WHITE PAPER

MICRO FRONTEND LIFECYCLE MANAGEMENT

Abstract

Micro Frontend (MFE) approach is a way to build more scalable and maintainable frontend applications, but it's important to have a clear strategy, a well-defined process, and good communication between the different teams to ensure a successful implementation. The future of MFEs is likely to see continued growth and adoption as more and more companies recognize the benefits of breaking down monolithic front-end applications into smaller, more manageable components. Managing the Lifecycle of MFE will play a pivotal role in successful adoption.

Infosys®
Navigate your next

Introduction

MFEs is an architecture pattern that allows you to split a monolithic frontend application into multiple smaller, self-contained applications that can run independently. Each MFE is responsible for a specific area of functionality or feature and communicates with the other MFEs via APIs or message passing. This approach allows for easier development, testing, and deployment of the individual MFEs, as well as greater scalability and flexibility for the overall application. It also allows for different teams to work on different MFEs concurrently and independently. With MFE, applications are split into granular units based on domain/feature and are owned by independent teams. The MFE Lifecycle with a strong DevOps practice ensures all stages from implementation to delivery and upgrades along with rollback happens in a seamless way.

Key focus of MFE Architecture is.

- Reusability across multiple platforms
- Better Separation of Concern
- Easier to test and deploy.
- Quick and easy to build new features.
- Easy to upgrade/roll out new features (micro)
- Resilient user event response
- Independent from Technology choices
- increased scalability, maintainability, and flexibility
- Micro releases (Micro features releases)

Integration:

MFEs can be integrated at build time or runtime, each approach has its own set of benefits and trade-offs..

Build Time Integration

Build time integration for MFEs refers to the process of integrating multiple MFEs together during the build process, rather than at runtime. This approach can have several benefits, including.

Improved performance: Build time integration can reduce the number of requests and the amount of runtime code needed to integrate the MFEs, which can improve the performance of the application.

Simplified Integration: Build time integration can simplify the integration process, as the MFEs are already integrated together when they are built, and there is no need to handle the integration at runtime.

Reduced complexity: No need to handle the integration of the MFEs at runtime.

Improved security: By integrating the MFEs at build time, it is possible to perform security checks and validations at the build time, which can improve the overall security of the application.

Runtime Integration

Runtime integration for MFEs refers to the process of integrating multiple MFEs together at runtime, rather than during the build process. This approach has its own benefits and considerations, such as:

Dynamic Integration: Runtime integration allows for dynamic integration of MFEs, meaning that MFEs can be added, removed, or updated at runtime, which can make the application more flexible.

Decoupled Communication: MFEs can communicate with each other through APIs, events, or message passing, which allows for decoupled communication between MFEs.

Easy scaling: With runtime integration, it's easy to scale up or down the number of MFEs based on the traffic, which can help to improve the performance of the application.

Better isolation: With runtime integration, MFEs can be isolated from each other, which can help to improve security and stability of the application.

Dynamic loading: With runtime integration, MFEs can be loaded dynamically, which can improve the performance of the application by reducing the initial load time.

Reduced complexity: No need to handle the integration of the MFEs at runtime.

Improved security: By integrating the MFEs at build time, it is possible to perform security checks and validations at the build time, which can improve the overall security of the application.

In the Build Time Integration Approach, in order to deliver a MFE, the parent shell application needs to be built everytime. Even though there could be a refinement to intelligently identify the MFE that has been modified and do a conditional build, the root application needs to be deployed for every change in any of the MFEs.

Whereas in the Runtime Integration approach, every Module (MFE) is hosted separately, and its location is described by Remote Entry endpoint and load requested module "on-demand". This will allow each development team to focus on their own application, giving them more autonomy over their application's direction and release schedule.

MFE Frameworks, Lifecycle and Limitations

An MFE framework is a set of tools and libraries that enable the development, deployment, and integration of MFEs within a larger front-end application. Some popular MFE frameworks are single-spa, Webpack Module Federation, Lerna, Bit.dev, Open-Components, Qiakun, Luigi, Piral.

These frameworks and tools can help to simplify the process of developing, deploying, and integrating MFEs, making it easier to create scalable and maintainable frontend applications.

The lifecycle of an MFE typically involves several stages, including:

Development: Each MFE is developed and tested independently, using the appropriate front-end framework and technologies.

Deployment: MFEs are deployed independently and as part of a larger front-end application.

Integration: The MFEs are integrated with the main application, and communication between them is established through APIs or other means.

Maintenance: MFEs are updated and maintained independently, allowing for faster and more efficient development and deployment.

Retirement: If an MFE is no longer needed or has reached the end of its life, it can be retired without affecting the rest of the application.

Limitations of Current MFEs:

Despite the benefits that exist in Runtime Integration, every time there is a change, a new deployable artifact is created, and the existing version is replaced. This is the typical approach that is followed for Microservices as well. While this seems to work like a charm, there is an imperative issue in this approach for MFEs.

While doing root cause analysis for any bugs, the development team would always want to know the version of the artifact that is deployed. In a Microservices world, this can be figured out using the image tag/version of the running container. This approach will not work out for MFE as there is always a single bundle that is deployed and consumed, and the image/tag will not change unless there is a change in the base(shell) application.

Uglification and Minification of bundles exponentially increase the complexity in identifying the version that is deployed.

In a nutshell, to enhance the Lifecyle of MFE, the following capabilities are required

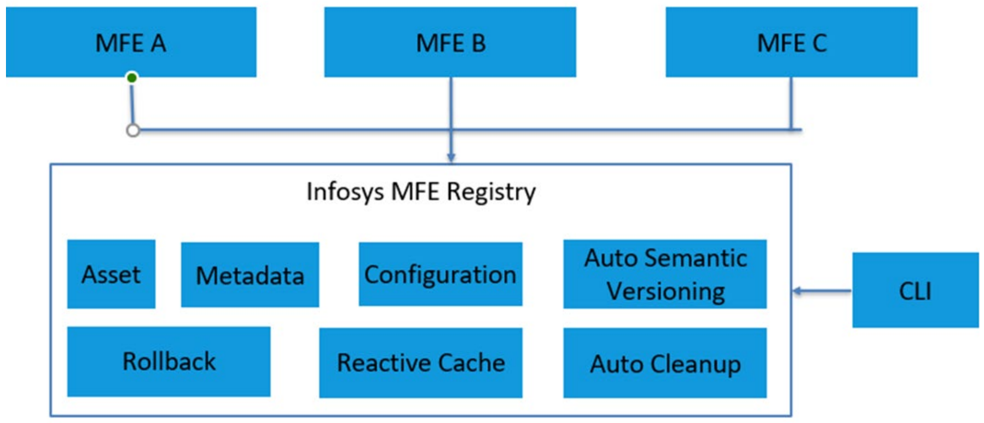
- Ability to build, test and deploy each MFE independently.
- Changes in any MFE should not require the base application to be redeployed.
- Avenue for easier root cause analysis in case of any issues
- Seamless way to rollback
- Support for Multiple Environments
- Capability to host per environment configuration details to enable instant consumption.
- Ability to use cached version of MFE and invalidation.

Infosys MFE Registry

Infosys MFE Registry is an MFE solution that allows developers to create, share, and reuse MFEs across different applications and teams. The goal of is to provide a simple and efficient way to build and manage MFEs, making it easy for teams to create and maintain scalable and modular frontend applications. It is built on the concept of MFE and it's a solution that can help teams to manage, share and discover MFEs across different projects and teams, which can make it easier to create and maintain scalable and modular frontend applications.

Infosys MFE Registry is built with all the capabilities listed above.

- Offers high-performance, Cloud compatible Object storage.
- Kubernetes powered Platform and can seamless be made available on every public cloud, every Kubernetes distribution, the private cloud, and the edge.
- Native Semantic Versioning supports that aims to communicate the level of compatibility between releases immediately.
- Provides a simple way to automate the build, test, and deployment process for MFEs, which makes it easier for teams to develop and deploy MFEs.
- Out of the box support for handling Multiple Environments for each MFE
- Inbuilt Caching, Compression, Purging and Scavenging capabilities.
- Greater Interoperability ensuring that MFEs can work together seamlessly, regardless of the framework or library used to develop them.
- Eventually as serverless technology becomes more prevalent, it is likely that MFEs will be developed and deployed using serverless architecture, which can provide cost savings and scalability.



MFE Registry Components:

Asset: Asset depicts the artifacts of an MFE. It could be html, JavaScript, stylesheets etc. Infosys MFE registry stores each MFE in separate structures and has an inbuilt mechanism to group the same MFE with different versions.

Metadata: Infosys MFE registry provides a unique capability to provide metadata for each MFE. These metadata are automatically generated by the MFE registry ranging from environment to version and other miscellaneous details.

Configuration: Configuration is an important component of MFE registry. For every environment, the configuration required for the MFE could be different. For instance, the backend URLs could differ based on the environment. Configuration provides the capability to support multiple environments for a single MFE. A better approach would be to supply a config file and depending on the environment and the MFE upon loading will access the configuration file and use the values specified in the configuration file. Infosys MFE Registry natively supports this approach and all the configuration changes for supporting different environments can be handled with ease.

Auto Semantic Versioning: Every time an artifact is uploaded to Infosys MFE registry, it auto generates the version adhering to Semantic Versioning scheme. This plays a pivotal role in identifying the latest version of an MFE and rendering all versions of a single MFE. Adopting Semantic versioning can increase the maintainability and scalability of the project, as it allows developers to better understand the impact of updates and make informed decisions about when to update their dependencies.

Rollback: Rollback provides a capability to remove a version of an artifact based on the criteria of the development team

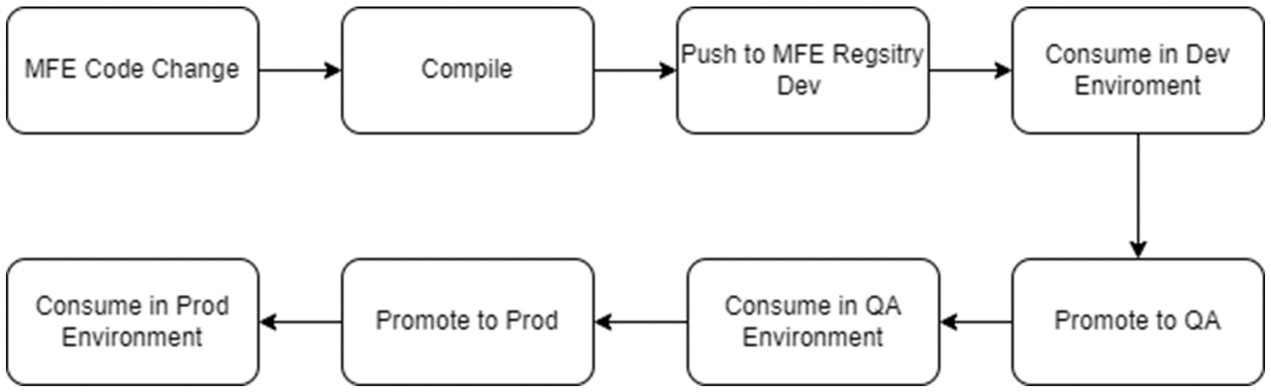
Reactive Cache: Reactive Cache is a unique capability which ensures if an artifact of an MFE has not changed, then it notifies the client, thus ensuring repeated download of the artifact doesn't happen. This plays a huge role in improving the Performance while rendering and savings the network bandwidth to a huge extent.

Auto Cleanup: Infosys MFE Registry provides the ability to clean up old Artifacts on a periodic basis. This is a configurable option. This ensures the last 3 versions of every artifact per environment is maintained. This capability is provided so that the disk space is optimally consumed.

MFE Lifecyle using Infosys MFE Registry

Below flow diagram summarizes how a micro-frontend journey will look like when Infosys MFE registry is used for MFE hosting and lifecycle management.

Raw flow


MFE Code Change
Compile
Push to MFE Regsitry Dev
Consume in Dev Enviroment
Promote to QA
Consume in QA Environment
Promote to Prod
Consume in Prod Environment

Using CI/CD setup:

MFEs and DevOps go hand in hand, as the MFE architecture requires a different approach to deployment, testing and maintenance. Some key considerations for MFE DevOps include:

Automation: Automating the build, test, and deployment process is essential when working with MFEs, as it allows for faster and more efficient development and deployment.

Continuous Integration and Continuous Deployment (CI/CD): Implementing a CI/CD pipeline can help to ensure that MFEs are always up-to-date and working properly.

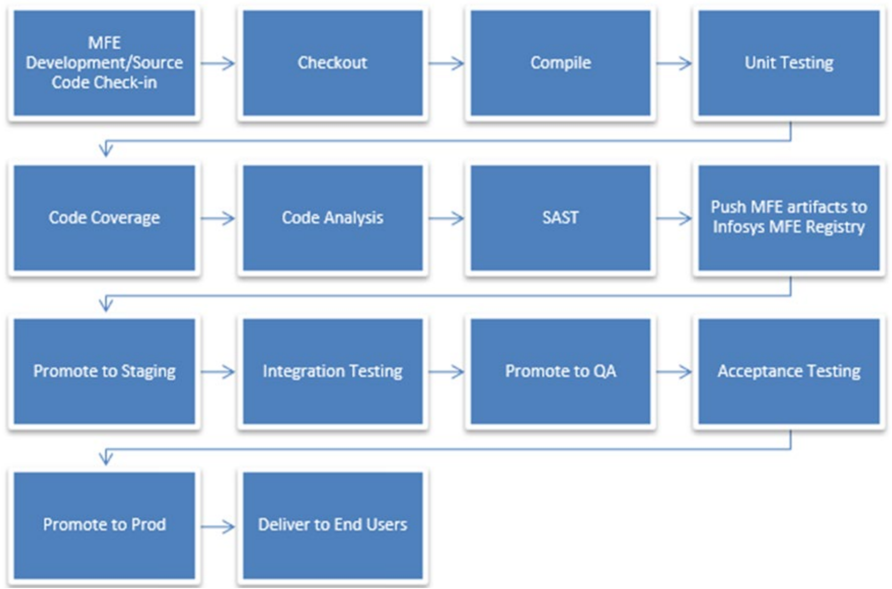
Monitor: The MFE is monitored to ensure that it is functioning correctly and to detect any issues that may arise.

Rollback: A rollback plan should be in place to revert to a previous version of the MFE in case of any issues.

Promotion: A foolproof Promotion strategy should be in place with an approval mechanism that ensures the process is consistent and reliable. The promotion process should also include testing and validations of the MFE in different environments, to ensure that it works correctly in each environment. This can include testing for compatibility, performance, and security.

Deployment: MFEs allow for easy experimentation and recommendation is to use at least one of A/B testing and canary releases.

Overall, the DevOps approach for MFEs is focused on automating the development and deployment process, and on ensuring that MFEs are always working properly, so teams can focus on developing new features and improving the user experience.



MFE Delivery – Platform Comparison:

As more and more teams have onboarded into the MFE journey, it is evident that to deliver the MFE to client, a paradigm shift is required from serving it as a static webapp to an API driven sophisticated endpoint. This will seamlessly enable Client-Side Rendering (CSR) or Server-Side Rendering (SSR) of MFE.

The following table gives a comparison of popular platforms for delivering MFE by features

MFE Delivery/ loading approaches	Bit.Dev	Open Components	Infosys MFE Registry
Integration	Build Time	Run Time	Run Time
CLI (create, build, publish)	Yes	Yes	Yes
Client-Side Wrapper	Yes	Yes	No
Environment Specific Support	No	No	Yes
Rollback	No	No	Yes
Scavenging (Auto Cleanup)	No	No	Yes
Reactive Consumer Cache (Request Component by Weak Version e.g.: 1.x.x)	No	Yes	Yes

Conclusion:

As enterprises move from legacy monolithic frontend applications to adopt an MFE framework, there are new challenges that arise. It is essential to have the right strategy and tools to help with these challenges. We highlighted an approach that enables organizations to be confident in their move to modernize their business operations.

Today, the MFE lifecycle is being implemented in more organizations to optimize the entire MFE development/ delivery process. It takes many steps for an MFE to go from initial idea to market to enable collaboration across teams. The major goal of the MFE Lifecycle management is to keep things running smoothly and maximize efficiency.

Infosys MFE Registry allows fast-moving teams to easily build and deploy front-end components. It abstracts the complexities and leaves teams with a very structured and systematic way to develop and deliver MFEs. It is also a battle tested solution currently used to deliver micro frontends at scale and enabling painless MFE delivery.

Authors

Chidambaram GS
Senior Technologist

Jaskirat Singh Sodhi
Senior Technologist

Mentor

Priyapravas Avasti
AVP - Senior Principal Technology Architect

References:

- https://github.com/topics/micro-frontend
- https://opencomponents.github.io/
- https://www.thoughtworks.com/en-in/radar/techniques/micro-frontends