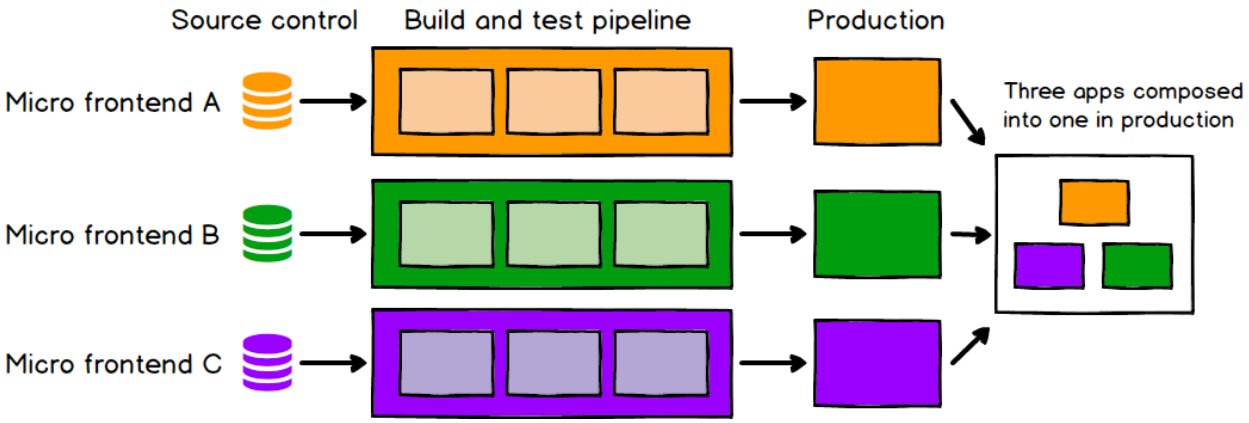
MICRO FRONT ENDS



- # Microservice

A microservice is an architectural pattern where applications are built as a group of smaller independent services. This is based in decomposing an app in single modules that have a well defined interface. They can be deployed by small independent teams.

Despite being independent, they will have to comunicate with each other, so this can be a challenge.

- # Monolith

A monolithic app is a single software product where the user interface and the data access code are together in a single platform. In the case of fronends it is common to see only a single monolithic front end with multiple micro services targeted to the back end.

- # Micro frontend

A common error that teams using microservices makes is creating a monolith frontend, neutralising the benefits of microservices. Micro frontends are a way to address this problem.

A micro frontend is an extension of microservices to the frontend, instead of having one team in charge of every part of the program, the software is divided into features. Each feature is controlled by a small team from the database to the frontend.

Each team can choose its own software to developed it and are totally independent with the other teams even the deploy process is not shared. This allows to update or refactor every feature individually without interfering with other parts of the frontend.

Even though the frontend is developed with different technologies and teams, what that the customer gets is a single cohesive product

Micro frontends can be used in two different forms, horizontal split and vertical split

- Horizontal Split: with this type of micro frontend, each view can be composed by the work of different teams, so they need to coordinate its fusion to deliver the final product to the user.
- Vertical Split: Each team is specialized in one (or more) views, but no other team is involved in those views. This allows the teams to master their fields without having to coordinate with the other teams.

To communicate micro frontends between them there are mainly two different approaches, the first one, using some kind of web-storage (local storage, session storage...) and the second one, requesting the backend the state of the user.

• Benefits

1. Independent deployment:

Independent deployability of micro frontends is key. This reduces the scope of any given deployment, which in turn reduces the associated risk.

Regardless of how or where your frontend code is hosted, each micro frontend should have its own continuous delivery pipeline, which builds, tests and deploys it all the way to production.

It does not matter whatever is doing the team that is next to you. If your code is ready to go to production, it should be able to do so.

2. Incremental upgrades:

One of the objectives of using micro frontend is that you are able to do incremental upgrades to you architecture, dependencies or user experience.

If there is a major breaking change in our main framework, each micro frontend can be upgraded whenever it makes sense, rather than being forced to stop the world and upgrade everything at once.

3. Simple and decoupled codebases

The source code of each micro frontend will be much smaller than the monolithic frontend.

These smaller codebases tend to be simpler and easier for developers to work with.

We avoid the complexity arising from unintentional and inappropriate coupling between components that should do not know between each other.

4. Autonomous teams

As a benefit of decoupling both the codebases and the release cycles we are now getting a mostly fully independent team.

Teams can have full ownership of everything which enables them to move quickly and effectively.

Teams also should move vertically and not horizontally (Presentation image).

- # Downsides

1. Payload size

Independently-build JavaScript bundles can cause duplication of common dependencies increasing the number of bytes that we have to send to the users over the network.

An example could be if every micro frontend owns a copy of React, then we are making our customers download React n times.

One approach could be externalizing all the common dependencies of every micro frontend.

2. Environment differences

We should be able to develop a single micro frontend without needing to think about all the other micro frontends in the application. We may be also able to see our micro frontend in a "standalone", in a blank page rather than the container application that will house it in production.

However there are risks associated to develop in those isolated environments that behaves different than the used in production it may appear that our micro frontend is broken.

A solution could be regularly check manually or automatically how our micro frontend works in that kind of environment.

This is a trade-of to consider. Is the productivity boost of a simplified environment worth the risk of integration issues? The answer will depend on the project.

• Webgraphy

- https://www.martinfowler.com/articles/micro-frontends.html

- https://micro-frontends.org/

-https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513