JOURNEY OF MICRO FRONTEND

Let us go through the journey of "Breaking the Frontend into smaller pieces with micro service approach & build the Micro-Frontends around Business Domains.

Agenda

- Background
- Micro service to solve backend
- Why not Micro frontend to solve frontend ?
- Approaches & frameworks available(Iframe, Single SPA etc.)
- Trade-offs
- My two cents

SAMSUNG

Background

Web moves fast

“Show me your Frontend Stack… and I’ll tell your Project’s Age”

Micro service to solve backend

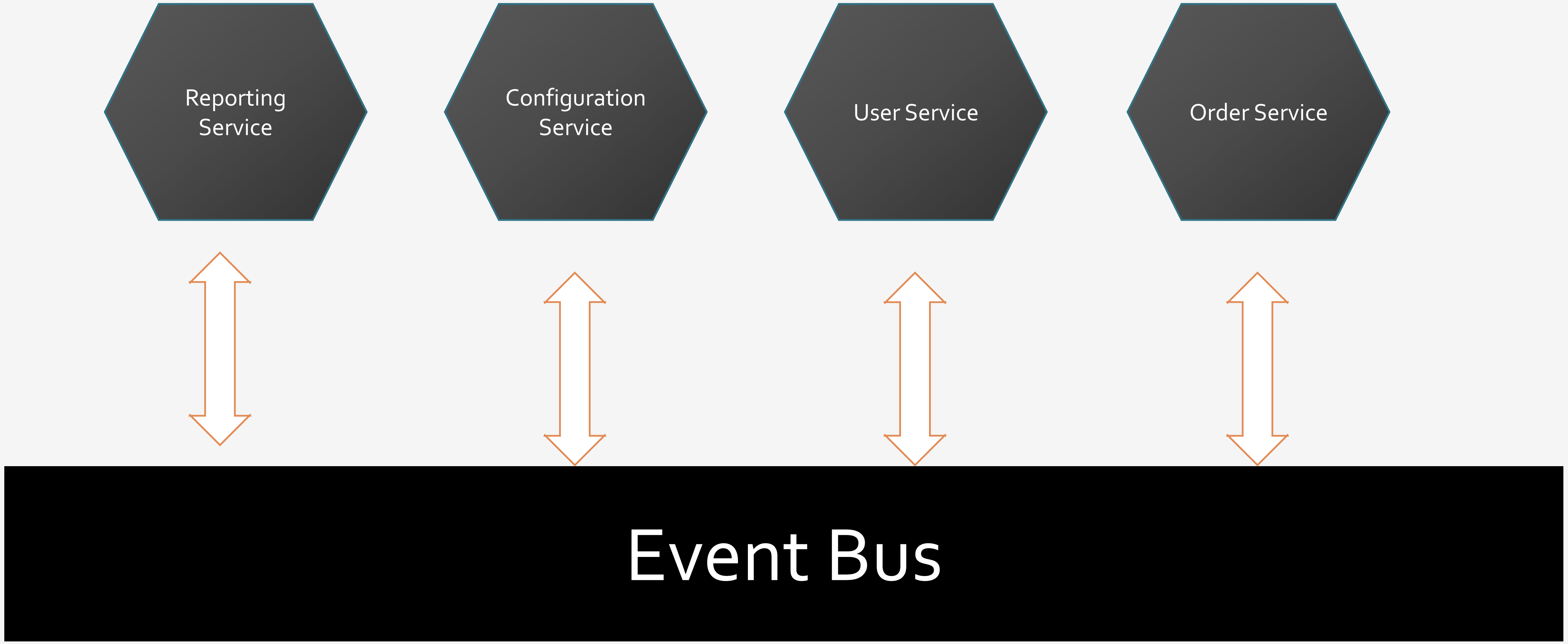
Martin Fowler and James Lewis

“Designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.”

The micro services architecture style is an approach for developing small services each running in its process. It enables the continuous delivery/deployment of large, complex applications.

- Highly maintainable
- Loosely coupled
- Better Independent deploy ability
- Improved fault isolation

SAMSUNG



Event bus allows pub-sub communication between components without requiring the components to explicitly register with one another

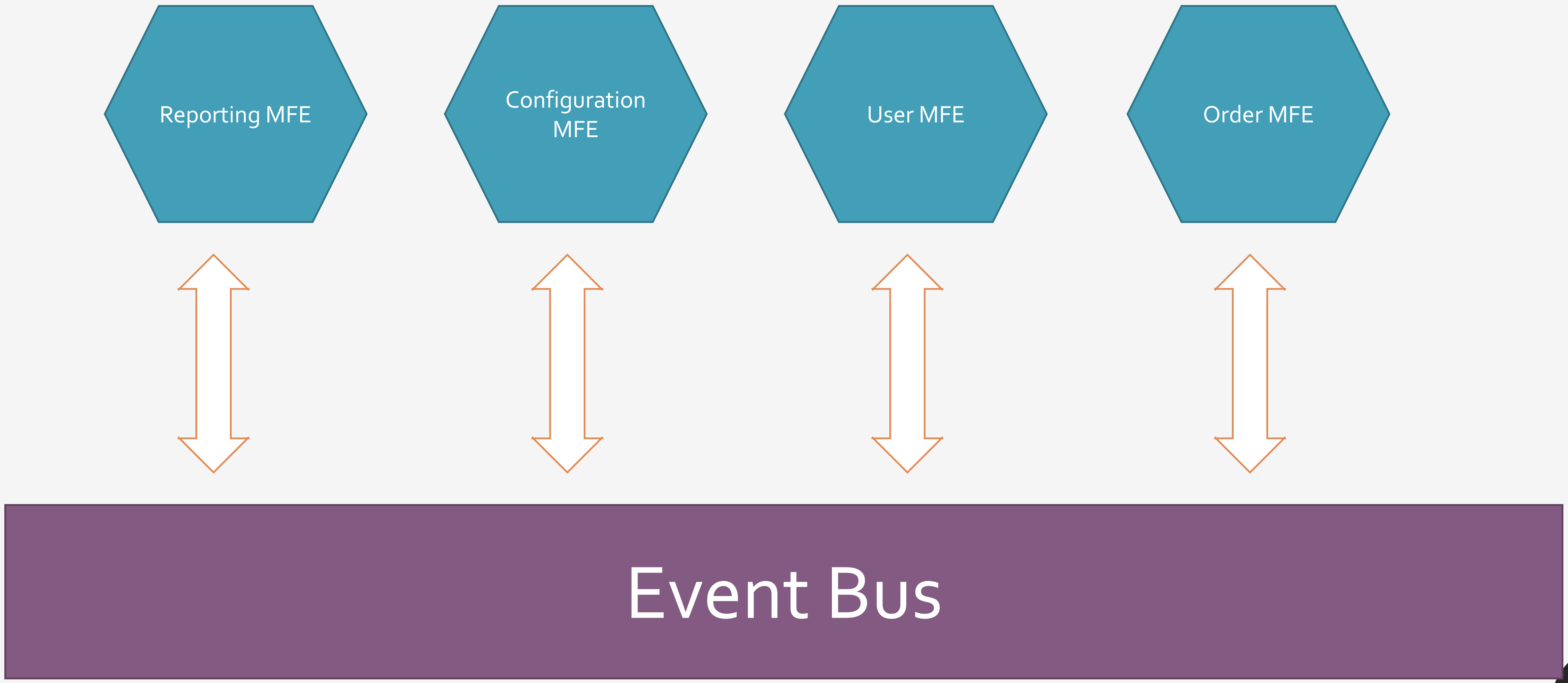
SAMSUNG

Why not Micro frontend to solve frontend ?

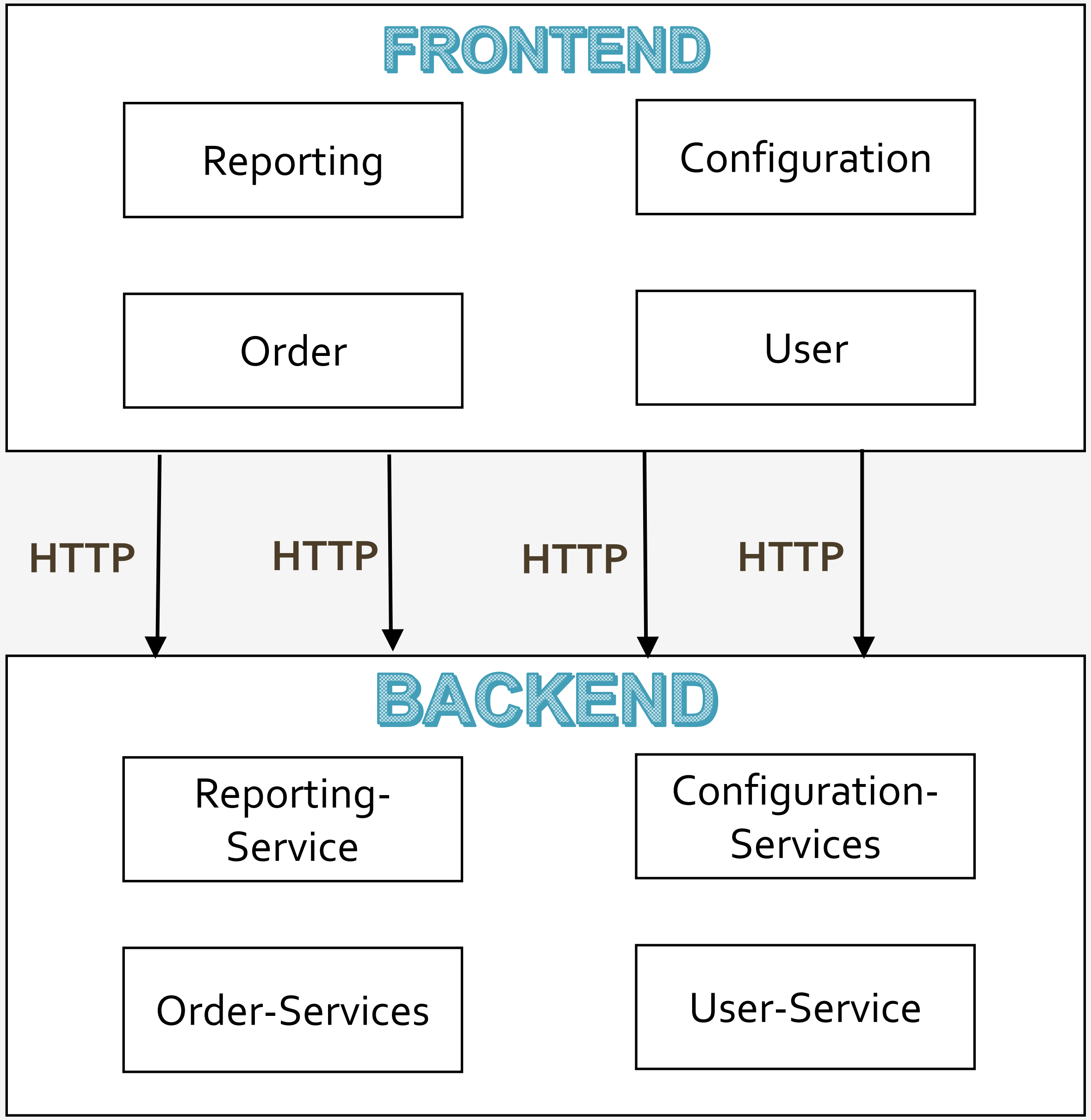
"An architectural style where independently deliverable frontend applications are composed into a greater whole"

Some of the principles of the Micro frontend architecture are:

- *The app is broken down into smaller micro apps bound by visual context.*
- *Each app is independent, it can fetch its own data and has its own encapsulated styling.*
- *These Micro Apps don't share a global state.*
- *Each team can work on their micro app independently and deploy them independently.*

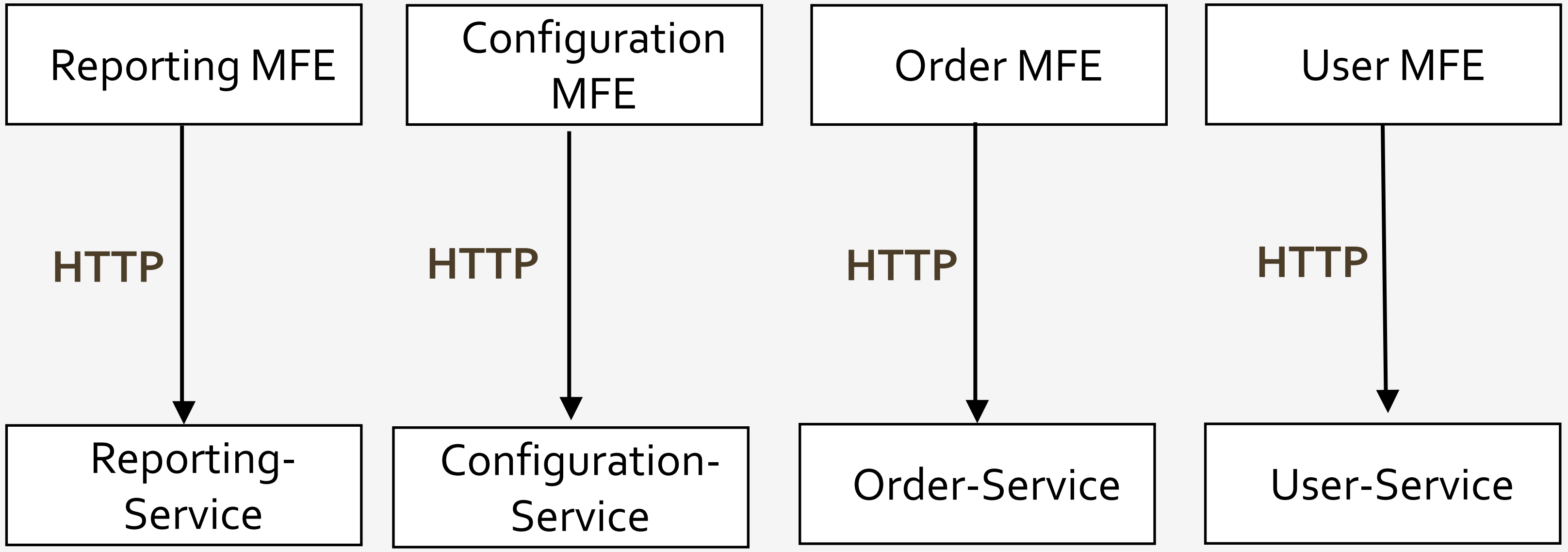


MFE -> Angular ,React, Vue with any combination

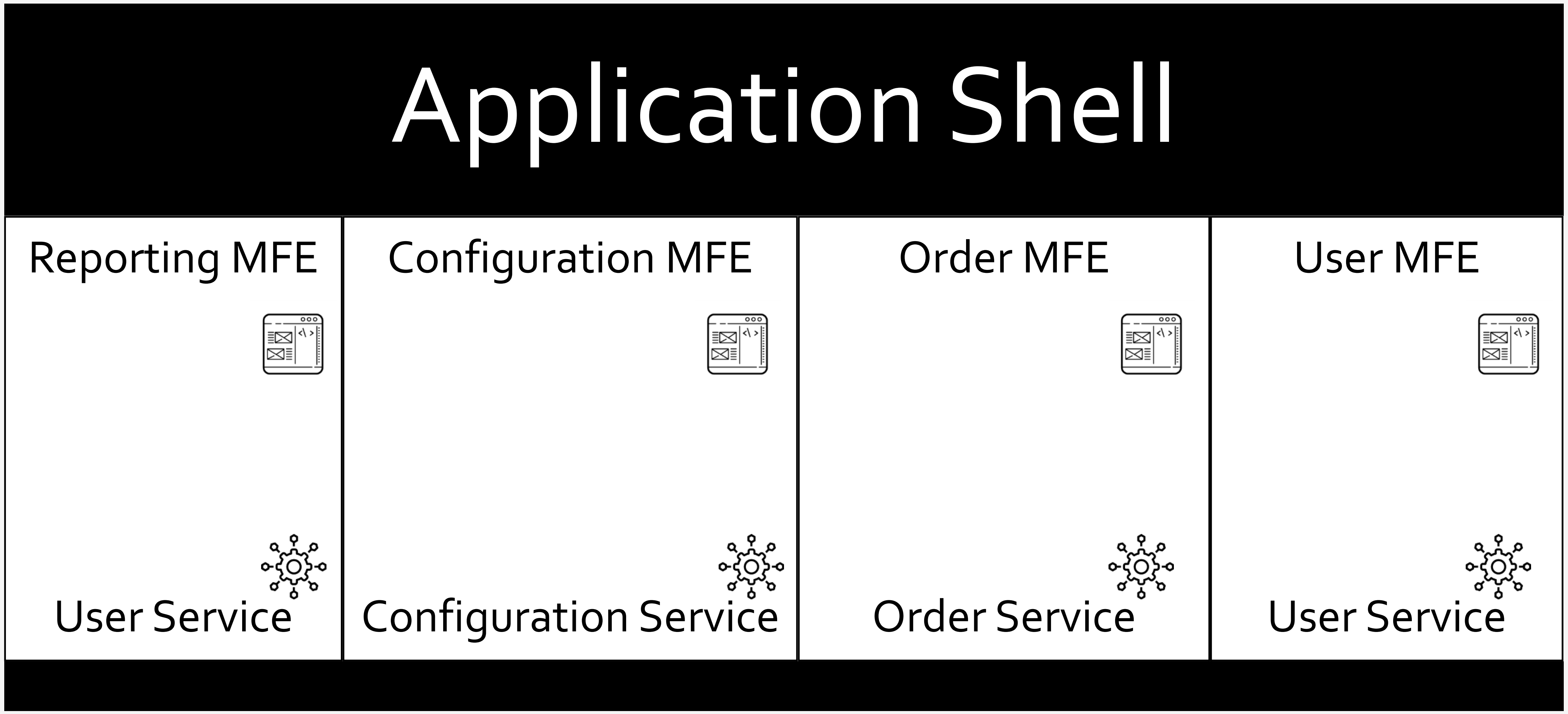


MONOLITH

MICRO SERVICES



MICRO (Services + Frontend)



FULLSTACK DEVELOPMENT TEAM

How to Split the Web Apps

Functionality on the pages

Pages

Sections

Approaches

Approaches	Development Costs	Maintenance Costs	Feasibility	Same Framework Requirements
Route Distribution	Low	Low	High	No
iFrame	Low	Low	High	No
Application Microservices	High	Low	Medium	No
Micro-Widget	High	Medium	Low	Yes
Micro-apps	Medium	Medium	High	Yes
Pure Web Components	High	Low	High	No
Web Components	High	Low	High	No

SAMSUNG

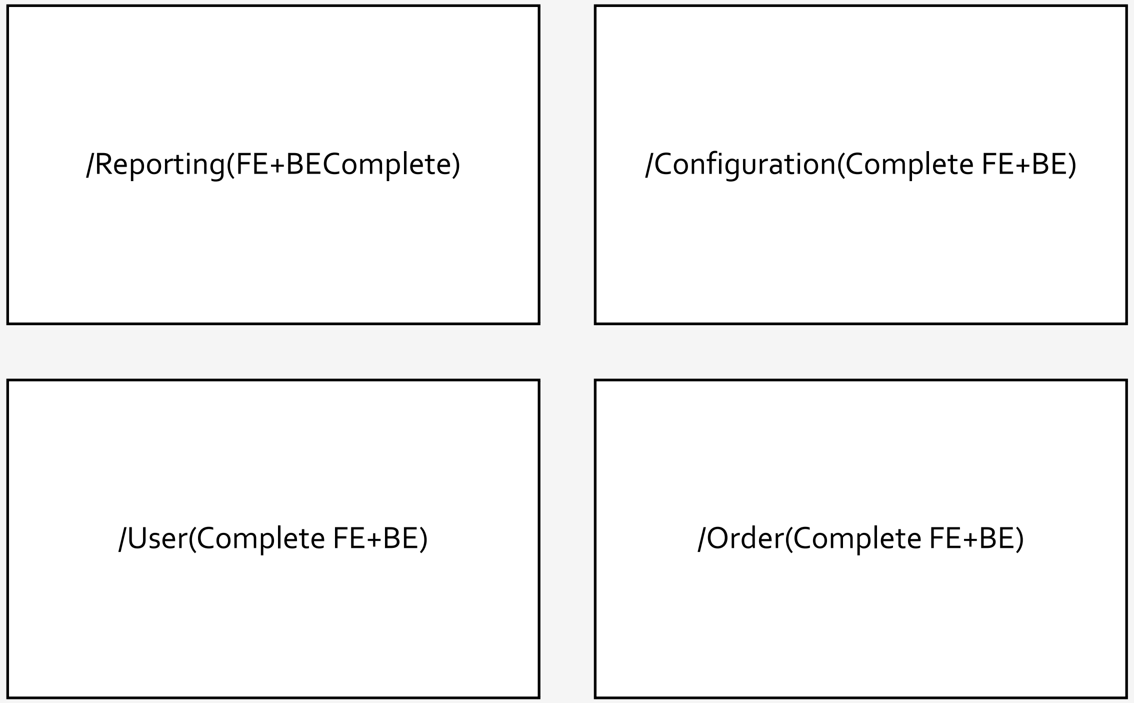
DIY Approach

Routing

Iframe

EventBus

Application Distribution Routing -> Route Distribution Application



Routing of the single front-end framework and the single-end back-end application are not much different: **Return the templates of different pages according to different routes**.

```
http {
  server {
   listen      80;
   server_name  www.application.com;
   location /reporting/ {
    proxy_pass url/reporting;
   }
   location /configuration/ {
    proxy_pass url/configuration;
   }
   location /user/ {
    proxy_pass url/user;
   }
   location /order/ {
    proxy_pass url/order;
   }
   location / {
    proxy_pass /;
   }
  }
 }}
 }
```

Create with iframe

Iframe is a very old technology that everyone feels ordinary, but it has always worked.

HTML Inline Framework Elements <iframe> represents a nested context being browsed that effectively embeds another HTML page into the current page

Point to consider:

1. **Load Mechanism**

How & when to load & Unload these independent application & what's the transition (animation etc.)

2. Communication Mechanism

Getting the Window object of the iFrame element through iframeEl.contentWindow is a much simpler approach instead of creating postmessageevent in each app & add listener.

Event Bus(Eev)

- A tiny, fast, zero-dependency event emitter for JavaScript.
- Less than 500 bytes minified + zipped
- Fast... see jsperf
- Zero dependencies
- Simple

https://github.com/chrisdavies/eev

Event bus

```
import Eev from 'eev'

const eventbus = new Eev()

eventbus.emit('event', { foo: ' Data' })

eventbus.on('event', function (context) {
    console.log('Received'+context.foo)  //received the Bar
})
```

Event bus

```
import Eev from 'eev'

window.eventBus = new Eev()

window.eventBus.emit('event', { foo: 'Bar' })

window.eventBus.on('event', function (context) {
    console.log('received' + context.foo) // received bar
})
```

Frameworks/ Library makes life easy

- Single-SPA
- Piral.io
- Luigi
- Mooa
- Frint
- Module Federation

Single-SPA

- https://github.com/CanopyTax/single-spa
- Use multiple frameworks on the same page without refreshing the page
- Write code using a new framework, without rewriting your existing app
- Lazy load code for improved initial load time

Single-SPA Library

- Code lives on the same server
- Everything is bundled and deployed at the same time
- Communication is done through:
  - Window object
  - Event bus
  - Shared state (Redux etc.)
  - Whatever you like & had worked for you…

```
import * as singleSpa from 'single-spa'

const reportingMFE = 'reportingApp'

const loadingFunction = () => import('./reportingApp/reportingApp.js')

const activityFunction = location => location.hash.startsWith('#/reporti
ngApp')

singleSpa.declareChildApplication(reportingMFE, loadingFunction, activit
yFunction)
singleSpa.start()
```

Single Spa App

```
export const bootstrap(props) => {}
export const mount (props) => {}
export const unmount (props) => {}
```

Bootstrap

```
let ContainerEl

//make a container for your app

export const bootstrap = (props) => {
    return Promise
        .resolve()
        .then(() => {
            ContainerEl = document.createElement('div')
            ContainerEl.id = 'reportingApp'
            document.body.appendChild(ContainerEl)
        })

}
```

Mount

```
//Mount the framework code

export const mount = (props) => {
    return Promise
        .resolve()
        .then(() => {
            ContainerEl.textContent = 'reportingApp is mounted'
        })
}
```

Unmount

```
//Unmount the framework code from the dom tree

export const unmount = (props) => {
    return Promise
        .resolve()
        .then(() => {
            ContainerEl.textContent = ''
        })
}
```

SAMSUNG

Module federation

It allows a JavaScript application to dynamically load code from another application, with the outstanding feature as if an application is consuming federated modules and does not have dependency needed by the federated code then webpack will download the missing dependency from that federated build origin. Amazing stuff

Module federation: loads the code from another application

A host: a webpack build that is initialized first during a page load (when the onLoad event is triggered)

A remote: another Webpack build, where part of it is being consumed by a "**host**"

Bidirectional-hosts: when a bundle or Webpack build can work as a host or as a remote. Either consuming other applications or being consumed

Approach for Module Federation

1. Webpack setup for host app

2. Create the useDynamicFederation() hook to lazzy load the sub-app from different remotes

3. Webpack setup for remote app

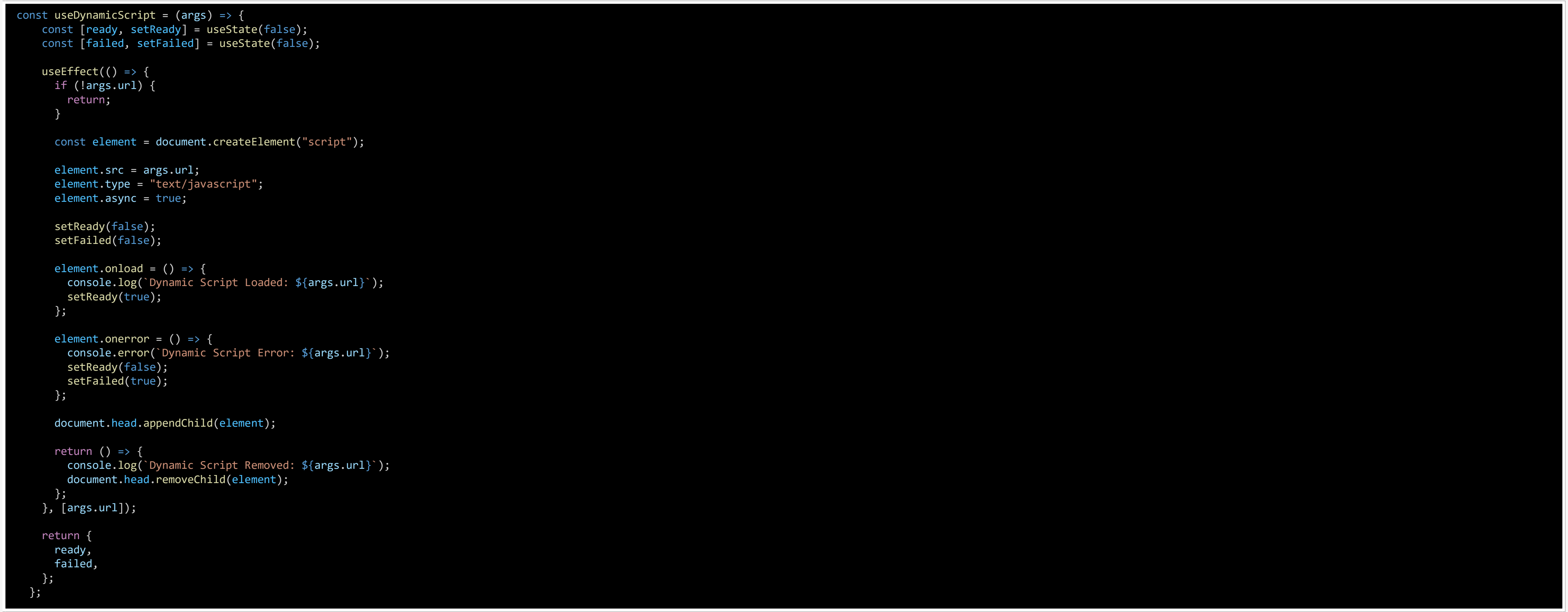
4. Create component in remote app that is going to load in host app remotely

webpack for host app

```
{
    plugins: [
       new ModuleFederationPlugin({
         name: "reportingApp",
         library: { type: "var", name: "reportingApp" },
                  shared: ["react", "react-dom"],
       }),
       new HtmlWebpackPlugin({
         template: "./public/index.html",
       }),
     ],
   }
```

SAMSUNG

Create the useDynamicFederation()



```
const useDynamicScript = (args) => {
    const [ready, setReady] = useState(false);
    const [failed, setFailed] = useState(false);

    useEffect(() => {
      if (!args.url) {
        return;
      }

      const element = document.createElement("script");

      element.src = args.url;
      element.type = "text/javascript";
      element.async = true;

      setReady(false);
      setFailed(false);

      element.onload = () => {
        console.log(`Dynamic Script Loaded: ${args.url}`);
        setReady(true);
      };

      element.onerror = () => {
        console.error(`Dynamic Script Error: ${args.url}`);
        setReady(false);
        setFailed(true);
      };

      document.head.appendChild(element);

      return () => {
        console.log(`Dynamic Script Removed: ${args.url}`);
        document.head.removeChild(element);
      };
    }, [args.url]);

    return {
      ready,
      failed,
    };
  };
```

Dynamic script function will take the url of the remote sub-app and inject that url to the script element of existing HTML at runtime so that script will load the remote sub-app dynamically.

SAMSUNG

Webpack setup for remote app

```
function loadComponent(scope, module) {
    return async () => {
      // Initializes the share scope. This fills it with known provided modul
es from this build and all remotes
      await __webpack_init_sharing__("default");
      const container = window[scope]; // or get the container somewhere else
      // Initialize the container, it may provide shared modules
      await container.init(__webpack_share_scopes__.default);
      const factory = await window[scope].get(module);
      const Module = factory();
      return Module;
    };
  }
```

SAMSUNG

Trade-offs

Bundle Size

Performance , a lot of code to download

UX consistency

Policy for creating MFE

Automated CI/CD pipelines with e2e testing

My two cents

- Don’t use this , if you have small team & simple app
- Don’t compromise user experience over developer experience
- Shell component as thin as possible.
- Shared component across MFE
- Use a BFF (backend for frontend)
- Rely on the Web API of modern browser
- Logging is important
- Share knowledge between teams

About Me

- Senior Chief Engineer @Samsung Research Bangalore
- @BglrSatyam
- satyam-pandey-339268176

SAMSUNG