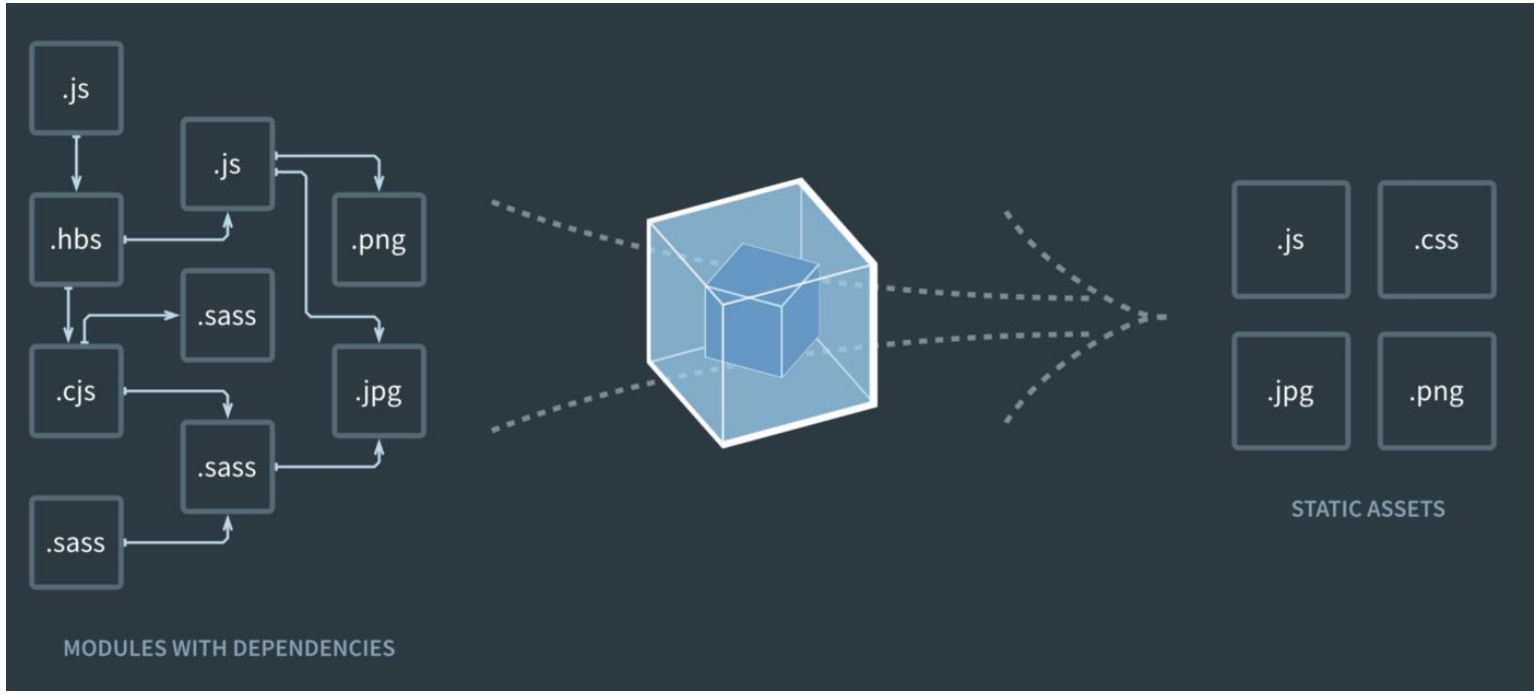


Image sourced from main page of https://webpack.js.org

What is Module Federation?

"Module Federation allows a JavaScript application to dynamically load code from another application"

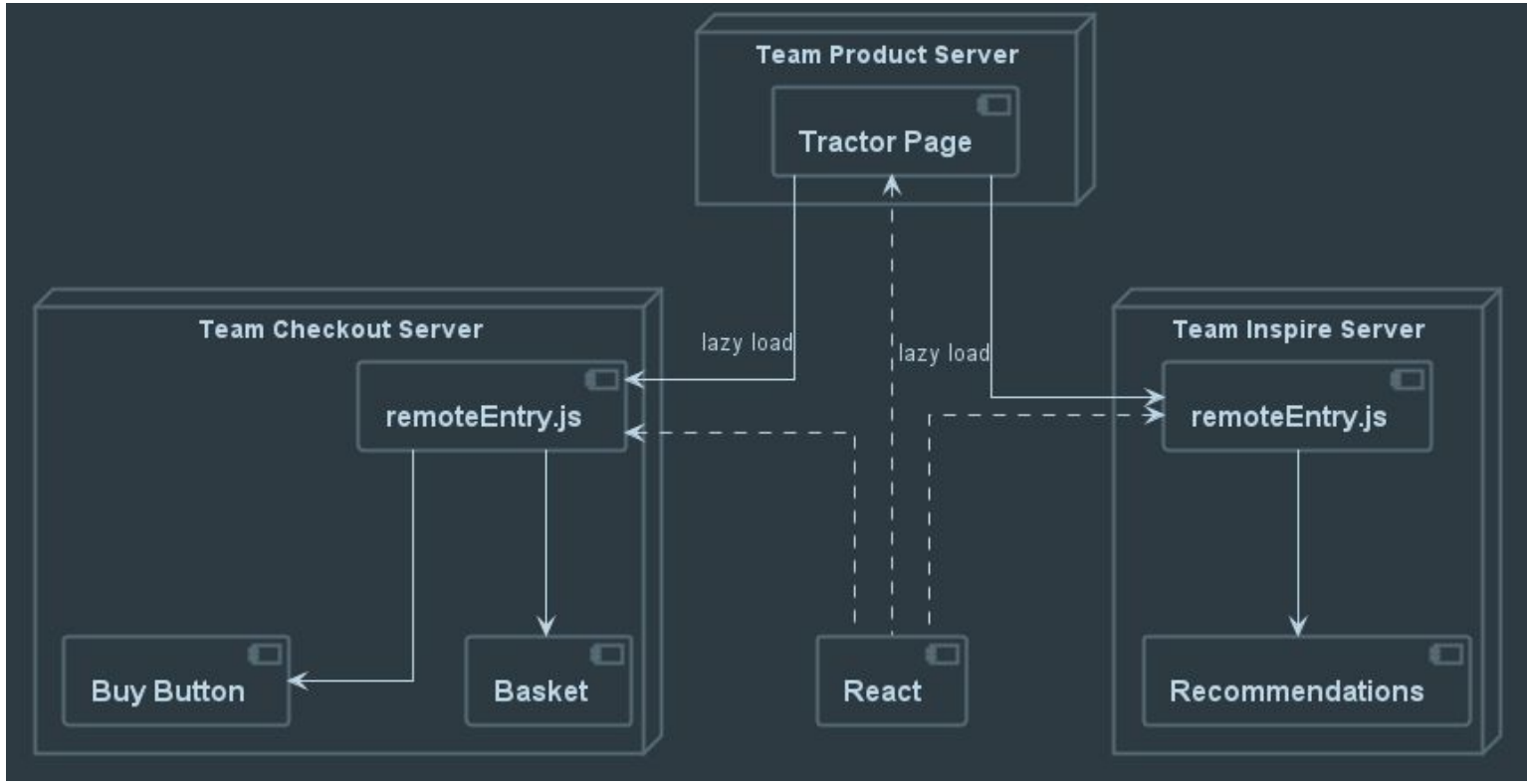
- Zack Johnson, Creator of Module Federation

Consists of:

- **Hosts:** Webpack builds that are initialized during a page load
- **Remotes**: Webpack builds consumed by hosts in part or in whole

Builds containers used for accessing remote content and functionality.

Facilitates sharing common dependencies between builds.



NOTE: Lazy load in diagram is for ***remoteEntry.js*** files, *not* React module.

What does Module Federation Do?

Module Federation has two distinct functions:

1. ***REMOTES:*** For a build that is to be exposed as an independent functionality, it defines a container for bundling the feature(s) together and defines how they are exposed.
2. ***HOSTS:*** For a build that will consume remote modules, it defines how to locate and ingest the modules (by naming the URL for the remote bundle and the file that exposes the container)

From the perspective of Micro Frontend (MFE) Architecture, we have the MFEs themselves and a host application which wraps them to make a single, cohesive system.

Each MFE is built taking advantage of function (1) of Module Federation to package the capabilities of the application in a way that allows them to be served remotely.

The host is built by function (2) specifying the names of the remote modules to be incorporated and where to find them. Lazy-loading during runtime allows the modules to be retrieved when needed to further augment the host application.

How Is It Used?

Exposed Remote Module

```
plugins: [
    new ModuleFederationPlugin({
      name: team-inspire,
      filename: 'remoteEntry.js',
      exposes: {
        './Module':
'apps/team-inspire/src/app/remote-entry/RemoteInspiration
Entry.module.ts',
      },
```

Declare the Module Federation plugin and pass in configuration. The **exposes** property indicates items that will be available to other applications remotely.

Consuming Application

```
plugins: [
    new ModuleFederationPlugin({
      remotes: {
        inspire: 'http://localhost:4201/remoteEntry.js',
        checkout: 'http://localhost:4202/remoteEntry.js'
      },
```

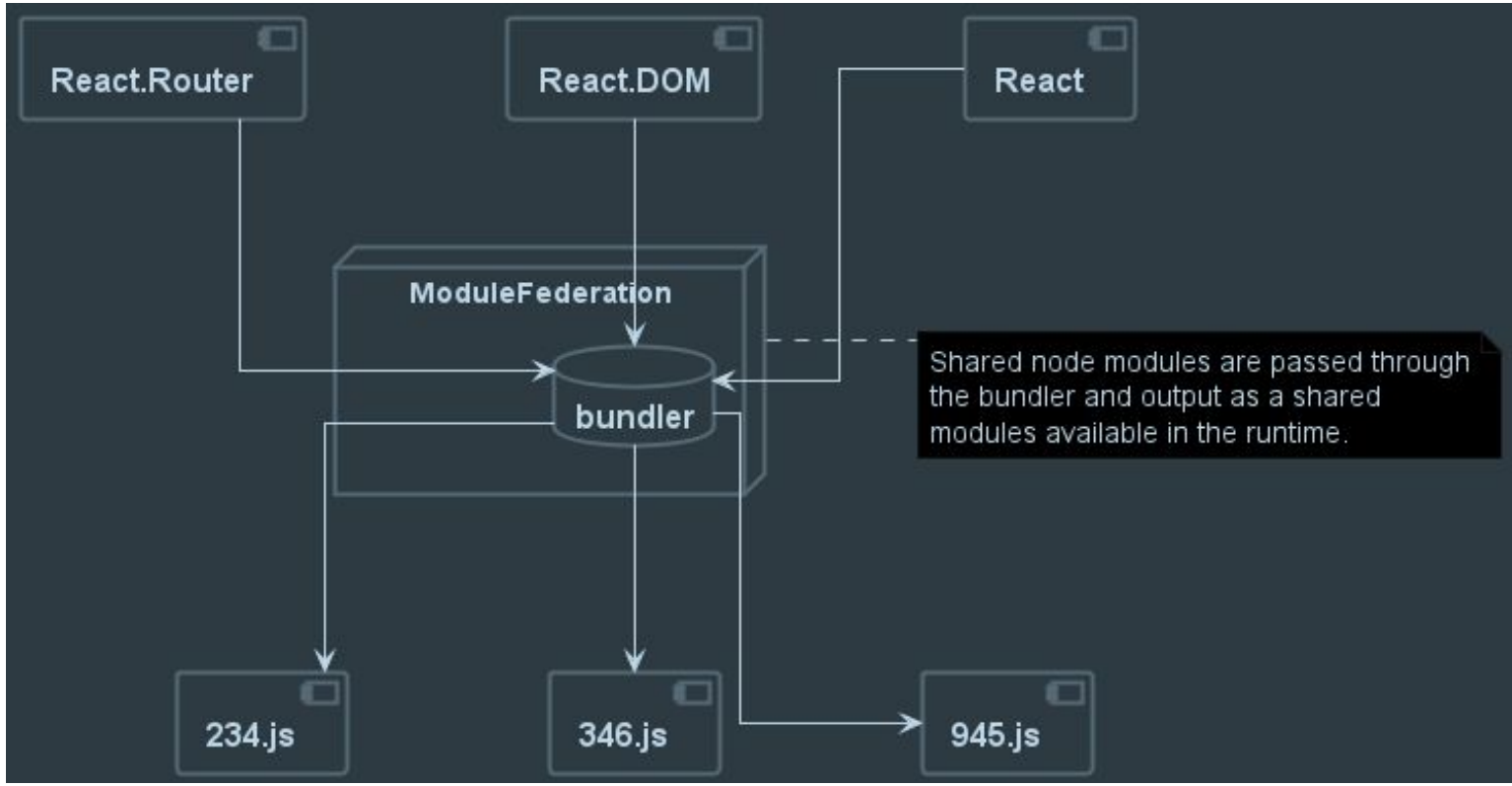
Declare the Module Federation plugin and pass in configuration. The **remotes** property indicates items that will be consumed from *outside* this application at runtime.

NOTE: Examples demonstrate usage in an Angular project

What About Supporting Libraries?

Module Federation allows for sharing of dependencies between containers. By configuring the webpack build to share dependencies, you can reduce overall payload size to the client and ensure required dependencies are loaded only once.

Each remote loaded will use the the host dependencies already present. If a new dependency is needed, the remote will download the necessary pieces that can also be shared among other remotes.



NOTE: Bundles names on output are for example only.

How Should I Divide My Application?

Breaking Monoliths Into MFEs

- This is HARD
- Start small, find portions of the application that have not or will not change often
  - Make logical “chunks” around this area that can be easily self-contained
  - Go SLOWLY
  - Ask yourself, “Is this really necessary?”
  - Have clear goals on WHY
- Look at existing teams and how their assigned areas are divided (if they are)
  - Ensure these areas have clear boundaries that would support isolation
  - Work towards clean, clear contracts, interfaces, etc to support the needed functionalities.

Designing for Micro Frontends

If working from scratch, have a clear understanding of the end-goal.

No code until boundaries are clearly understood.

Have a plan for reusable components and functionality!

Use Domain Driven Design to help clarify boundaries for isolation. This approach helps give clear indications of where the natural divisions in an application can be made.

Go SLOW

Start with a small, stable, easily testable feature.

Design the Micro front-ends so they can be tested and run independent of the orchestrator (shell, host) application to ensure all functionality is independently testable.

Dev Tools To Consider

Dev Tools To Assist In MFE Development

Build system, schematics and tooling

- Nx
- Nx Console
- VS Code



State Management

- NgRx
- Redux


NPM Scripts For *rapid* App Scaffolding

- **`create-nx-workspace`** (uses Nx build system)
- **`create-mf-app`** (creates bare-bones MFE application using Module Federation)

Recent Implementation Lessons Learned

General Lessons We'll Review

Migration is HARD	Converting a monolithic application to a collection of MFEs is possible, but extremely difficult. Planning is key, reasonable delivery timelines and a phased approach are a ***MUST***.
Use State Management	Due to the runtime nature of MFEs, state management is your friend. Once loaded into the client browser, the runtime memory space is available to each component, making authentication and data sharing a must for an efficient, performant user experience.
Mono-Repo Is Preferred	It is absolutely possible to manage micro-frontends in multiple repositories. However, it is infinitely easier to manage the inevitable shared component libraries when using a single mono-repository or a hybrid repository approach.

Plan a SLOW, Phased Migration

- Identify “low-hanging fruit”, simple aspects of the application that are mostly isolated already
- Select one for a pilot attempt and break it off as an independent MFE
- Work out the general build pipeline and library sharing strategy during the pilot phase
  - What does your host Webpack configuration look like?
  - How will independent components be built?
    - What’s the branching strategy for independent MFE releases?
  - Where will they be hosted?
  - Answer the infrastructure questions during this small phase
- Gradually migrate other aspects of the application, one domain at a time

OR

Design your application to be MFE *from the beginning*. This ensures you can implement a full-stack MFE strategy.

State Management With NgRx

NgRx allows multiple micro-frontend implementations to retrieve and share state information. Key among these is authentication.

- Azure B2C auth token pulled to NgRx store to enable authorization across MFEs
- User permissions managed from client-side store to reduce API calls (shared across MFEs)
- Guard implementations utilized store information to control site access without recurrent API calls

Be selective on store usage.

- Large amounts of temporal data should not be held in the store
- Design pagination with the above in mind, back-end or front-end? Where does the data reside during page view?
- Smaller “packets” of data that are reused across pages / MFEs are best stored in the state management store.

Mono- Versus Multi- Repositories

Proof of Concept utilized Nx workspace and a mono-repository. Customer added requirement that *every* MFE implementation be in it's own repository.

- Nx Workspaces built to inherently support single, large repository with all code.
- Git Submodules utilized to satisfy customer requirement and align with PoC structure
- Fully independent repositories would have been possible *however* shared code libraries would have been redundant or required packaging and added management

- Separate build pipelines were created for each MFE

Lesson:

- Discounting shared code libraries, each app could have been fully in its own repository utilizing Webpack Module Federation as the driving force for remote module access.
- Hybrid repositories are not a bad thing. Git submodules still have a place in the development world.

Questions?

cognizant softvision | PROGRAMMERS´ WEEK 2022

Thank You!