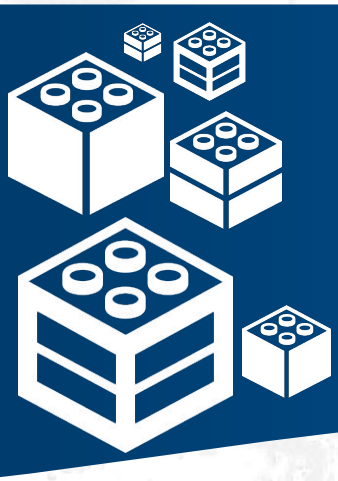
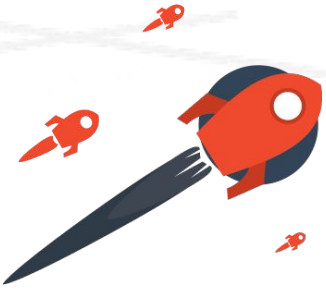
Eleven labs

SOFTWARE ARCHITECTURE



... all you need to know

White book



Vincent Composieux
@vcomposieux

www.eleven-labs.com

SUMMARY

WHAT WIKIPEDIA SAYS

> **Software architecture**[1] refers to the high level **structures** of a software system, the discipline of creating such structures, and the **documentation** of these structures. These structures are needed to **reason** about the **software system**.

This definition, although broad, is not sufficient to understand in detail what software architecture is.

This book aims to help you understand what software architecture is, who the architects are and what their roles are. Finally, we will see some design patterns[2] that can be applied in our web environment.

After several years of experience with various clients, Eleven Labs has acquired the software architecture expertise necessary to successfully complete projects.

Each customer context being different, it is important to know how to respond precisely to a business need, while keeping a flexibility of evolution.

1 https://en.wikipedia.org/wiki/Software_architecture
2 A design pattern is a reusable logic for solving a problem

Moreover, these architectures evolve with time and the technologies that come out.

In the past, many applications were developed in a monolithic way, i.e. an application had a lot of features embedded in it.

With the arrival of cloud hosting offerings and tools to simplify containerization, concepts such as service-oriented architectures (SOA) emerged. With them, all kinds of technologies and tools to facilitate their orchestration, deployment and communication.

Indeed, patterns such as event-oriented architectures have emerged, allowing these services to react to each other based on events they transmit.

These few lines to explain you that it is important to study changes in application habits in order to try to predict future uses.

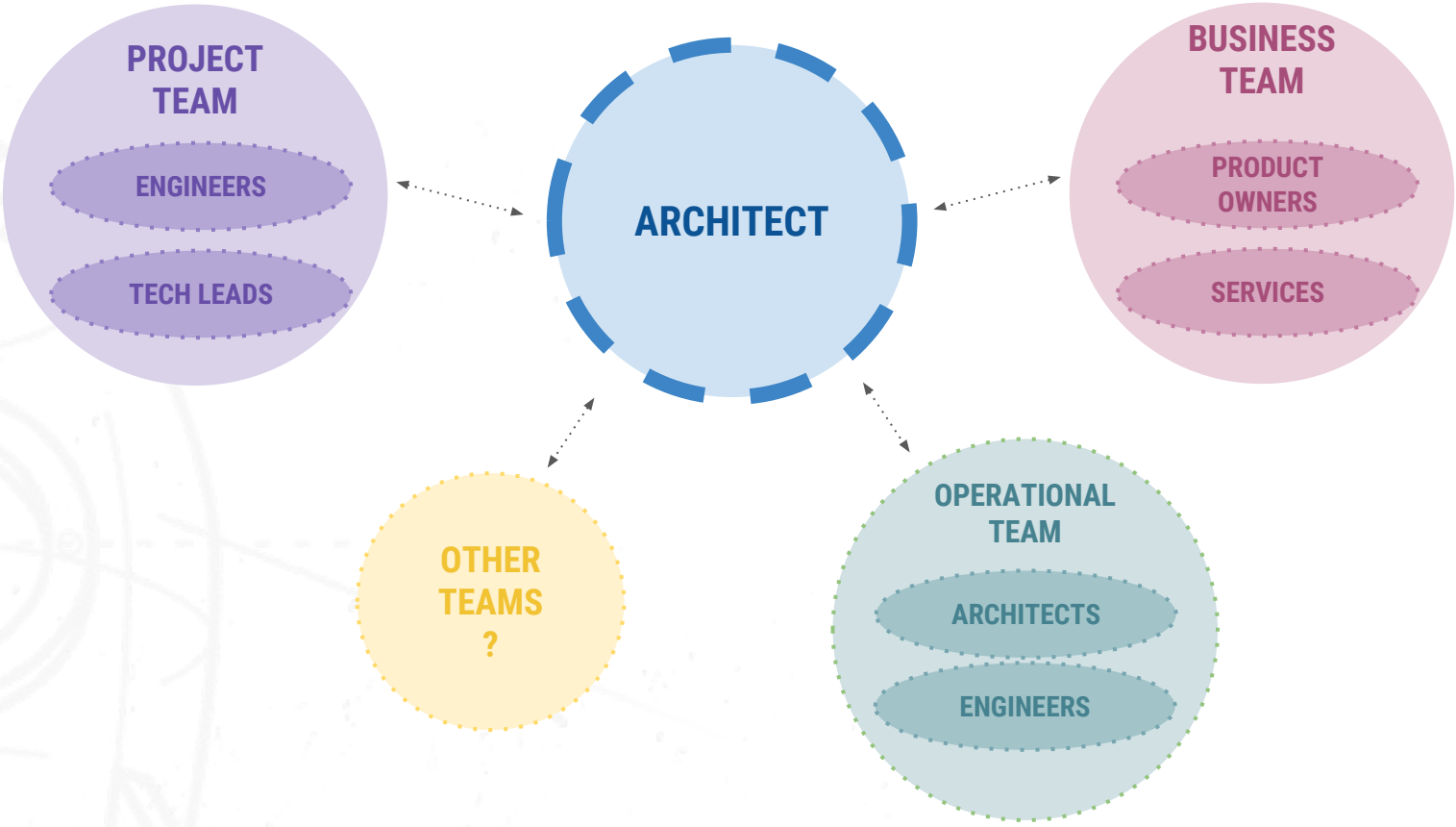
However, although these types of architecture are the trend, it is still important to think carefully about the needs of the projects in order to identify if they are justified. The architectural choices are very structuring to determine the success of the projects.

Each choice must be studied and challenged upstream.

Eleven labs

POSITIONING AND ORGANIZATION

POSITIONING



The architect must position himself as the central element of a project.

Indeed, he must interact with the various actors to be able to articulate the project in the best possible way because a lot of useful information for architectural choices can be brought by these teams:

- **The business team**: it is important that the architect understands the business challenges of the project so that his technical vision fits the needs as well as possible.

- **The project team**: it must intervene to give a clear vision of the architecture defined to the project team and exchange with them on the solution.

- **The operational team**: also often called the operating team, it is the team that manages the platform on which the project will be hosted. It is therefore important that the chosen architecture can best match the available infrastructure. The more consistent they are, the better the application will work in production.

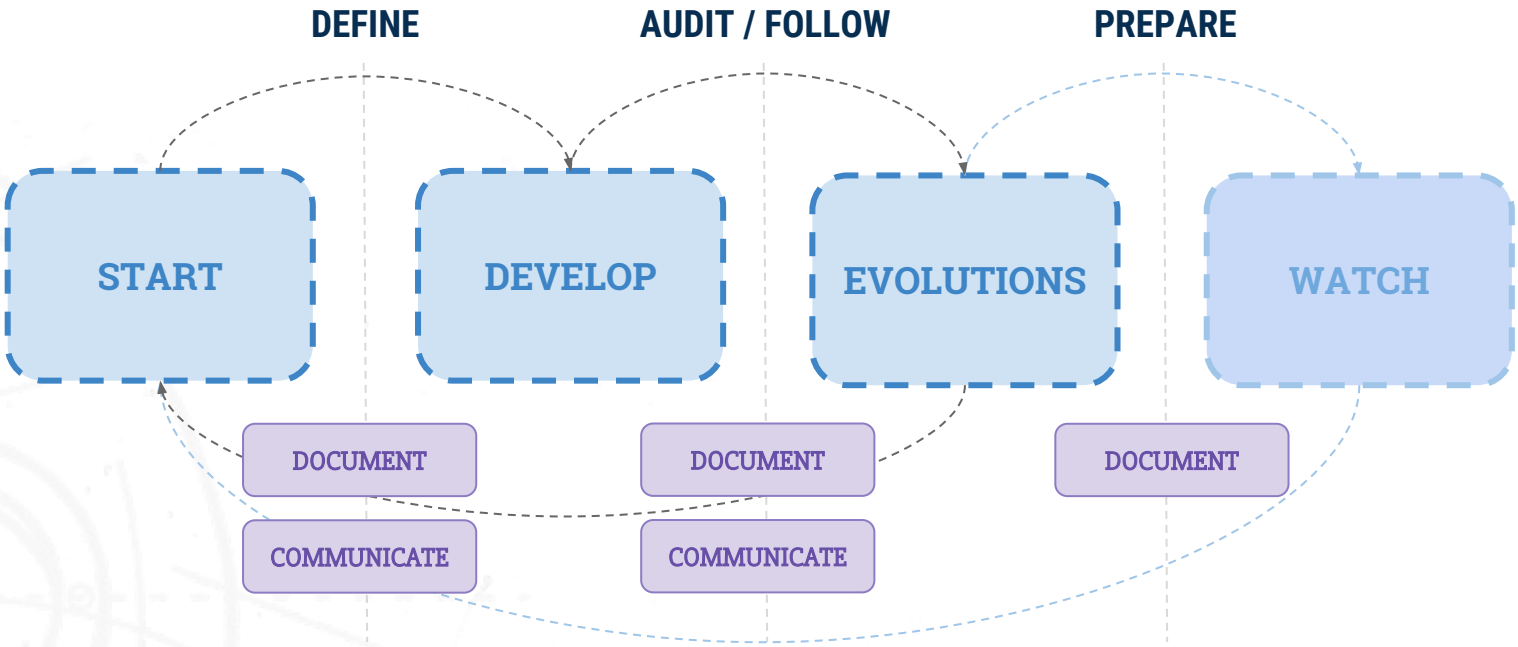
- **Other project teams**: feedback is very important when you have to make choices and it is always good to gather information from several project horizons of the same company: it can indeed make it possible to discover and anticipate certain problems already encountered by others.

The role of an architect is therefore to define an architecture that meets needs while synchronizing the different teams together to find agreements that will then establish an architecture file, referencing all decisions taken and thus give technical and overall visibility to the project.

This file will be shared with the development teams to challenge the selected solutions as well as the operational teams to enable them to set up the most suitable infrastructure for the project.

Personality is also a very important element for this type of role. Patience and diplomacy are needed to allow the different actors to question themselves and find a point of agreement.

The objective being that the teams are satisfied to work on the selected solutions and that they correspond perfectly to the need.



→ Project start-up phase

On the organisational side, the architect is of course present at the start of the project in order to define and freeze an architecture file. This file will then be presented and discussed with the development teams to ensure the feasibility and understanding of the technical elements proposed.

The architecture file depends directly on the business needs. The objective is to provide the technical answer to these needs while keeping in mind several parameters such as :

- the resources available on the project and the technologies mastered by them;
- the ecosystem of tools available and made available for possible reuse;
- the most suitable tools to meet different needs.

However, once this phase is over, its role does not end there.

→ Project Development Phase

Above all, it is important to coordinate the operational teams with the beginning of the development phase so that the recipe platforms and tools necessary for the industrialization of the project do not block the beginning of the developments and recipe.

It also involves project follow-up and exchanges with the teams throughout the development process to ensure that the project is carried out correctly, but also that the implementations carried out comply with the architecture file that had been defined.

The architect must then make himself available to respond to a problem encountered by the teams.

It is also important that the architect audit the application under development to ensure that the product source code has been implemented following the appropriate development patterns and that the code is secure from any flaws.

The OWASP[1] organization provides the top 10 safety rules to absolutely respect on your projects.

Project performance must also be monitored to ensure that the technical implementation achieved delivers a robust and reliable application into production. The flows and data exchanged must then be controlled.

1 https://www.owasp.org

→ Project evolution phase

Once the project is in production, he must anticipate the evolution needs on the project by exchanging with the teams, mainly business teams.

Indeed, a desired evolution could have an impact on the choices made at the initialization of the project: it is then necessary not to hesitate to question these choices even if it means changing a software brick which would no longer be sufficient by a new one which would be better.

The purpose of this is to avoid twisting a tool so that it meets the new need, which also avoids subsequently a technical debt[1] on the project.

The initialized architecture file must then be updated with the new choices.

Throughout the workflow, each decision or change must be documented and above all communicated so that everyone has a clear and up-to-date vision of the project.

→ Technology watch phase

As soon as possible, the architect must practice an important technological watch in order to be informed of each change on the technologies and tools currently used on his projects but also in order to discover new tools allowing him to respond with a better finesse to a need.

1 https://en.wikipedia.org/wiki/Technical_debt

He must then be able to take time to compare two competing tools or technologies in order to get out the positive and negative aspects, according to different contexts of use. This is what will then make the difference on the projects on which he will wish to implement these tools.

Also, it should not hesitate to test freshly released technologies upstream in order to quickly identify a rising technology that would bring a significant gain for a project. It is indeed important not to stagnate on a technology and to be attentive to those available.

These tools are often pushed by web giants[1] such as Google, Facebook, Netflix or Airbnb and Uber who encounter big problems of high traffic or large structure on their projects.

This monitoring phase is often seen as a "bonus" phase by companies. However, it is very important that the architect can free himself time to practice his technological watch. Good choices make good projects, and to be able to choose, you need to know the subject.

1 https://en.wikipedia.org/wiki/Big_Four_tech_companies

REQUIREMENTS TO BE MET

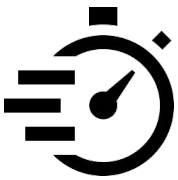
When defining an architecture, and more globally when developing features, the following key points should always be kept in mind:

DATA EXCHANGE AND STORAGE

The data that passes through your application is an important source of user response time.

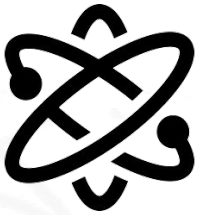
In mobile times, we must pay attention to these data exchanges but also to their storage, which can quickly become voluminous.

EXECUTION TIME



The processing time of the features you write also has a strong impact on the user experience.

Think of the features so as to optimize this processing time, at least so as not to block the user interface.

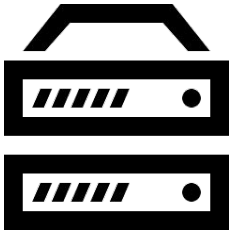


NETWORK
BANDWIDTH & SECURITY

In connection with the data passing through your network, also pay attention to the cost that this can have on your infrastructure (data flows, backups, etc.) as well as the exposure of these.

Data must always be served securely!

PHYSICAL
RESOURCES



Your application, whether hosted in the cloud or not, uses physical resources that have a defined capacity.

Always keep in mind that these resources are limited.

NOT TO MENTION,
THE COST

Finally, the total cost of your architecture, with all these components combined, is often an important element in decision making.

Eleven labs

ARCHITECTURAL PROFESSIONS

SOFTWARE ARCHITECT

Specialist on a **specific technology**.
It describes how the application should be **developed** and **deployed**.

SOLUTION ARCHITECT

Practices **several technologies**.
He knows how to organize **complex applications**.

ENTERPRISE ARCHITECT

Has a **strong experience** on the organization of **several applications** in an information system.

ARCHITECT PROFESSIONS

The previous page aims to summarize the different architectural professions that you may encounter today when working with architects.

Thus, I identified three professions, starting from the software architect, expert on his technology which allows to carry out the software architecture of the projects on which he is brought to work: he will decide on the various applicative patterns designs to implement on the project of which he must define the architecture, as well as the suite of tools to use in the ecosystem of the project.

Often, he is called upon to intervene on several projects in order to give his opinion on technical subjects and then lets the lead developers implement the solution and transfer it to their teams.

We then find the solution architect whose role is to identify the technologies adapted to meet the needs of projects, often complex and composed of several tools and/or technologies.

He may have to exchange with a software architect who can give his point of view as a technical expert on his technology.

In a larger information system context, the enterprise architect allows to keep a coherence between the different projects in order to, for example, re-use a software brick implemented in a first project.

It is also responsible for providing the tools that will best suit the projects in its application park. It's always better when you can share the same tool, the same license, for all applications.

As the roles are not always clear, we will now go into more detail on the three architectural roles identified according to the different project phases presented previously in order to describe their typical scope of intervention.

Of course, depending on the companies and project contexts in which you may work, these perimeters may vary. The architect may also have a tendency to be sucked in as a developer by the team and therefore put aside his important role. During this time, he has no time to devote to his tasks and it is often difficult to get out.

It is therefore crucial to clearly establish the architect's role with all project teams from the outset so that it is properly perceived.

This is not a reference but rather a broad vision to help you better understand the roles of each on a daily basis.

Let us take here the example of a software architect based on PHP technology.

→ Start-up phase

During this phase, its objective will be to:

- Define the good practices to use on the project: the respect of PSR rules, good Symfony practices, designs patterns to use, etc....
- Define the design patterns adapted to the different needs of the application.
- Define the procedures and tools for the industrialization of the project: unit tests, functional tests as well as the delivery (deployment) of the application.
- Ensure that each developer has a good technical understanding of the project and follows good practices.

→ Development phase

Once development has started, its objectives will rather be to ensure that the application is fully operational and functional.

He will then have to ensure that the technology and possibly the framework selected are used correctly by the teams. The application must be fully functional and operational.

The rendering to the end user must be correct, which implies a good run on the server side but also that interactions with third party tools like databases are not a problem.

Finally, it is essential to ensure that no security holes are exposed by the application.

→ Evolution phase

Once the project is in production, it must anticipate the application's new functionalities so that they can be implemented without technical debt. It is also the right time to simplify procedures that have become too tedious to maintain.

As technologies evolve over time, its role is also to anticipate migrations to new versions of the technology, framework and tools present on the project.

SOLUTION ARCHITECT

The solution architect must respond to more varied problems on projects: he is not necessarily an expert on a particular technology but must have a good visibility on all available technologies.

→ Start-up phase

At the start of the project, he has the complex task of choosing the most appropriate technologies to best meet the need, but also to be careful to have the human resources necessary to use these technologies.

Once the technologies have been selected, it should also describe the different hardware resources required for each technology.

He can of course rely on the help of a software architect to obtain more precise answers on a technology.

→ Development phase

Once development has begun, the solution architect ensures that the technical teams do not encounter any problems in using the various software components selected. He then takes feedback from the teams and it may happen that he questions a technology: this is not the ideal case but it is necessary to know how to adapt and solve problems before they happen.

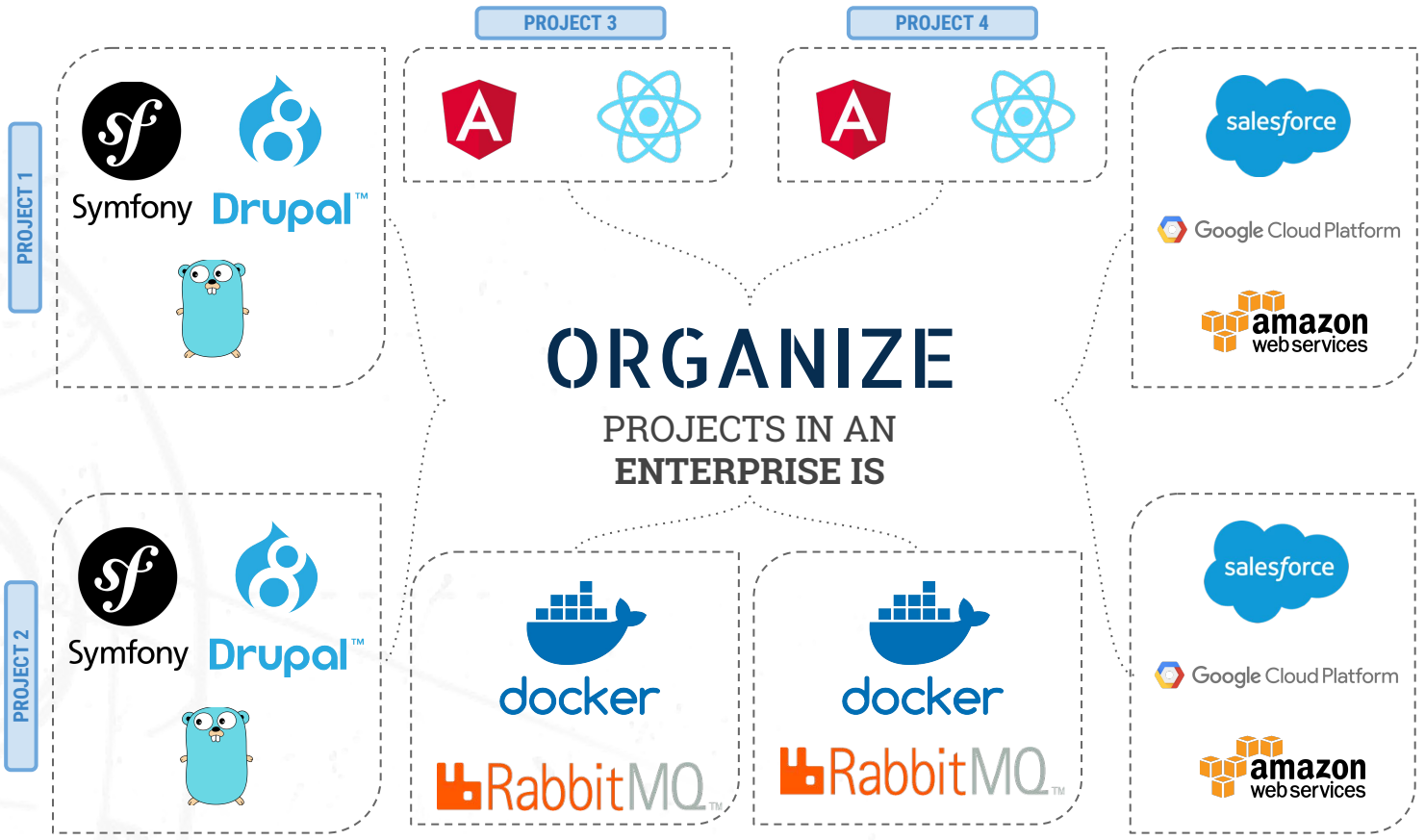
He will then audit the platform under development in order to detect any security flaws, performance concerns, data flow problems between the bricks, so that they can be corrected as soon as possible.

→ Evolution phase

Our project is now in production, it must take advantage above all to verify that the versions of the tools used are up to date in production in order to maintain good stability. If necessary, a migration plan must be carried out with the technical teams in order to migrate to a new version that would not be backward compatible with the current version.

The main objective of this phase is to prepare the future evolutions on the application, which will potentially require new tools and/or technologies and to make evolve the material resources in order to take into account these new needs.

ENTERPRISE ARCHITECT



The role of an enterprise architect is rather to keep a visibility on all the projects developed in the company in order to guarantee a coherence of the technological choices.

He is responsible for the global mapping of the company's information system. This allows him to ensure that a first application can for example interact with an already existing second application, rather than redeveloping an identical solution.

Having an eye on the cartography also means that it has a good visibility on the data exchanges carried out in all the operating system and, more generally, on the core business of the company, which it must maintain coherent via the technology and be able to make evolve.

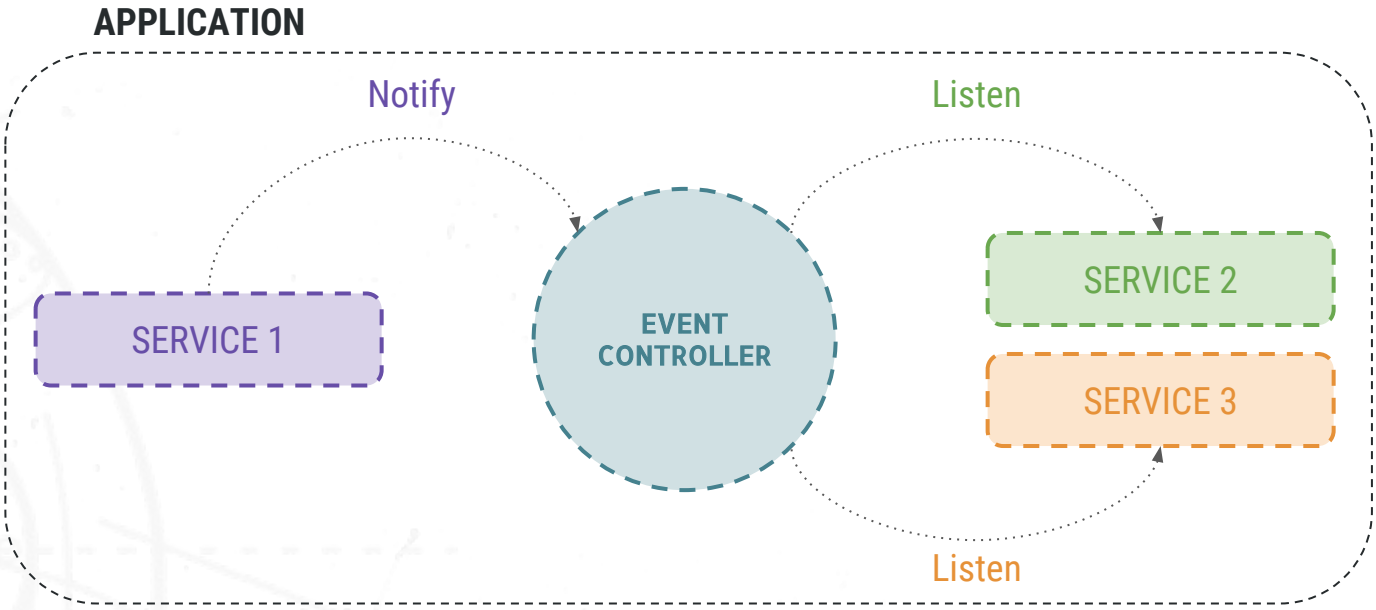
It must also identify third-party solutions that meet the needs of most applications and have common bricks between them, which avoids making mapping more complex.

We are thinking here, for example, of an ERP solution, tools to monitor applications in production or tools to provide business data analysis (BI).

It collects feedback from solution and/or software architects in order to evaluate the risks and technical impacts of architecture choices on all applications in relation to the existing information system. He can thus make his choice of solution.

The enterprise architect differs from software and solution architects because he manages the complete architecture of an information system, and is not restricted to its technical aspect only.

SOME APPLICATION DESIGN PATTERNS



Let's start with a simple design pattern and at the same time very often implemented in different frameworks today.

It allows to develop applications whose application code reacts to events, which allows to better break down its code into business services, the goal being to avoid mixing different business processes.

The idea is that in case a **service A** would trigger an event (change of state, for example), **services B** and **C** would connect to this event in order to take into account the change and thus execute the code portion that concerns them.

The whole is controlled by a central element allowing the good trigger of the services to the good events.

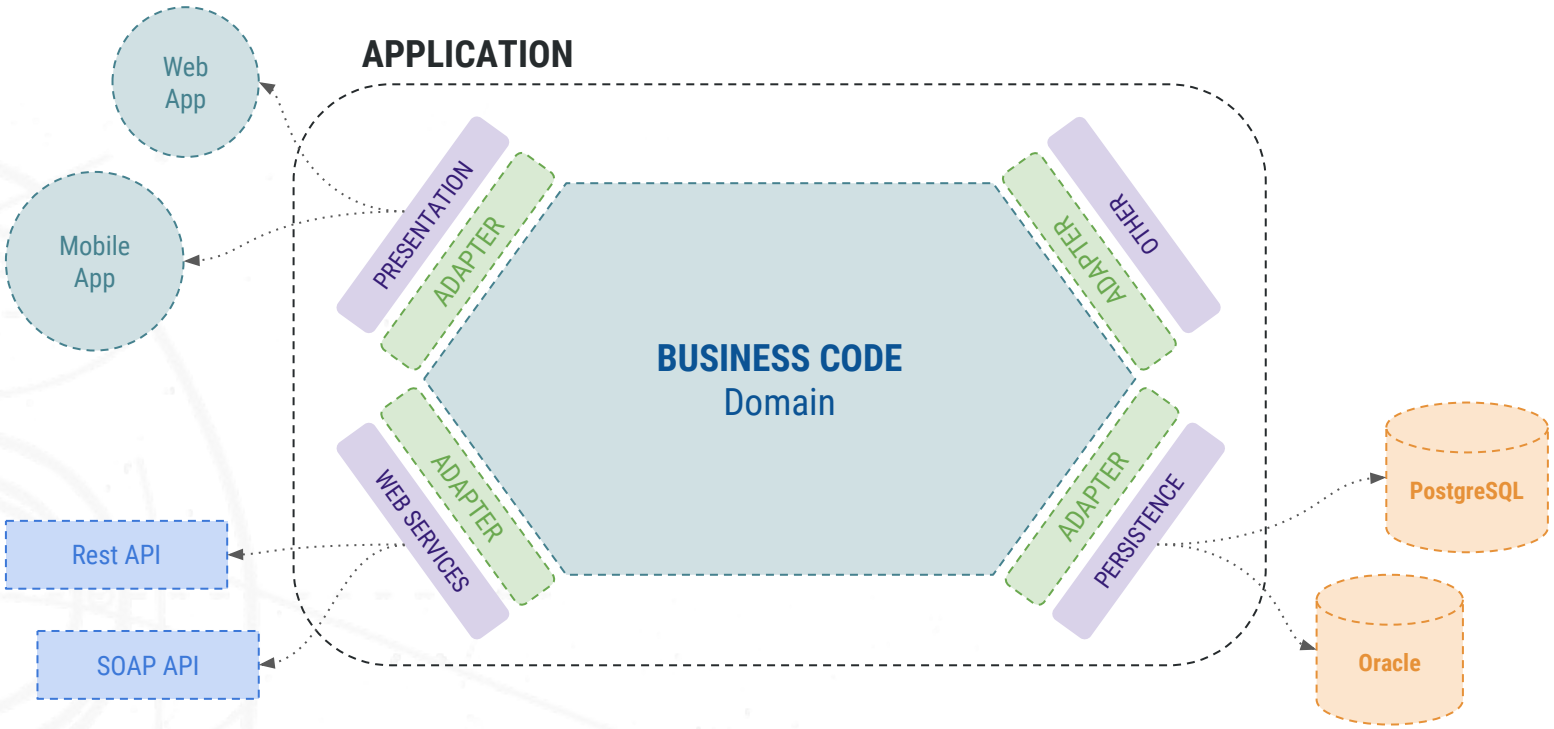
→ Pros

- Service-oriented code: each portion of business code is cut into a dedicated service, which reacts to event updates
- Code easier to test: it is indeed enough to test each portion of service individually
- Less code coupling between services
- Resilience to failure: if a service A contains a bug, a service B can still be executed correctly

→ Cons

- In some contexts without business complexity, using this pattern can complicate understanding the code
- This pattern only passes one object: you must couple the data if you have several objects to pass through

HEXAGONAL ARCH.



Application architects may sometimes have to use more structuring patterns for the application. This means that the source code must be organized in a much more structured way.

The hexagonal architecture pattern, for example, greatly structures your application code by centralizing business code and developing abstraction layers.

Thus, you will have to write abstraction layers for the following elements :

- communicate with databases (persistence layer);
- make different views such as web and mobile rendering (presentation layer);
- accept several types of access by exposing REST, GraphQL or SOAP web-services.

→ Pros

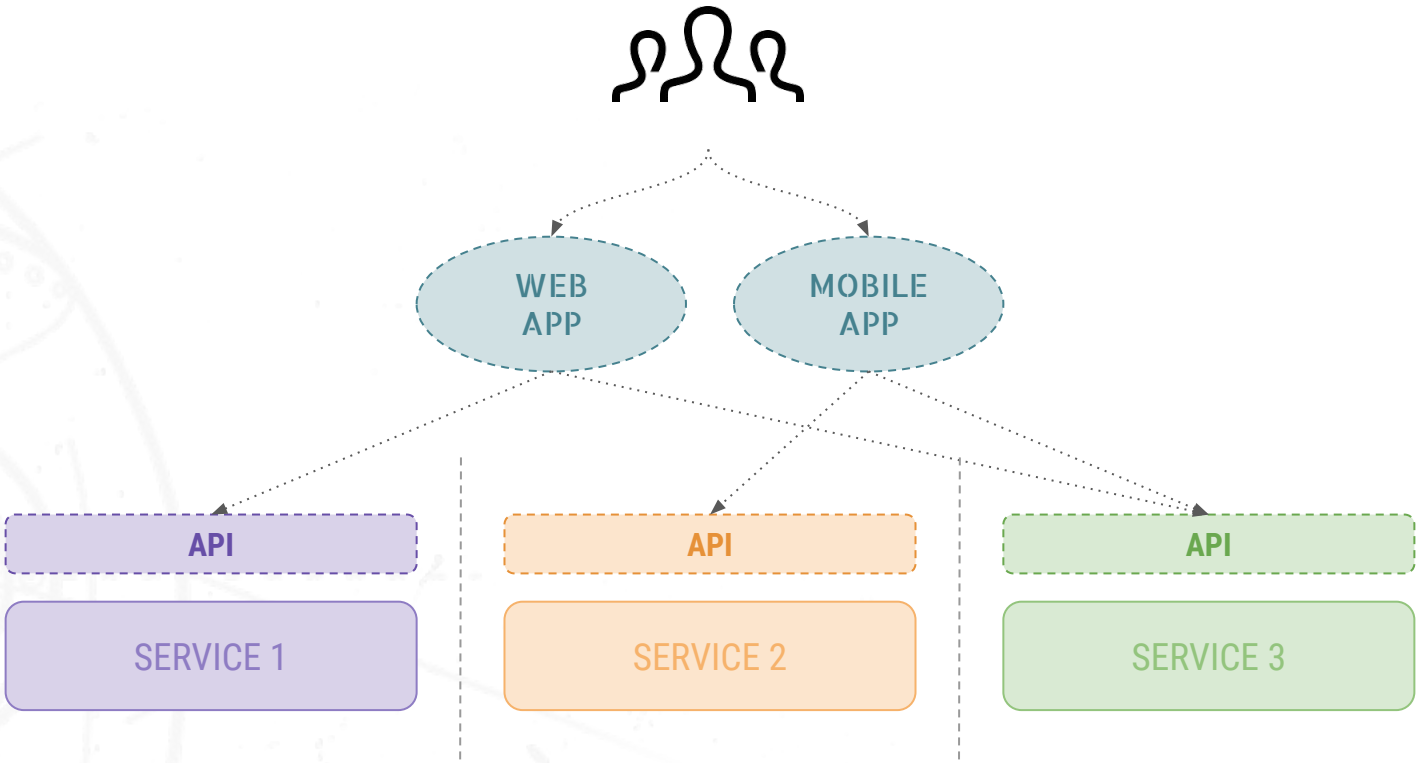
- Allows to keep the business code completely agnostic of the different layers
- Allows you to focus on business code when a feature needs to be developed, without worrying about the rest
- Simply add an adapter, based on business interfaces to integrate a new service

→ Cons

- From an architecture point of view, this implementation can appear complex at first glance
- YAGNI[1] (You Aren't Gonna Need It): the implementation of this architecture pattern requires the need to set up several layers of abstraction, at different levels, so that its choice is perceived as beneficial

1 https://fr.wikipedia.org/wiki/YAGNI

SOME SOLUTION DESIGN PATTERNS



Previously, many applications were developed in so-called "monolithic" mode.

This means that all requirements were centralized in one application and under one technology.

With the advent of cloud hosting and containerization, it is now easier to break these applications into several independent applications (services): we then speak of microservices.

In concrete terms, a micro-service responds to a particular business need, for example: taking care of all invoicing management.

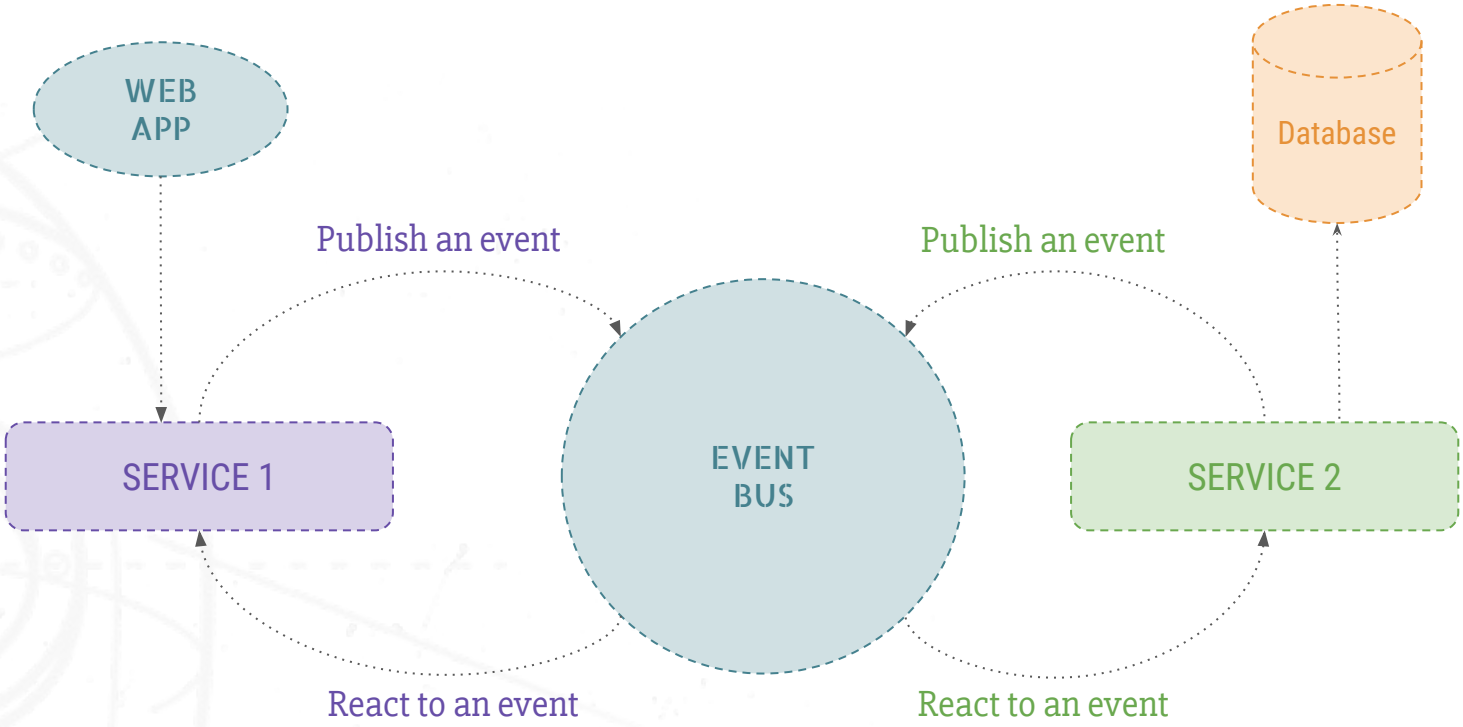
→ Pros

- A business need corresponds to a project: the code within each project is therefore concentrated to meet a single business need
- Ability to frequently deploy one service without impacting others
- Ability to deploy additional servers for a single service only (scalability)
- Possibility to use a different technology for each service

→ Cons

- Need to spend more time on industrialization and deployment tasks as they vary from one department to another
- It is necessary to be able to manage communication errors between services with the implementation of systems such as a breaker circuit, fallback, retry or other systems.
- It is necessary to take into account the concerns of degradations of network performances: to set up a system of election, data synchronization, ...

EVENT DRIVEN



As is often the case with micro-services, services must be able to communicate with each other, or at least to react to the events of others.

If you imagine an online order taking micro-service, when it receives a new order, a billing management micro-service must take over to generate an invoice.

With an event-oriented pattern, the order taking service will here publish an event in an application bus (type Kafka[1] or RabbitMQ[2]) and the billing service will then be able to subscribe to this type of event and react accordingly.

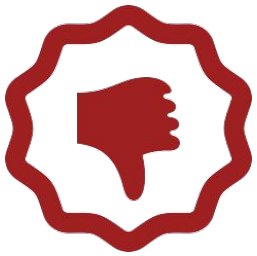
1 https://kafka.apache.org
2 https://www.rabbitmq.com

EVENT DRIVEN

→ Pros

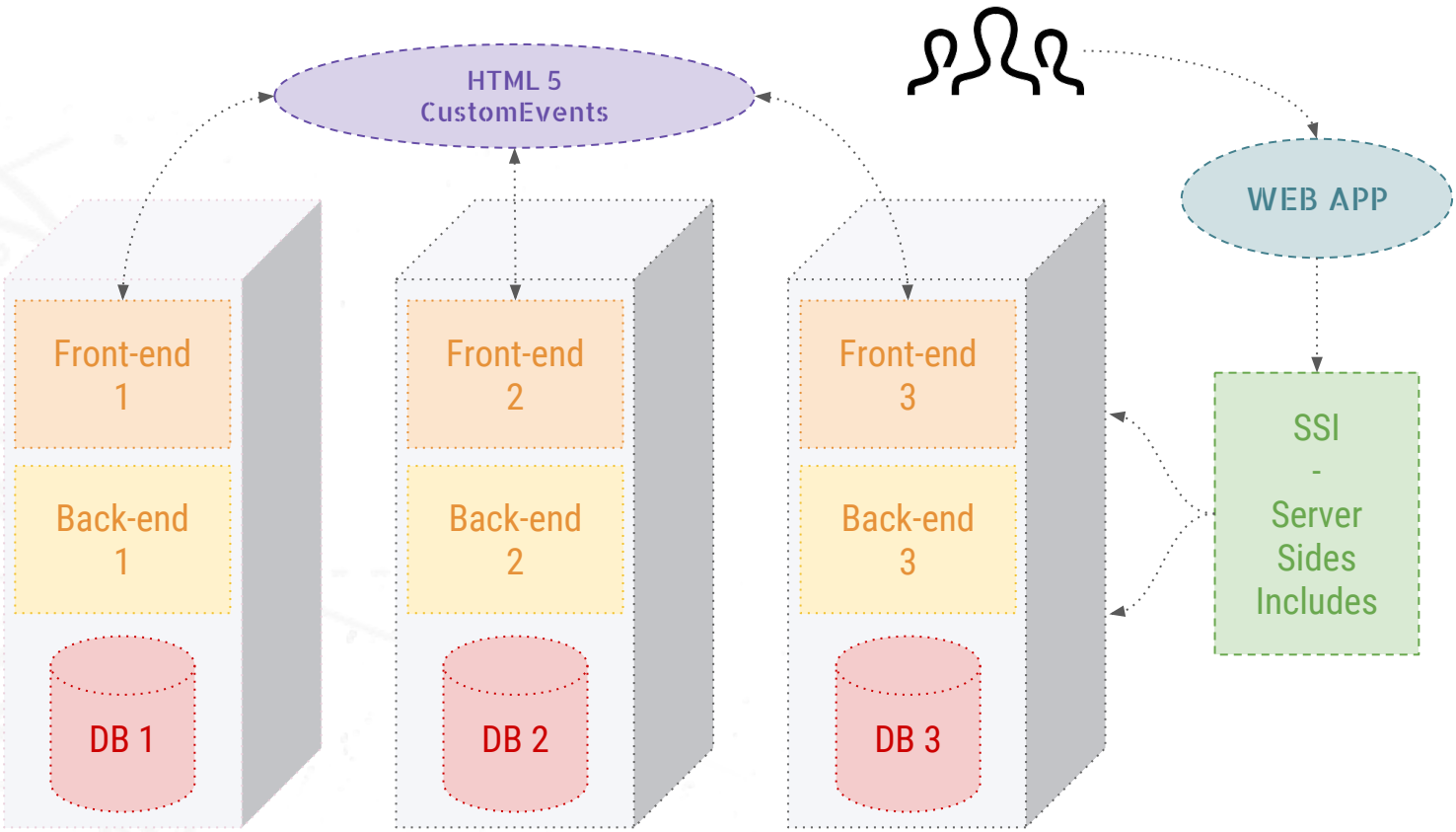
- This pattern makes it possible to clarify the exchanges: the application bus centralizes indeed all the flows of events which can take place
- Each service may freely subscribe to the notifications that interest it
- Increased scalability: it is possible to increase the consumption resources of a particular event in case of heavy traffic, rather than altering the entire application

→ Cons



- Care must be taken to avoid potential loss of events / messages / status changes. Requires retry and network error mitigation as much as possible
- Need to document the solution as the number of services or features grows
- Also, it is important to have a clear naming nomenclature to quickly identify applications and actions impacted by an event

MICRO FRONTENDS



In order to go a little further on micro-services, especially on front-end technologies, it is also possible to implement this type of pattern, which will make it possible to split a front-end project into several business needs as well as several development teams.

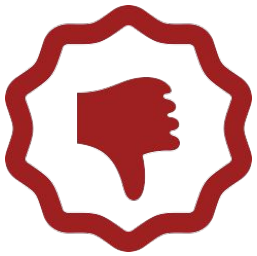
Although particular, this pattern can be useful in some professional contexts.

This pattern allows to include several front-end projects on the same page thanks to HTML 5 "Server Sides Includes" and uses HTML 5 "CustomEvents" to make them communicate with each other.

→ Pros

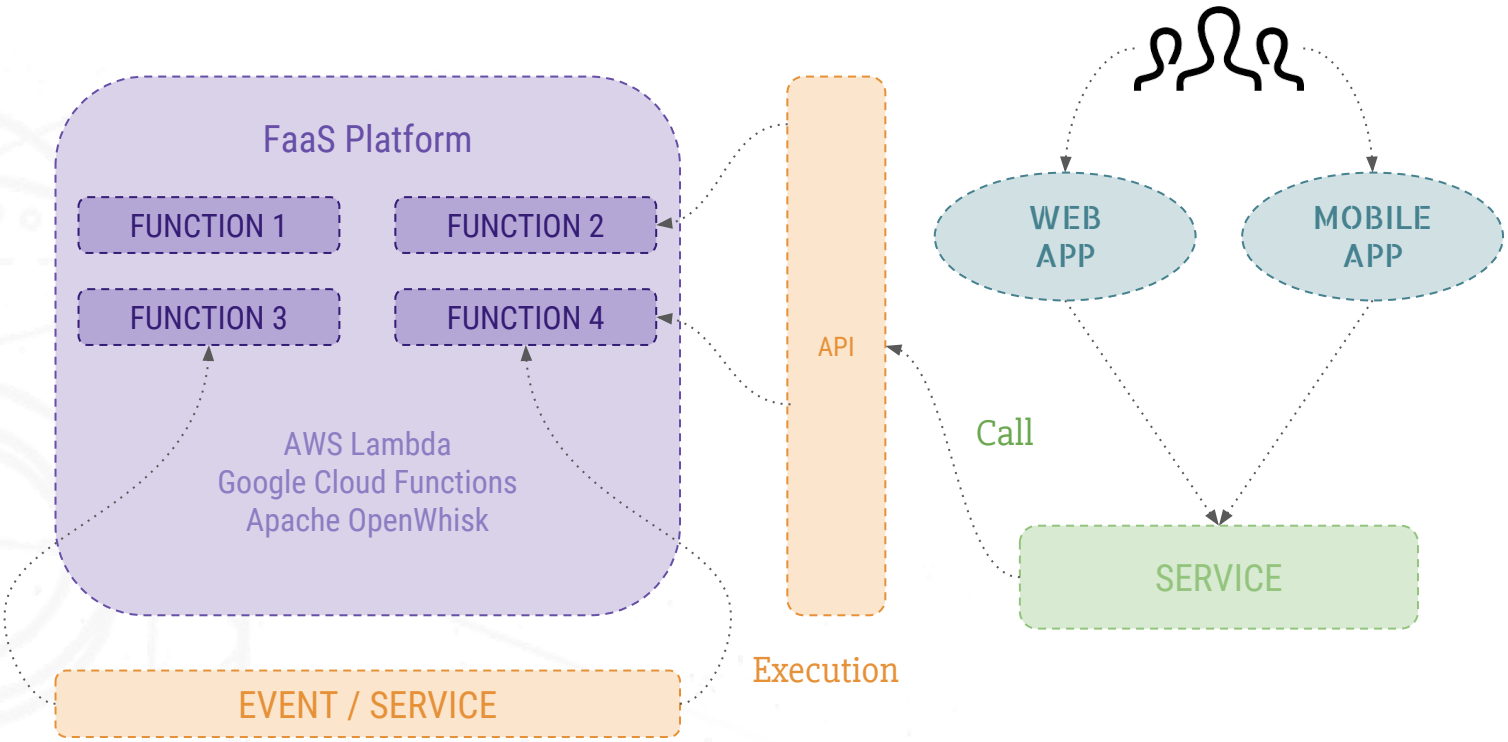
- Allows more flexibility: feature development and deployments are separate for each service
- Bug risk reduction because the perimeter of each service is controlled
- Functional tests are easier to perform and are broken down by functionality

→ Cons



- Complexifies feature development and deployment workflows
- Tendency to reuse third-party components (dependencies) in different services, creating code redundancy
- The initialization time of the application can be increased because it is necessary to load the various micro-frontends by making requests on the SSI server side (Server-Sides Includes)

FAAS - SERVERLESS



After seeing some patterns that allow you to cut your application code into business service, the trend today is to "Function as a Service" (FAAS), also called "Serverless". Don't get me wrong, your code will be executed on a server, but the hosting platform would abstract you from it.

The idea is to trigger functions (code portions) each executing a particular action: adding data to a database, recovering data, for example.

These functions can be triggered by API calls but also by events (periodic or not) on your platform. For example: when sending a file to your storage server.

FAAS - SERVERLESS

→ Pros

- No system administration required: this is fully managed by your platform, you only have to deploy your functions
- Reduced infrastructure costs because resources are used only when functions are triggered
- Reduce application complexity by dividing code into business functions
- Fast to develop: it is possible to obtain an MVP[1] quickly

→ Cons

- The functions are limited in time (often 5 minutes) and in memory consumption therefore not suitable for long or very heavy treatments
- A short cold start time is required when the function has not been executed for a short time (a few tens of milliseconds).
- You depend on your platform: the trigger functions are specified on the platform on which you are developing

1 Produit minimum viable pour la sortie en production

Eleven labs

TO CONCLUDE

INFORMATION RESOURCES

Eleven labs

In order to deepen the different patterns seen in this book, I invite you to visit the following websites:

→ Pattern: Microservices

Microservices.io: http://microservices.io

→ Pattern: Event driven

The Reactive Manifesto: https://www.reactivemanifesto.org

→ Pattern: Micro frontends

Techniques, strategies and recipes: https://micro-frontends.org

↳ SCS : Self-Contained Systems

Independing systems: http://scs-architecture.org

THE FINAL WORD

I hope I have been able to demonstrate, through his role, his interactions with the different teams, the concepts he must defend on his projects as well as the various patterns he may have to put in place that the architect plays an important role in the success of web applications.

Indeed, it makes it possible to articulate all the components of a project so that everything goes as well as possible at the time of the setting in production but especially that the project remains evolutionary thereafter and that it is not necessary to rebuild everything because the application crumbles under the bugs or that it simply was not thought so as to be easily maintainable and evolutionary.

I also emphasize that all concepts cited in this book must be known to every developer. It is very important to always keep in mind the performance and security of the application (runtime/responses, data transferred, etc...).

Finally, the few design patterns presented in this book are only a small sample, I invite you to discover others because it will allow you to cover a broader scope of technical answers.

If you want to know more about software architecture, design patterns, when to implement them and if in your case this is useful or if you have any questions, I would be happy to answer you via :

- E-mail : architecture@composieux.fr
- Twitter: @vcomposieux

If you would like more assistance, do not hesitate to contact Eleven Labs to study your projects.

ACKNOWLEDGEMENTS

This white paper is written and edited by Eleven Labs.

Eleven Labs
15 Avenue de la Grande Armée
75116 Paris
France

www.eleven-labs.com

contact@eleven-labs.com
+33 1 82 83 11 75