Enterprise-Level Migration to Micro Frontends in a Multi-Vendor Environment

István Pölöskei[1], Udo Bub[2]

[1]adesso Hungary Software Kft.
Infopark sétány 1, 1117 Budapest, Hungary
E-mail: istvan.poeloeskei@adesso.eu
[2]Eötvös Lóránt University (ELTE) Budapest, Faculty of Informatics
Pázmány Péter sétány 1/C, 1117 Budapest, Hungary
E-mail: udobub@inf.elte.hu

Abstract: With new web applications rapid growth in size, the monolithic frontend approach has been increasingly challenged over the last years. The micro frontends concept has been proposed to match the architectural needs facing increasing complexity, e.g. for newly emerging cloud-native solutions. It provides function-level granularity and lets the developer adjust each process's performance. In the majority of relevant cases for which the micro frontends paradigm is considered, the system should be migrated, or an existing monolith should be converted to new technologies, whereas, the scientific community mainly focuses on greenfield designs. In this paper, we validate the paradigm, with a case study, in the context of migration for enterprise information systems, in a multi-vendor environment. Based on our analysis, we propose a method for the migration of frontend monoliths, along with guidelines and recommendations for future work.

Keywords: micro frontends; multi-vendor; migration

1 Introduction

Web interaction layers are crucial for rendering a feature-rich browser application. The service is deployed online through a pointer like a public web endpoint, serving a frontend as a target web application user interface. This layer's code is growing bigger because some of the new logic implementation. The latest applications are behavior-driven; the frontend application may also contain some business logic. Concentrating on the frontend is reasonable because the administered components may affect the global usability of the product.

Some frameworks and principles have been revealed to enhance its general effectiveness in recent years, like service composition and web services [1]. The introduced concepts have delivered software elements for accomplishing the business needs based on technology, being controlled by different developers and

teams or organizations. Few of the designed architecture components have become legacy later because they are built on a deprecated technology or framework.

In enterprise applications, typically, the components have been developed evolutionally. It implies that the applied frameworks have been replaced continuously. Modern software architectural concepts should sustain the coexisting of different approaches. Without a modern frontend framework, realizing such an outstanding frontend application involves additional complexities [2].

The monolithic strategy has been challenged in the last years. In an enterprise software system, the code-base can regularly be so huge that it is not maintainable. The first advances were the modularization and component-based solutions, leading to the service-oriented architecture paradigm [3], where e.g. [4] gives an overview. Recently it has evolved further to a micro-service or micro frontends architecture (Figure 1) for getting the dynamism of cloud computing [5].

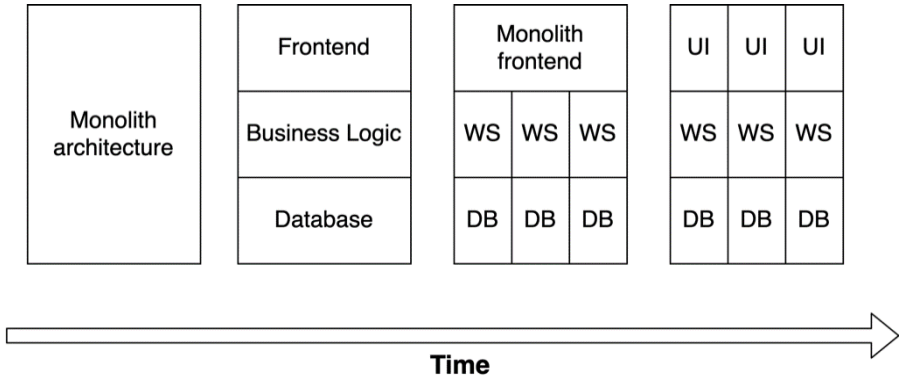

Figure 1

Evolution of architectures from the frontend point of view. Monolith – 3 layered – Micro-services/SOA – Micro frontends. While systems are growing, it involves the growing diversity of technology. WS (Web Service), DB (Database), UI (User Interface).

The current research examines an efficient monolith frontend migration strategy to the cloud by using a compound approach to find a cost-effective way to migrate the UI tier.

2 Frontend Approaches

2.1 Micro-Services and Frontends

Micro-services are getting more and more popular. Some corporations have preferred this architecture against the conventional monolithic one. Initially, this strategy was more or less restricted to backend services. It was intuitive, splitting the entire backend logic into miniature independent services. Each service is accessible through the API, realizing the REST principle. Although, a comparable breakdown had not proceeded with the frontend tier.

The growth of the frontend was also commenced but barely on the code level. After server-side rendering, contemporary JavaScript-based frameworks were becoming popular. Nowadays, some organizations attempt to realize their Single Page Application (SPA) based software in their landscape, presenting a single HTML page with dynamically loaded page content [2].

In brief, the monolithic frontends survived, along with service-oriented architecture and micro-service backend design, where they were still generally used [6]. In that case, the backend was innovative, but the frontend tier remained conventional. The frontend code had not evolved; it used just the traditional modular strategy, a bright compilation of tiny applications. The deployment of a complex bundle was challenging as well.

The micro-service architecture advantages are formulated, but why does the industry not use it on frontend projects [5]? It might have some positive consequences, producing the features of the same domain by a single team. However, there are still some dependencies between the frontend and backend. It makes it more complicated to discover a bug in the productive system if there is no transparent responsibility for the business domain's set of features [6].

An uncomplicated solution to the dilemmas discussed above is the micro frontend design, which complements the micro-services architecture in the presentation layer. Each company has its resolution to this obstacle, using the equivalent principle as micro-services. They were directed to the same outcome while producing micro-apps using isolated endpoints; they experienced some positive attributes [5]. The isolated applications can be merged into a portal-like single page application as well. From the user perspective, it remains only one application, but architecturally, they are self-governing entries.

By the micro frontends pattern, the frontend element synthesis is done in the client's browser. Frontend services are implemented by the frontend (JavaScript) code; through the web applications, the backend services can be requested. The host application assembles the components as services, qualified only for the routing, component selection, and communication [7].

2.2 Frontend Monolith

As the brand-new web applications are getting bigger and bigger, it serves unusual, unique difficulties to the architects. For example, the monolith frontends may suffer performance issues because it is not a trivial task to fine-tune each unit's performance.

The deployment of such a big client bundle is also challenging since DevOps pipelines are not well supported by the monolith architecture [8]. Moreover, the reusability of each unit is also not irrelevant. As a result, the complexity remains high compared to the well-structured applications.

The introduction of new technologies is not trivial in this landscape. Some parts must be reworked before a different element has been injected. The platform dependencies are potential bottlenecks in massive projects.

2.3 Main Features of Micro Frontends

The functional division has its benefits. The main one is scalability. This concept fits well into the latest cloud-native solutions [9]. A function can be scaled freely. The architecture remains on the frontend side as adjustable as on the backend side. A process with a more extraordinary load (request number) can take more extra resources by allocating more instances. It could be achieved in a "self-driving" manner through modern auto-scaling solutions [10].

The advantages of this pattern are identical to working with a cloud-native architecture. Each project can be managed by separate repositories (multi-repo approach) [11]. The separation, on the code level, establishes the final software structure. An arrangement by autonomous repositories makes it attainable to have a well-defined system in the organization [11]. A product could be developed in a flat structure; the teams can be systematized according to their business domains. Only a thin architect layer is obliged to define the direction of the entire application. It also decreases the business dependencies between teams. It has some privileges in grouping the functionalities by areas [6].

The deployment is also a necessary aspect. The micro frontends concept fits not only into cloud-native landscapes but also into DevOps philosophy [12]. It promotes the more durable build and release cycles natively. The small chunks have another positive characteristic as well, more satisfying testability. Atomic automatized tests can be executed if the complexity is low.

The application can handle different technologies (Figure 2), but the synthesis remains manageable. Each team is free to use its own preferences to formulate its unique user interface. From a product point of view, the implementation of each micro-app is hidden. The provided application does not depend on the built-in technologies. The micro apps are framework agnostic, but following some usual guidelines and naming conventions is advised for the complete picture.

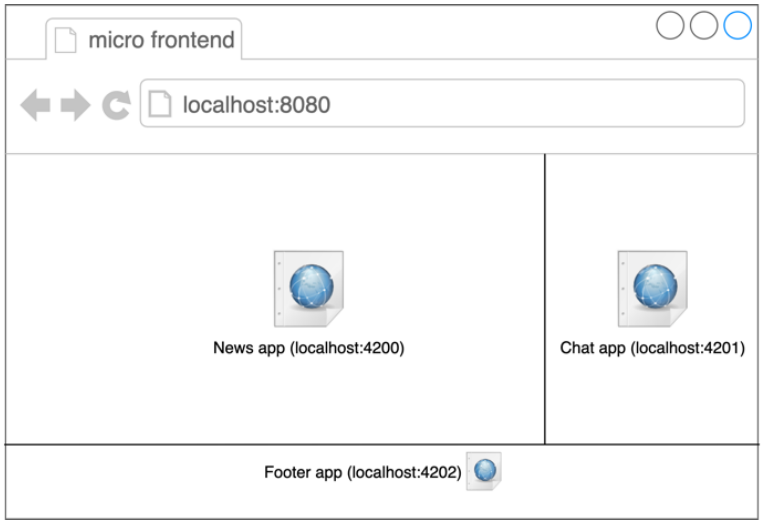

Figure 2
Frontend as a set of applications from different endpoints

2.4 Micro Frontends in a Cloud-Native Environment

Cloud-native solutions, like Kubernetes, provide their open-source container orchestration services [13]. Scaling and deploying the application is sufficient with multiple instances, allocating further resources for business-critical processes. This method actively supports the micro frontends theory because the well-defined granularity lets the orchestrator adjust each process's performance.

2.5 Drawbacks

This approach with multi-repositories involves a complexity baseline. The communication between each micro application has overhead; the polyrepo is recommended only for huge enterprise solutions, providing the choice of organic evolution [11]. Because of built-in complexity, some expertise is required for the micro frontends.

Debugging such an intricate and fragmented landscape is a complicated task. It may take time to locate a bug among dependencies, and they might easily get outsourced. Issues seldom occur during the authentication (single sign-on) mechanism because the creation of the tokens' transportations can be problematic. Since current browsers could apparently handle several built-in applications on the same screen, it might suggest different logins claiming three authentication mechanisms. For fixing this issue, introducing the backend for the frontend (BFF) layer is recommended (Figure 3). BFF can regulate the backend API calls [14].

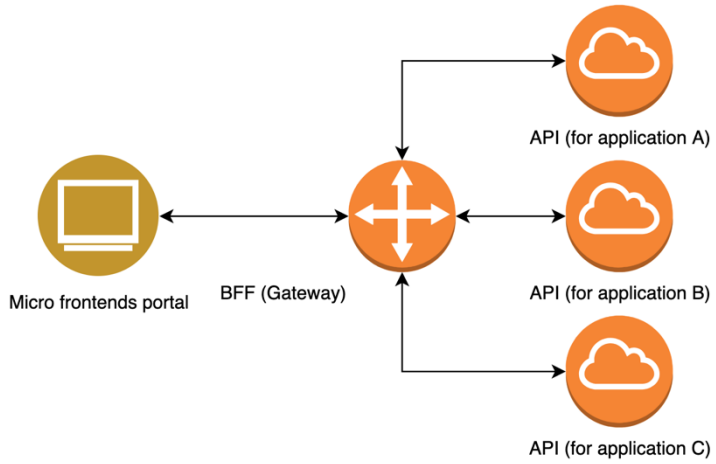

Figure 3
Backend for frontend pattern

Standard guidelines are required. In a portal-like structure, the apps should follow the same design principles. Redundancy in the source code may occur because the framework elements and base controllers could be bundled together in each micro application.

2.6 Domain-driven Approach

Assume an application that has some sovereign functionalities. Despite having enough developers, it could be challenging to handle it. Providing the necessary business knowledge for each developer is a hurdle, but it is possible to organize the human resource according to business domains or architectural layers during the preparation.

The micro frontends concept is a modern approach where teams own the feature sets of business areas. They are independent or loosely coupled, oriented by business domains and specialists. The teamwork is full-stack, affecting each layer, from the database until the user interface.

The implementation is effective. The team can use the DevOps features at their full capacity [12]. The micro applications are detached; they may have their versions. The versions are independent of the other releases. The micro-app can be introduced individually.

The procedure is matching well the agile methodology. Achieving in increments is also a potent compound with micro frontends architecture. The combination of new features or redesigning the traditional ones is feasible in tiny increments. This mixture continuously contributes business value through the project [15].

3 Integration Techniques

3.1 Multiple Endpoints

The original approach of multiplied endpoints is not an imperative section of the micro frontends idea, but historically it is the main root and initial trigger. It means a set of applications, which live under different endpoints (for example, each route is provided by reverse proxy). This approach does not use the advantage of Single Page Application. After switching the route, complete side reloading is required.

3.2 Iframes in a Single Page Application

The first, native HTML5 based approach uses Iframes [16]. They can split a compact web application into smaller elements; the components get isolation and security by the Iframe. These parts are assembled into a frame-like formation. Currently, it is one of the most generally used solutions. It realizes the simple combination of technologies, by providing a certain degree of sovereignty among Iframes.

Through its native API, communication between components is feasible. While providing separation, the event flow is also possible. The Event bus (like message queue) is the state-of-the-art recommendation for extending the communication between Iframes [16].

The usage of Iframes is not such a universal solution [7]. It can provide a remote endpoint as a built-in frame, but it is not such a flexible structure. Providing a responsive modern application requires some additional structures and complexities. The usage of other solutions based on the modern JavaScript frameworks is recommended because some unexpected issues may occur during the routing and navigation between Iframes [7].

3.3 Web Components

Since 1995, JavaScript became the number one language of the web. Currently, the websites frequently use one of the three main JavaScript frameworks: Angular, React, and Vue [17]. Since enterprises frequently choose an Angular framework, its features as a web component supportive framework will be described.

The Web Components concept is a standard by W3C; some modern browsers have already implemented it, providing a run-time integration possibility for the components [7]. The web component concept is the remote extension of the original components approach. According to the DRY (“Don’t Repeat Yourself”) principle, any web application can easily import the frontend components. This way, the entire frontend application can also be imported from diverse

remote locations; like serving a remote resource, the pieces are encapsulated apart from the host code.

The User Interface components provider is a service, which is not just a conventional REST endpoint; it also publishes the required User Interface components. The web component can encapsulate various technologies and frameworks, as well. Through the provided UI component, the business logic is also available by calling the backend services.

Since the web components use the browser API, some compatibility issues may occur [18]. It is not guaranteed to be compatible with all possible browsers. If too many components are imported from remote sources, some network performance issues may occur. Since the implementation may involve some complexities, popular UI frameworks are recommended, like Angular Elements [18].

By custom elements, each micro frontend service can be converted into components [6]. A published element is a reusable component with its CSS and JavaScript logic. It could be installed directly on the shell web application. Each team can develop its traits and declare them as a pure service. They can also be extended by lifecycle hooks, as call-backs. It is possible to assign properties to them. The events are triggered if DOM's manipulation occurs. DOM (Document Object Model) means the abstract tree-like model of the HTML page, where the injected components are inserted or rendered. This technique includes some intercommunications with DOM; each element fits into the original DOM by using namespaces.

If the micro applications are Angular applications, they can be pushed as angular libraries (like NodeJS modules) into a shared repository. The published modules can be loaded in a lazy loading manner to reduce network traffic [19].

4 General Frontend Migration Use-Case

4.1 Problems with the Frontend Migration

Enterprises try to realize efficiency gains by using cutting-edge technologies once they are mature enough to compensate for the attached efforts. A new frontend framework serves some advantages. It increases the user experience, making the developer's work more efficient and effective [17].

Upgrading the existing landscape is challenging; it slows down the introduction of new business features. Some new functionalities can be restricted by technology constraints on the frontend side as well. Getting rid of legacy factors is never simple because they include complexity based on last years' developments. Since the implementation is growing over the years based on new technologies, the complexity mirrors this trend because its volume is more significant.

The migration use case was increasingly crucial for service-oriented concepts [4] and has gained further momentum nowadays [14].

In most cases, when the micro frontends pattern is under consideration, the system should be migrated, or the existing monolith should be converted to new technologies. This pattern allows continuous modernization by operating the legacy components until they have been replaced.

The migration of monolith systems is not a trivial task. Some necessary backend services might belong to the frontend applications, not only strongly UI applications. One migration methodology fits well to the backend dependencies; another one might be better for the UI use-cases. Businesses need to be cost-conscious. The migration to the cloud is managed cost-effectively. The migration to the new technology involves potentially cost-savings.

Our research question targets the effective frontend migration strategy. Based on our literature research, the various frontend applications can be handled as a micro frontend portal. To identify the appropriate method for migration, we investigate the different approaches in the area of migration to synthesize a compound one for our use case. The applied strategy needs to improve the company's overall productivity.

4.2 Proposed Solution

At the beginning of the product lifecycle, a single frontend monolith method is a popular strategy. It's suitable until the volume becomes too huge. After a critical size, the structure becomes a limitation; the development becomes unscalable. The introduction of modern cloud-based solutions could not cooperate with a frontend monolith. It involves some difficulties to the development. Domain separation presents independence for the developers.

The preparation for the migration and transformation between the monolith and micro frontends should begin with identifying bounded contexts and domains. The domain-driven design is an undecayed approach for discovering monolith landscapes. It involves some code investigations as well. After the modeling of the AS-IS landscape, the granularity of the migration can be defined. The costs are critical because the granularity of the redesign is considerably influencing it.

The commonly used migration strategies are mainly applied in cloud migration projects: lift-and-shift, re-platforming, refactoring [20]. Lift-and-shift is the most straightforward and cheapest strategy. State-of-the-art public cloud services can practically automate it. The combination with other techniques is also possible since the remaining two methods could be more effectively managed if the application is hosted by the cloud initially. The primary rehosting by lift-and-shift approach is a good milestone before the notable restructuration while using the public cloud services as soon as possible. The re-platforming involves a few

modifications on the application level as well. The cloud services frequently require at least a few changes on the application's side, setting the necessary interfaces. The re-architecting necessitates an in-depth business review to fulfill the function and non-functional requirements. Since the reimplementation takes some time, the operation and implementation costs are higher.

The migration is advised when the previous application turns too complex or architectural modernization is essential. The agile mindset is vital in cloud migration projects, bypassing the big bang implementation style. The Minimum Viable Product (MVP) milestone is a valid option for collecting the customers' feedback. The migration projects strive to minimalize the expenses, pushing the lift and shift approach [21]. This could also be performed by re-platforming the original applications. The refactoring is costlier, but provides the option to use the serverless technology that makes it possible to hide the dependency of the servers by the services of the public cloud providers, providing a more affordable and productive usage of cloud computing. The contemporary design methods fit well in the event-driven logic.

5 Frontend Migration Case-Study

We will validate the applicability of the approach for migration in a case study. It is structured as proposed in [22] with the following sections: Situation Faced, Action Taken, Results Achieved, and Lessons Learned.

5.1 Situation Faced

According to the global decommission program, a particular location of a multi-national company gets rid of the legacy software elements. This process has been made together with infrastructural modernization and cloudification. The demand is to migrate the legacy on-premise service-oriented architecture to the united micro-services-based architecture. The examined work-stream focuses on migrating the presentation layers to find an adequate way to migrate frontend monoliths. The existing monolith frontend applications should be reconstructed for a more stable usability. The outcome of the study is discussed in the following sections.

During the migration, the critical measurement is the costs. A superior but costly solution can be achieved by reapplying each feature as a new project based on recent technologies. In real-world projects, the budget is restricted. Bound resources should realize the relative best technology level. It comprises some cost-effective methodologies.

Based on the existing design documents, and architectural snapshot has been created for having an input for the migration (Figure 4). The architecture is high level, focuses only on the separation of layers and endpoints.

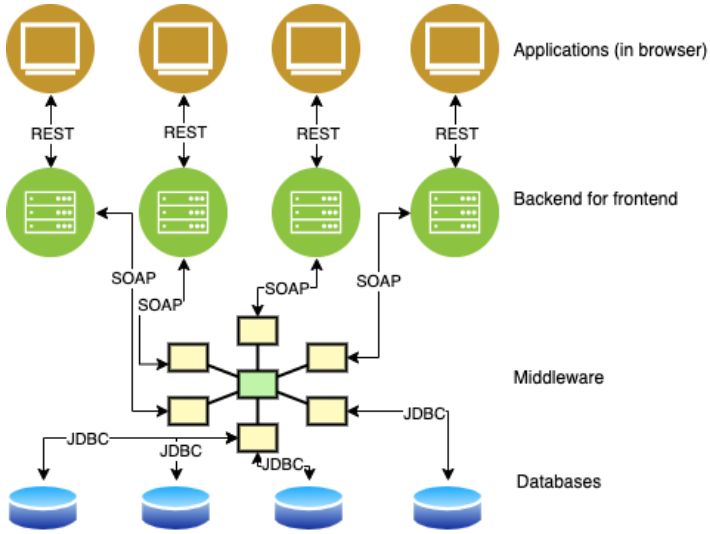

Figure 4

High-level enterprise AS-IS architecture. Each monolith frontend application has its backend for the frontend provider. Through middleware, the backend for the frontend layer can communicate with databases. On the database layer, some logic has been implemented.

The system is service-oriented and on-premise. The service-broker middleware element provides the services. The backend functionalities are accessible through the backend for the frontend layer for consolidating the API. The migration of the backend part is not part of the scope, but the new frontend must connect to the legacy and new backend services. The original goal is the migration of the full set of business processes. Current research focuses only on the frontend; this paper will not describe the migration of the backend side.

5.2 Action Taken: Target-Architecture Design

The proposed target architecture will serve as a target solution for global technical requirements. The main requirement is cloud-nativeness for implementing a modern architecture in the location. The frontend layer needs to support and extend cloud-based approaches. The new architecture should be compatible with the guidelines and cost-effective as well.

During the preparation, the potential approaches of the scientific literature have been investigated. The architectural guidelines already determined the set of concepts. The combination of a portal-like frontend as a collection of micro

applications has been chosen. The related proof of concepts has clarified the implementation details.

Functionally, the applications are getting united. Owning only one application has a tremendous user experience and business value. The frontend codes are behavior-driven; they contain some JavaScript parts as well. However, it increases the complexity and the size of the bundle in the browser. Lazy-loading is necessary for satisfactory client-side performance. The required modules should be loaded if they are necessitated. Reusability is also a fundamental aspect; some frontend functionalities might be served directly to the various platforms.

Previously discussed integration methodologies have already been reviewed: route level separation by reverse-proxy, Iframe, Web Components, Angular specific web-components implementation. For this business need and baseline architecture, the introduction of Angular-based web components serves potentially a robust solution. A Proof of Concept has proved this statement. It is possible to handle the existing applications as independent components by web components, but they are merged into a portal frame. Transformation of each element to web components is not challenging. Its micro frontend service can host this component. Each frontend service must provide the UI with the exact design requirements. The introduction of an ordinarily used framework or style is reasonable.

The Angular framework is often used in enterprise-grade complex frontend solutions. Some organizations have built their application portfolio like islands, having separate web endpoints for each service or just navigating between them, but managing a massive monolith might be less productive.

With micro frontends, merging the whole distributed application portfolio is conceivable by implementing a single portal for all purposes (Figure 5). Therefore, the research has been looking for the best option of merging some individual applications into one modern single-page application.

First of all, the project manager should clarify the scope. The AS-IS architecture probably contains some legacy or deprecated elements. Over the years, some processes or components might be deprecated from a business perspective. The management can cut unnecessary costs by planning the migration precisely. The detailed plan contains the migration schedule of each dependency as well.

Concerning scheduling, providing results as early as possible is essential. The migration must start with the core modules or the low complex ones. The minimal viable product involves some support from the management side. This approach is strongly supported by micro frontends architecture. By them, old on-premise applications and modern cloud-based ones can be served together in the same portal shell. At the early stage, the portal can publish the old on-premise applications; after each sprint, they can be replaced easily without effort.

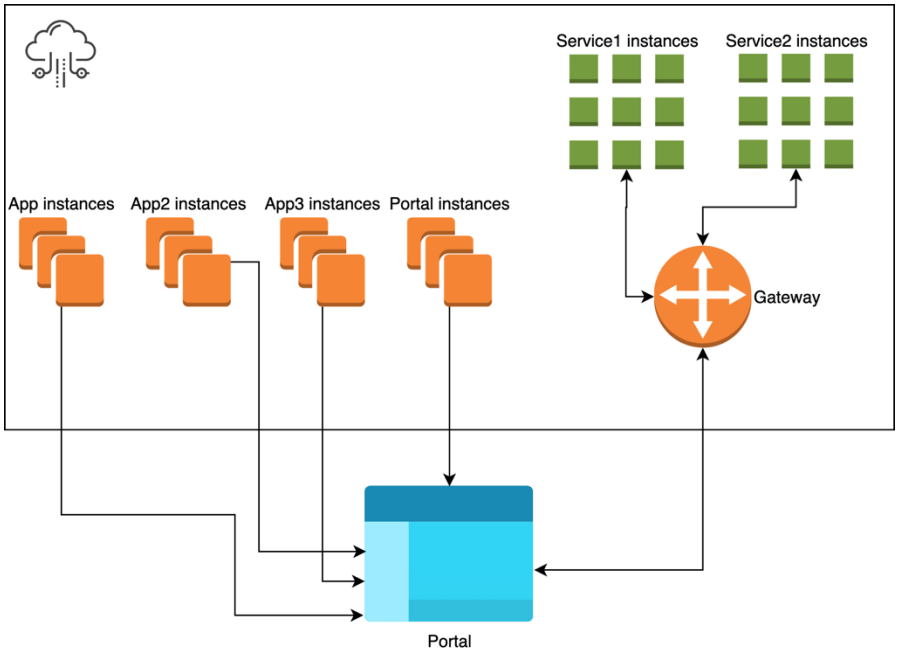

Figure 5

The proposed high-level target architecture in a cloud-native environment [13]. Each application provides its user interface. Each app is integrated into the portal frame. The applications can use the backend API through a consolidated gateway (Backend for frontend pattern).

The Micro-services approach supports the usage of different technologies for each component. The simplest method is the implementation of web components. They created independent web components that can be easily injected into a Single Page Application Container [16].

For building such a system, some other dependencies are needed. The modern JavaScript ecosystem natively supports the implementation of micro frontends by web components. Each component is a different app; they listen to different ports. The Single Page Application Container (host or shell application) will merge and serve them as a single website. From the user perspective, the fragments seem to be a homogeneous web-application.

Modern JavaScript frameworks can create components. They need to be encapsulated into web components, to publish them through the network. In Angular, the AppModule root module should export the whole (micro) application as a web component. In React and Vue, the implementation is a bit more complicated than in Angular, but each frequently used framework has its solution. Angular has another added benefit: Angular CLI supports the whole build – the deployment process natively. For other modern frameworks, some build and deployment tools are required, like webpack.

There are many possible ways to build micro frontends based portal applications. The most obvious choice is the usage of Web Components or Iframe with an event transmitting system. These native elements can fulfill this goal, but some bottlenecks may occur, like the poor communication between elements and code redundancy. For avoiding the weaknesses, some frameworks were introduced [17].

The Angular elements library was created for sharing components between applications; it provides simple integration for the micro frontends. Angular is a well-known product in the industry. It is a battle-tested solution from Google for enterprises because unknown open-source libraries and frameworks might contain some security vulnerabilities. A possible alternative of Angular Elements is the injection of Angular modules for reaching additional code-level features. In that case, both applications should use the Angular framework [19].

The cloud-native techniques are used to reach the cloud goals, based on a containerized infrastructure, not being dependent on a cloud provider. The cloud-native deployment can be effectively supported by DevOps tools, serving the demand of high deployment frequency [12]. The migration tries to minimize the development costs, applying the lift-and-shift approach as many times as possible [21]. For the introduction of a container-based platform, the refinement of the modules is needed. The modules contain the backend for frontend (BBF) parts as well. In a frontend project, the BBF layer is mainly responsible for serving the frontend code and applying the authentication mechanism, transforming the middleware services into REST API. In our case, the processes' refinement is just optional because the containers can be launched without any problems.

The pages and applications should have the same design. The integration from the user perspective should be smooth. The inconsistencies might damage the customer's satisfaction. In our case, the design of each application should be the same based on a shared library. The library provides common design elements and components.

5.3 Results Achieved

The current enterprise's primary goals are cloud migration, merging frontend applications supporting a portal-like structure; each built-in application will be rendered in the same style. The migration is driven by cost optimization; it involves a low amount of resources for attaining the goals.

The migration of frontend applications uses different migration strategies to combine lift-and-shift, re-platforming, and refactoring. Since the core backend services contain an inefficient and difficult code basis, the lift-and-shift strategy was the only option. Without the necessary expertise of the legacy services, their migration will be done at a later stage. The frontend application migration offers more extensive flexibility. Without much risk, the User Interface can be re-

architected according to a different approach. It also gives the possibility to present functionally noticeable results directly to the end-user.

Angular-based web components support the theory of micro frontends. The application can be bundled into a single JavaScript file, and each backend serves it for the frontend instance. The portal frame on the client-side can manage the provided applications like a native angular component, merging them, hide, or make them visible according to the client's credentials.

This architecture can use the cloud native's full potential on the frontend side as well. The migration to cloud-based micro frontends architecture lessens the calculated operational costs in the above-discussed case.

The accomplished proof of concept is built on Angular Elements web components. The elements are united into a portal structure. The main benefit of the polyrepo approach is separation. The idea extends the micro-services, providing a platform for accessing the old and new functionalities together.

Strengths:

- The newest Angular versions support the Ivy compiler; the generated components are reusable by other frameworks
- Only the relevant apps are loaded in the portal frame
- Independent repositories can manage the project. Repositories have their version number
- Short deployment cycles
- The flexibility of the structure
- There are no compatibility problems with the frame's Angular-based logic
- The effort for the integration of the existing applications is reduced
- Project security is improved, the repository access can be limited to the application, which is helpful in a multi-vendor environment
- Applications can have their dependency versions
- Legacy and modern can live together

Weaknesses:

- Some dependencies are redundant; each web component may include its own
- Memory and the network load are higher because of redundant dependencies
- Analysis of the issues is challenging because of dependencies
- Merging the application's frontend code into one bundle makes debugging challenging

Modifications made:

- The applications are being embedded into an Angular shell application
- Usage of shared libraries. They are registered in the global scope

5.4 Lessons Learned

The above-discussed proposal is a good fit for the migration use case. Current applications are behavior-driven, a huge code-base for complex business logic is demanding for the client and network.

The containerization-based lift-and-shift approach provides a valid option for promptly migrating the core standard services, like having a suitable wrapper around the application. It gives a further granularization, after the central cloud re-platform step since the lift-and-shift initially doesn't reach the cloud-natives principal gains. Like breaking down the software into services and bigger units, several elementary modifications can serve the scalability. For the most cloud-optimized result, strong refactoring is needed.

The approach supports the incremental delivery natively, being a good combination with the agile paradigm. Having a flexible software structure is a real advantage. Old on-premise frontend applications can be integrated into a micro frontends frame as non-native components instead of navigating to a new tab in the browser. The legacy components can be replaced later by a native sub-application if the feature has been migrated to the cloud. Resource constraints sometimes limit migration projects. The project manager needs to find the simplest solution for realizing the target. Part results and milestones are essential for getting the support of project sponsors.

Conclusions and Future Work

During our research, the polyrepo micro frontend solutions were investigated. The principal goal was to determine the optimal solution for our migration use case. For being cloud-native and having some separate applications on the frontend side, the implementation needs to follow the micro frontends pattern, with a polyrepo approach. This approach provides a run-time native integration in the browser. The cloud-native multi-instance environment can allocate additional resources to each process, giving a dynamic response to higher request quantities at any point of the designed system. This guarantees higher efficiency and lower costs.

The case study presents an example of the potential use-case of the micro frontends pattern. In an enterprise, each application can be delivered by vendors. In a multi-vendor environment, each legacy app has its restricted know-how. If the target is a common standardized portal, the baseline diversity makes the designing troublesome. The micro frontends concept grants freedom for the implementation teams, but it has its obligations as well. The standardization of the UI design and

having a well-defined integration methodology is a necessity. The teams are limited to their domain by their own repository. It provides built-in security, in a multi-vendor environment.

The web components have been used in the case study. For an enterprise-level Angular project with distinct teams, the micro frontends pattern should be considered. The benefits concerning the separation of applications could be helpful to migration and re-architecting projects.

The concept of micro frontends is a business-oriented strategy. Business domains separate teams and applications. The members of each team are specialists. Since the micro frontends provide flexibility, each business need could be realized through the best fitting technology.

The micro frontends pattern is used in industry, but still, there is room for improvement. Based on the careful evaluation of the state-of-the-art and the above-discussed case study, a new generic strategy could be introduced, for micro-frontends-based migration.

References

[1] A. L. Lemos, F. Daniel, and B. Benatallah, "Web service composition: A survey of techniques and tools," *ACM Computing Surveys*, Vol. 48, No. 3, 2015, doi: 10.1145/2831270

[2] F. Shahzad, "Modern and Responsive Mobile-enabled Web Applications," *Procedia Computer Science*, Vol. 110, pp. 410-415, 2017, doi: 10.1016/j.procs.2017.06.105

[3] P. Offermann and U. Bub, "A method for information systems development according to SOA," *15th Americas Conference on Information Systems 2009, AMCIS 2009*, Vol. 2, pp. 908-918, 2009

[4] P. Offermann, M. Hoffmann, and U. Bub, "Benefits of SOA: Evaluation of an implemented scenario against alternative architectures," *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, pp. 352-359, 2009, doi: 10.1109/EDOCW.2009.5331973

[5] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality," *Proceedings - 2019 IEEE International Conference on Software Architecture - Companion, ICSA-C 2019*, No. March, pp. 187-195, 2019, doi: 10.1109/ICSA-C.2019.00041

[6] C. Yang, C. Liu, and Z. Su, "Research and Application of Micro Frontends," *IOP Conference Series: Materials Science and Engineering*, Vol. 490, No. 6, 2019, doi: 10.1088/1757-899X/490/6/062082

[7] A. Pavlenko, N. Askarbekuly, S. Megha, and M. Mazzara, "Micro-frontends: Application of microservices to web frontends," *Journal of Internet Services and Information Security*, Vol. 10, No. 2, pp. 49-66, 2020,

doi: 10.22667/JISIS.2020.05.31.049.

[8] M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," *Journal of Systems and Software*, Vol. 170, p. 110798, 2020, doi: 10.1016/j.jss.2020.110798

[9] N. Kratzke and P. C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *Journal of Systems and Software*, Vol. 126, pp. 1-16, 2017, doi: 10.1016/j.jss.2017.01.001

[10] S. Kho Lin *et al.*, "Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes," *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, pp. 327-334, 2019, doi: 10.1109/UCC-Companion.2018.00076

[11] N. Brousse, "The issue of monorepo and polyrepo in large enterprises," *ACM International Conference Proceeding Series*, 2019, doi: 10.1145/3328433.3328435

[12] E. Dornenburg, "The Path to DevOps," *IEEE Software*, Vol. 35, No. 5, pp. 71-75, 2018, doi: 10.1109/MS.2018.290110337

[13] K. Indrasiri and P. Siriwardena, *Microservices for the Enterprise: Designing, Developing, and Deploying*. 2018

[14] A. Henry and Y. Ridene, *Assessing Your Microservice Migration*. 2020

[15] D. Sjödin, V. Parida, M. Kohtamäki, and J. Wincent, "An agile co-creation process for digital servitization: A micro-service innovation approach," *Journal of Business Research*, Vol. 112, No. January, pp. 478-491, 2020, doi: 10.1016/j.jbusres.2020.01.009

[16] M. Mena, A. Corral, L. Iribarne, and J. Criado, "A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things," *IEEE Access*, Vol. 7, pp. 104577-104590, 2019, doi: 10.1109/access.2019.2932196

[17] S. Peltonen, L. Mezzalira, and D. Taibi, "Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature Review," *arXiv*, 2020

[18] P. J. Molina, "Quid: Prototyping web components on the web," *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2019*, 2019, doi: 10.1145/3319499.3330294

[19] M. Hajian, *Progressive Web Apps with Angular*. 2019

[20] N. Ahmad, Q. N. Naveed, and N. Hoda, "Strategy and procedures for Migration to the Cloud Computing," *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences, ICETAS*

2018, pp. 1-5, 2019, doi: 10.1109/ICETAS.2018.8629101

[21] D. S. Linthicum, "Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points between," *IEEE Cloud Computing*, Vol. 4, No. 5, pp. 12-14, 2017, doi: 10.1109/MCC.2017.4250932

[22] J. vom Brocke and J. Mendling, "Frameworks for Business Process Management: A Taxonomy for Business Process Management Cases," pp. 1-17, 2018, doi: 10.1007/978-3-319-58307-5_1