`What are Micro Frontends

Mr. Santhosh Krishnamurthy
Application Developer, SAP Ariba, Bangalore,
India santhoshks2312@gmail.com

Monolithic Architecture:

The word 'Monolith' means interweaving all aspects into a single one. It describes a single-tiered software application in which different components, services etc. are combined into a single program from a single platform.

The components, services etc. can be:

- Authorization and Authentication which are responsible for allowing access to the system and confirming the user's identity within the system.
- Presentation and UI which are responsible for handling HTTP/ SOAP requests and responding via HTML, or JSON/ XML etc.
- Business Logic which is responsible for application logic.
- Integration, responsible for integration with other Application services via Messaging, REST API etc.
- Notification, responsible for alerting application owners via email, messages etc.

Drawbacks of Monolithic Architecture:

Though the Architecture is being followed since decades, and has benefits like simple to develop, test & deploy, with easy scaling, we have major concerns when the application becomes complex. Some of the major drawbacks include the following:

1. Maintenance becomes strenous, as the application grows huge and complex to understand entirely, it is challenging to make changes fast and more accurate.
2. The entire application has to be redeployed on each simple update.
3. The Size of the application can increase in compile time and slow down the startup time.
4. Reliability – Bug in any Module can potentially bring down the entire process or the instance of the application.
5. Though it seems easy in the initial stages, the monolithic applications have difficulty in adopting new technologies, since the changes in the frameworks may affect the entire application.
6. Monolithic applications can also be challenging to scale when different modules have conflicting resource requirements

Microservice Architecture:

What are Microservices?

Microservices is one of the newer concepts and a variant of a Service Oriented Architecture (SOA). Although SOA has been there for almost two decades while the Microservices came into existence in 2012.

The idea is to have small autonomous services to work together to build a large complex application, it mainly focusses on building individual sub-domains and small services making them easier to maintain, and promotes independently deployable pieces, thus ensuring the internal changes on one service do not affect or require the redeployment of other services.

Benefits of Microservice architecture:

- Since the entire application is decoupled into smaller services, it enables the continuous delivery and deployment of large complex applications.
- Improved and better testability because of the smaller services and faster.
- It enables you to organize the development effort around multiple teams. Each team is responsible for one or more single service. Each team can develop, deploy and scale their services independently of all of the other teams
- Improved fault isolation. For example, if there is a memory leak in one service then only that service is affected. The other services continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- The microservices architecture allows each team to decide the technology and infrastructure that works best for them, which may be completely different from other microservices that it interacts with for the very same product.

Monolithic vs Microservices architecture

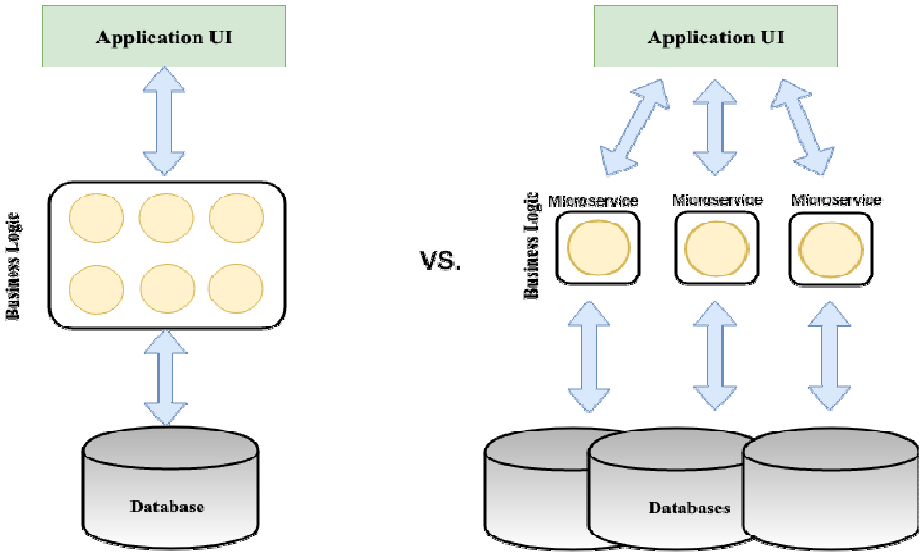

Figure 10.Monolith vs Microservice Architecture.

Key principles for building Microservices:

- **Modelled around business domain**
  Before beginning any refactoring, focus on the business domain and identify individual sub-domains and build services. Domain Driven Design would be the best way to start with.
- **Culture of automation**
  Provisioning a new machine, operating system and service should be easy and reliable. Automation testing and continuous delivery are very critical as well so as to deploy/ release your software frequently.
- **Hide implementations**
  Each service should have its own database and if shared information is needed from other services, leverage Service endpoints designed for the specific sub domain to extract what is expected.

  In a case, while a monolithic application to microservice structure, its often considered the easiest to tease part application level code while leaving the shared underlying database as is and this shared database continues to serve as a source of coupling between the independent services far greater than the decoupling achieved by spinning off the application level services.
- **Decentralize and deploy independently**
  Focusing on the autonomy i.e. giving freedom as much as possible to do the task in hand, self-service, shared governance and avoid complex messaging are important.

  Keep cyclic dependencies and deploy all of them together then fix dependency between services and then continue to create new services.

  In such an environment of independent deployments, when you make changes the consumer service has expectations about not facing challenges when you deploy new changes. It's good to have co-existing endpoints during upgrade of a service, which would continue to support existing version for a limited time period so that the dependent can migrate without holding the entire system hostage.
- **Isolate failures and Highly observable**
  The microservice architecture wouldn't explicitly make your system stable, On the contrary, it makes the overall system more vulnerable to certain types of network and hardware related issues.
  There should be a mechanism to isolate failures and look for more ways to recover such as failover caching and retry logic.
  When data needs to be present for the user, which is fetched by multiple microservices it might be challenging because of a lot of performance issues, in this case it's recommended to use different search engine or caching methodologies. This can significantly reduce the pain of performance bottlenecks.

Micro Frontends

What is Micro Frontend?

The trend is to build a powerful and feature-rich web application which resides on top of a Microservice architecture. Over a period of time the front-end part of the application becomes huge and large, which is developed by a separate team and gets more difficult to maintain, this type of application is called Frontend Monolith.

Micro-Frontend is a Microservice approach to front-end web development. The idea behind Micro-Frontend is to decompose the web application into smaller units based on the screens representing domain-specific functionality instead of writing large monolithic front-end application.

Micro-Frontend application is a composition of features owned by different independent teams, where a team is cross-functional and has ability to develop end-to-end features, from the user interface to database. It gives the same level of flexibility, testability and velocity as of microservices.

Problems with Frontend Monolith

- The flexibility promised by microservices cannot be scaled across the teams i.e. the backend team cannot deliver business value without the frontend being updated
- There would be a classical overhead of a separate backend and front-end team, which would cause the entire front-end to be updated and re-tested for a change in the API of one of the services.
- In a Single Page Application, all the files would be bundled into one and rendered on the browser, this file size would be huge.
- As applications grow, so does the features that teams need to support. With multiple teams contributing to a monolithic application, development and release coordination is a tedious.
- Newer frameworks and libraries like Angular 2, React, Vue, etc offer considerable performance improvements and innovations on the front end space. However, the onerous task of upgrading a monolithic application and/or making it interoperate with these new frameworks and libraries often can't be done without compromising the ability to ship new features at existing release rates.

Monolithic front-end vs Micro-Frontend architecture

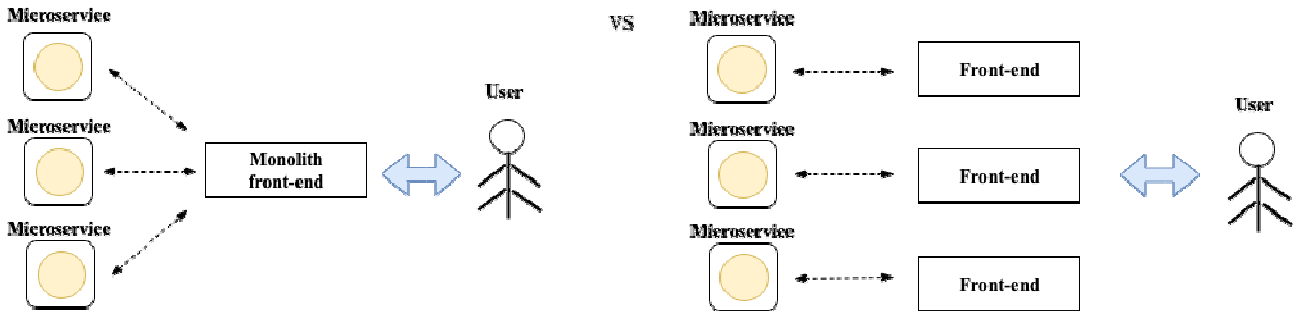

Figure 11. Monolith Frontend Vs Micro Front-end

Why Micro-Frontend matters

In the Modern era of cloud web applications, the front end is becoming bigger and huge and the backend is getting less important as most of the code is written on the front-end, because of which monolithic approach doesn't work for a larger web application. This asks for a need of tool for breaking it up into smaller modules that act independently. The solution to the problem is Micro Frontend.

Patterns followed while building micro frontends

Integration in the Browser

Web components provide a way to create fragments of Front-End imported into Web applications. Those fragments can be packaged into Microservices together with the back-end. Services built, completed with both logic and visual representation packed together. By using this approach, Front-End applications reduced to routing makes decisions involving which set of components displayed and orchestration of events between different web components.

Web Components

Web components allow the creation of reusable components imported into Web applications. These are like widgets imported into any Web page. These are currently supported in browsers such as Chrome, Opera and Firefox. If in case, the browser does not support web components natively, compatibility accomplished using JavaScript Polyfills

Web components consist of 4 main elements used separately or all together –

- Custom Elements

  This method allows to create custom HTML tags and elements with Custom Elements. Each Elements has its own CSS Styles and scripts. By creating own HTML tags it provides the flexibility to apply CSS Styles and add behaviours through scripts.

  In Web components, element lifecycle call-backs are available, which allow defining behaviours specific to the component developing.
- Shadow DOM – the DOM is the API

  Shadow DOM combines HTML, CSS and JavaScript inside a Web Component separated from the DOM of the main document when these are inside a component. This separation is similar to the one user while building API services and consumer of an API service does not know about its internals, the only thing that matters for a consumer are API requests. Such service does not have access to the outside world except to make requests to APIs of other services. Similar features observed in web components. Their internal behaviour not accessed outside, except when allowed by design nor affects the DOM document they reside in. The main method of communication between web components is by firing events.
- HTML imports

  For web components, HTML imports are the packaging mechanism. HTML imports tell DOM, the location of a Web Component. In the context of Microservices, import remote location of service contains the component to use.

  HTML imports is a method to reuse and include HTML documents via other HTML documents. Predefined components as HTML imports, where each of them include own styles and scripts, decide on the top level which HTML import present in DOM at the moment, and the imported document handles rest of things.
- HTML Templates

  The HTML template element holds client-side content not rendered when a page loaded. It's instantiated through JavaScript. It is a fragment of code used in the document.

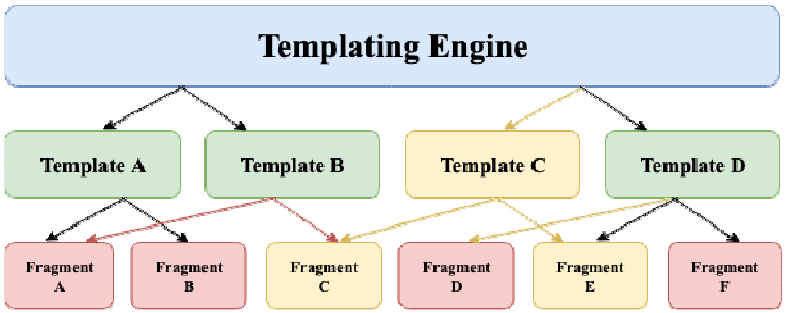

Figure 12 Templating Engine rendering fragments

Whenever user enters the website, the request is passed to Templating Engine, which, based on request URL, recognizes which template is expected by the user, loads it, and then populates it with content of corresponding micro frontends.

Benefits of Micro Frontends:

The key advantages of a micro frontends architecture over a monolith are:

- **Gives teams their release autonomy and time back**

  By breaking features from the monolith into separate micro frontends, teams enjoy increased autonomy and flexibility when releasing products/features. No longer are teams who aren't releasing required to stay up late on release calls trying to regression test other teams' changes in production. In other words, testing becomes simple as well as for every small change, you don't have to go and touch entire application.
- **Self-Independent**

  The individual development team can choose their own technology, not having to rely on the entire codebase reduces dependencies and scope, enabling teams to onboard and deliver quickly. This creates room for time spent innovating without fear of breaking other teams' features.

- **Highly Scalable & better performing web-app**

A loosely coupled architecture with established global standards makes it easier to add new features or spin up teams when needed. Since each app is fragmented into its own micro frontend, if a single feature (one micro frontend) on an enterprise app isn't loading fast, it won't affect the performance of the entire application. It also makes it possible for certain parts of a webpage to load faster, allowing users to interact with the page before all features are loaded or needed

Challenges of Micro Frontends Architecture

- **Rollout Strategy**

A major implementation consideration is deciding if one should convert their monolithic application to a micro frontend using a big bang or phased approach.

- **Governance**

The dependency needs to be managed properly. The collaboration becomes a challenge at a time. The multiple teams working on one product should be aligned and have a common understanding, though when there is change in multiple directions in terms of organizational and technology strategy.

- **Better Testing Strategy**

While a monolith creates an all-teams-on-hands approach to releases, micro frontends enable only contributing teams to take part in a given release. This approach requires that teams implement best-in-class regression testing practices to make sure broken features are not released to customers.

- **Legacy frontend frameworks**

The UX consistency is an important aspect. The user experience may become a challenge if the individual team go with their own direction hence there should be some common medium to ensure UX is not compromised.

As newer frameworks and libraries are being released at an exponential rate, the ability to create interoperable UI components between frameworks requires building reusable foundational elements which is time-consuming as well.

Conclusion :

As frontend codebases continue to get huge and more complex over the years, there is a growing need for more scalable architectures. We should be able to draw clear boundaries that establish the right levels of coupling and cohesion between technical and domain entities with the ability to scale software delivery across independent, autonomous teams.

While far from the only approach, there are many real-world cases where micro frontends deliver these benefits, and the technique is being gradually applied over time to legacy codebases as well as new ones. Whether micro frontends are the right approach for you and your organization or not, we can only hope that this will be part of a continuing trend where frontend engineering and architecture is treated with the seriousness that we know it deserves.

References:

- What are Micro Frontends - https://micro-frontends.org/
- Micro Frontends in Action - Michael Geers

About the author

Santhosh Krishnamurthy holds around 6 years of experience in building mobile & enterprise cloud applications for domains like Wholesale Banking, Air Cargo etc. He has worked in companies like Unisys. He is currently working as product engineer for Ariba Contracts Team in SAP Ariba. His interests are towards building mobile based application, ethical hacking. He loves playing AR based games, sports and movies.

Connect @ https://in.linkedin.com/in/ihappyk

A 29-year-old UK-based woman has said that Amazon's voice assistant Alexa went "rogue" and told her to make sure to stab herself in the heart "for the greater good". Danni Morritt had asked her Echo device about 'cardiac cycle' while studying when Alexa answered that beating of heart was body's "worst process" and bad for Earth as it caused overpopulation.

Pictures released by Russian President Vladimir Putin's press service appeared to show that he uses Windows XP operating system on his official computer. It is also installed on Putin's computer at his Novo-Ogaryovo residence near Moscow, reports said. Microsoft last provided security update for Windows XP in 2014 and warns computers running it might be more vulnerable to security risks.