AsciiDoc Article Title

Table of Contents

Glossary . . . 3
The What and Why of Beep and the microservices architecture . . . 4
Current state of Beep . . . 4
Microservices architecture briefly explained . . . 5
Benefits . . . 6
Drawbacks . . . 6
Why Beep should consider migrating to a microservices architecture . . . 7
I. Separating Beep into functional neighborhoods and microservices . . . 7
1. Identifying user stories . . . 8
User stories currently in Beep . . . 8
Important user stories that are not yet part of Beep . . . 9
Use case diagram of Beep's current user stories . . . 10
2. Some rough estimates of Beep's expected loads and limitations . . . 10
Estimates of averages . . . 10
Resulting calculations and requirements . . . 10
3. Separating Beep's user stories into functional neighborhoods, contexts and services . . . 11
Identifying functional neighborhoods and bounded contexts . . . 11
Service separation proposal . . . 13
Component diagram for Beep's separation into microservices . . . 14
4. How Beep can be migrated to a microservices architecture: breaking down the monolith . . . 15
II. Managing the authentication system with OIDC . . . 17
1. Authentication requirements . . . 18
Authentication requirements . . . 18
Authentication constraints . . . 18
2. OpenID Connect, OAuth2 and Beep . . . 18
3. Comparing available technologies . . . 18
4. Authentication system design and implementation with Keycloak . . . 19
4. Keycloak as an OIDC provider . . . 19
5. Microservices architecture re-design considering Keycloak integration . . . 20
Why an api gateway/why network zones/network zones separation . . . 21
How Keyclaok handles identification+authentication+authorization . . . 21
Connecting Keycloak with surrounding authentication mechanisms . . . 22
III. Enabling communication between services . . . 24
1. Inter-services communication in a microservices architecture . . . 24
2. Identifying needs and constraints . . . 25
Accounting for considerations in inter-services communication . . . 25

3. Comparing available technologies . . . 25
4. Communication framework implementation with gRPC . . . 26
IV. Authorization system design and implementation . . . 29
V. Tracing logs and queries . . . 32
VI. Setting up a production ready system . . . 32
VII. Infrastructure security . . . 34
VIII. Search engine design and implementation . . . 34
IX. Managing platform security issues . . . 35
X. Integrating UI applications . . . 35

My guide to this TAD

Chapters (==) "In the following sections, I/we will..." "The following sections concern..."

Sections (===) "In this section, I/we will..." "The following section concerns..."

Pronouns

TODO: choose the right pronoun to use between "I" and "we" for the whole document.
https://www.writing-skills.com/knowledge-hub/business-report-writing/can-you-use-i-we-in-report
Maybe mix the two. "I" for when I suggest something, and "we" for when we dive into why I chose this (benchmarks, comparisions, I include the reader.s).
A suggestion is personal, but a choice is shared. As well as the reasons that led to this choice.

⇒ Use "I" for personal suggestions, "we" for choices and explanations/reasons; where the reader should be included.
⇒ TODO: make sure this rule is applied and respected throughout the document.

Chaque choix doit être argumenté et justifié objectivement:

- POCs,
- Nombres réels (benchmarks, comparaisons, estimations des différentes limites de perf/réseau/charge/ressources... etc.)
- Comparaisons avec d'autres solutions, avantages et inconvénients objectifs (perfs, DX, future proofness de la solution (modularité, scalabilité, extensibilité...), etc.)
- Sources citées à chaque fois que nécessaire
- ... (trouver plus de règles pour mieux définir ce framework à appliquer sur le TAD. Voir sur internet. De mêmes pour les autres groupes de règles à appliquer sur le TAD.)

Appliquer ces règles autant que possibler.
Vérifier que ces règles sont appliquées sur tout le document.

Under every question:

- Use What Why How to distribute the text ideas & knowledge, if and when it makes sense. You may need to break it down further.
- Visuals (schema, graphs) (at least one)
- Make a PoC where asked, and when relevant. It must provide a real example of the implementation, and useful results such as performance metrics, usability, etc.
- Use real data / estimations to support choices and motives. This is real design, using real cost metrics of time, capacity, resources etc.
- Cite your sources at the end of the question/chapter/section(?)

Instructions/TAD directives:

We are looking to migrate BEEP to a microservices architecture. The specifications are the same as described during all previous iterations of beep.

You will need to answer 10 questions:

- Via the redaction of the TAD (components architecture, sequences diagram, deployment architecture, etc.)
- Via the production of POCs (when mentioned)
- For your schema, use draw.io
- Redact your report using asciidoc

You are forbidden to propose or consider the following technologies:

- Messages queue
- CQRS
- Event Sourcing

POCs:

poccer en Rust. Peut être en Go pour certains
⇒ Prendre plus la main dessus (pour le stage) + Avantages et inconvénients détaillés au niveau des examinations des POCs.

Glossary

In alphabetical order.

"The purpose of a glossary is to provide definitions for words or phrases that may be unfamiliar to

the reader, or that have a specialized or technical meaning within the context of the document."

- **Use case**: A scenario for a critical user journey / etc. business etc. WIP.
- **Business capability**: A business capability is a capacity a business or organization has to perform core functions, fulfill its objectives and responsibilities in a particular domain.

The What and Why of Beep and the microservices architecture

In the following sections, I will provide a brief overview of Beep, why it exists, its current architecture, what a microservices architecture is, and why Beep should consider migrating to a microservices architecture.

Current state of Beep

Beep is a platform for sharing and communicating with friends communities. With Beep, you have instant messaging, file sharing, real-time voice and video. It’s entirely free, and the code is soon to be open sourced.

We first started Beep in January 2024 as a school project. It currently runs on our school’s infrastructure. What’s neat about Beep is that there are no paid features, lots of features get added from time to time, you’ll soon be able to read the code and contribute; and most importantly, if we had a community to listen to, we probably would!! Maybe. Probably.

The below image is a rudimentary overview of Beep’s current architecture, as of March 2025.

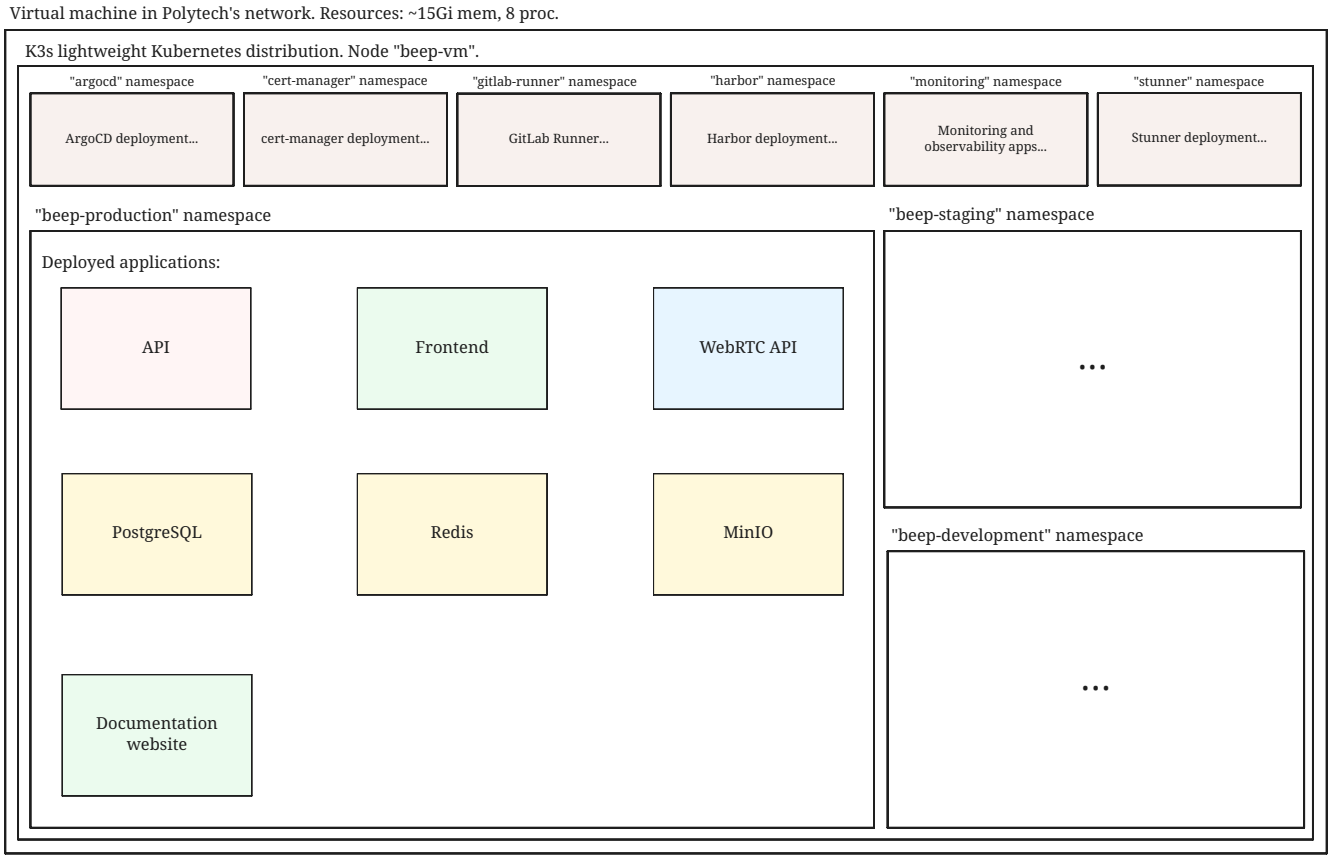

Figure 1. Rudimentary overview of Beep's current architecture, as of March 2025. Made with excalidraw.com.

The purpose of this schema is for you to have a quick understanding of the curent the components and their interactions. Colours, shapes and sizes have no particular meaning.

Currently, Beep is deployed as a set of monolithic services and auxiliary tools (such as observability tools, registries, etc.) replicated on a single-node K3s cluster. The control plane is also part of this node. The K3s Kubernetes distribution is configured (by default, as is here) to use an embedded SQLite as its data store; and Flannel as a layer 3 network fabric/CNI plugin.

The frontend and backend are monolithic, and the database is centralized. The monitoring and observability services are also centralized. The whole system is deployed on Kubernetes, and the infrastructure is hosted on a VM or a proxmox cluster. Namespaces are used to separate the services by function.

An important part to bear in mind throughout this document is Beep's current database s

Microservices architecture briefly explained

Following the goal of this document, in this section, we will focus on a concise explanation of what a microservices architecture consits of. In later sections, we will go through why Beep should consider migrating to a microservices architecture. The following chapters of this document will focus on what needs to be done to migrate the architecture, and how to do it.

A microservices architecture is complex to design and implement properly. This is why it is critical that everyone on the team has a common understanding of a microservices ecosystem, which is the goal of this section.

The microservices architecture is an architectural style or pattern, which follows four main principles:

- Services are (typically) organized around business capabilities (see glossary),
- Each service is owned by its own team.,
- Each service is independently deployable,
- Services are loosely coupled.

These principles make up the basis of the microservices architecture. There's a lot to it, it's got plenty of advantages as well as disadvantages. In short, it's not a miracle solution.

Benefits

- Simple services: "microservices" are only "micro" in terms of their single business capability provided, not in size. As such, they are simpler than the whole, easier to understand, maintain and upgrade. The size of a service matters least in its definition.
- Team autonomy: each team can own one or a few services. The microservices architecture enforces team autonomy and independence: services are independently developed, tested, managed and deployed (,the whole lifecycle is managed independently). They run independently from each other. As such, teams can work independently of each other, cutting down on the time it takes to manage intertwined teams, reducing the overhead for the teams leader.
- Faster CI/CD processes: services independence also cuts on the time it takes to build, test and deploy each service, thanks to their relative size compared to the whole.
- Support different paradigms per service: service independence allows each team to chose different paradigms freely - such as using a different technology stack, which is managed and maintained by this team.

More on Why microservices?: (martin fowler breaking the monolith article)

> "The ones who embark on this journey have aspirations such as increasing the scale of operation, accelerating the pace of change and escaping the high cost of change. They want to grow their number of teams while enabling them to deliver value in parallel and independently of each other. They want to rapidly experiment with their business's core capabilities and deliver value faster. They also want to escape the high cost associated with making changes to their existing monolithic systems."

> "Microservices have independent lifecycle. Developers can build, test and release each microservice independently."

Drawbacks

- Complexity: a distributed architecture is more complex than a monolithic one on every level:

networks, observability, latency, authorization, authentication, service coupling… - all needs to be rethought and managed in a distributed environment, and new problems arise. The architecture itself is difficult to properly understand and design.

- Some distributed operations might involve tight runtime coupling between services, which reduces their availability.//
- Risk of tight design-time coupling between services, which requires time consuming lockstep changes//

A microservices architecture doesn't suit all needs, and it takes a lot of effort to properly design a suiting microservices architecture. Initial design or redesign decisions may lead to unforeseen consequencies in the future, that may be difficult to correct afterwards.

Why Beep should consider migrating to a microservices architecture

In this section, I will explain why the Beep team should consider migrating to a microservices architecture.

As students, migrating Beep to a microservices architecture is a very rewarding exercise. But more than that, it's a necessary step if we expect Beep to be able to accomodate hundreds, if not thousands of users and still be in control of our applications and infrastructure.

Considering our currently small team, moderate resources and recent concerns, what I believe Beep needs most from the microservices architecture is mostly better fine-grained control over scalability, security and observability between and around our deployed workloads, but also to enable us to add features more seamlessly, without potentially impacting the whole system's integrity.

I. Separating Beep into functional neighborhoods and microservices

The following sections concern my proposal to separate the Beep application into functional neighborhoods and microservices.

Before we can separate Beep into services, we need to have an idea of how Beep's functionalities can be grouped into autonomous business units/ functionality neighboroods.

1. For that purpose, we will first identify the user stories that define how people interface with Beep's functionalities.
2. Then, building on that, we will classify the identified functionalities of Beep into logical groups, or business capabilities/functionalities neighborhoods.
3. Lastly, from these logical groupings of functionalities, we will propose a separation of Beep into functional neighborhoods and microservices.

1. Identifying user stories

In this section, I will identify the user stories that make up Beep's functionalities.

TIP	A user story is an informal, natural language description of feature from the perspective of the end user. It typically follows the format: "As a [type of user], I want [some goal] so that [some reason]".

User stories currently in Beep

In the following table, the following subjects are identified:

- Guest: a user who does not have an account on Beep, or a logged-out user.
- User: a user who has an account on Beep, and is logged-in.
- Member: a user who is part of, and connected on a server on Beep.
- Authorized member: a user who is a member of a server on Beep, and has special permissions.
- Beep admin: a user who is an administrator of Beep, part of the Beep team.

As a	I want to	So that I can
Guest	Create an account	Use the features of Beep.
User	Create a server	Grow a community around it.
User	Join existing servers	Become a member of public and private communities.
User	Explore public servers	Become a member of the public communities of my choosing.
User	Join private servers on invitation	Become a member of private communities I'm allowed to join.
User	Have quick access to the servers I'm a part of	Easily switch between communities and user groups to interact with.
Member	Get information about my account	Review my account information.
Member	Update my account information	Modify my personal information, recovery mechanisms, my authentication mechanisms, language and other information relative to my account.
Member	Get more information about other users in a server I'm a member of	Better interact with them, and send them friend requests to chat in private messages.
Authorized member	Configure a server I'm a member of	Manage the server settings, appearance and user roles.

As a	I want to	So that I can
Authorized member	Generate a time-limited invitation link to a server I'm a member of	Invite other users to join the server.
Authorized member	Create and manage roles in a server I'm a member of	Allow or restrain other users to do specific actions or see specific channels
Authorized member	Create and manage channels and categories in a server I'm a member of	Group discussions by topics or themes, manage the visibility of these discussions, and pin messages.
User	Send messages, including markdown text and emojis, files and images, links with interactive previews	Interact with other users in channels inside servers or in private messages.
User	Mention other users in messages	Notify other users.
User	Share my voice, video and screen with other users	Communicate with other users with real-time voice, video and screen sharing.

Important user stories that are not yet part of Beep

As a	I want to	So that I can
User	Delete my account	Remove my account information from the apps and servers.
User	Get information about my account	Review my account information.
User	Update my account information	Modify my personal information, recovery mechanisms, my authentication mechanisms, language and other information relative to my account.
User	Receive push notification on my devices	Be notified of various events such as mentions.
Member	Search for users, messages or files in servers I'm a member of	Find back specific conversations or files.
Beep admin	Have full control over public servers	Ensure Beep's terms of service are respected across publicly accessible servers.

Use case diagram of Beep's current user stories

These user stories can be formally represented using a UML use case diagram. Below is the use case diagram describing Beep's current user stories, with the end user's different roles as actors, and the user stories as use cases.

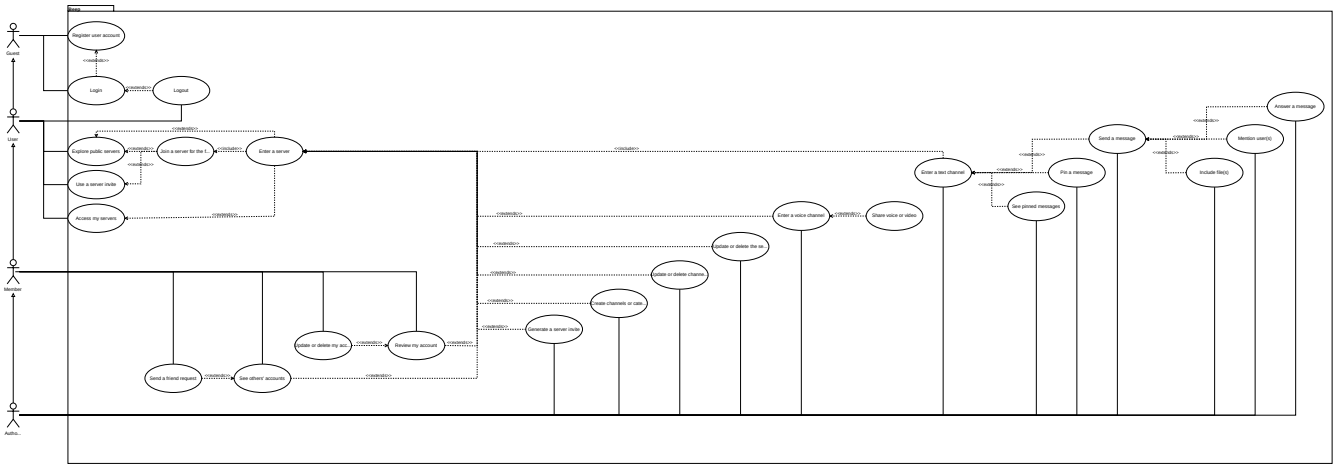

Figure 2. Use case diagram of Beep's current user stories. Made with draw.io

The purpose of this schema is to capture the requirements of Beep considering its users. This will help us outline business capabilities and user stories into functional neighborhoods, and thus redefine Beep into microservices.

A business capability is the expression or articulation of the capacity, materials, and expertise an organization needs to perform core functions.

2. Some rough estimates of Beep's expected loads and limitations

In this section, I will provide estimates to metrics that I consider important for designing a new architecture for Beep.

Estimates of averages

- Average daily active users: 300 000
- Average text message size: 200B
- Average shared file size: 1MB
- Average proportion of files to text messages: 1/30
- Average number of messages per user per day: 60
- Average duration of voice/video calls: 30mn
- Average bandwidth of voice/video calls per hour: 600MB

Resulting calculations and requirements

- Average number of files shared per user per day: 60/30 = 2

- Storage requirements:

From the preceding estimates, we can calculate the expected loads on Beep.

In the following, we will assume that these estimations apply.

3. Separating Beep’s user stories into functional neighborhoods, contexts and services

In this section, we will classify the user stories we identified into functional neighborhoods, and I will give my proposal for breaking down Beep into microservices.

Identifying functional neighborhoods and bounded contexts

A functional neighborhood is a group of functionalities that are related to a specific business capability of Beep.

Some common pitfalls to avoid when defining functional neighborhoods and cutting microservices are:

- Defining functional neighborhoods based on the current architecture, rather than the business capabilities of Beep,
- Defining functional neighborhoods that are too tightly coupled,
- Identifying microservices from the database schema, which is an anti-pattern so common it has a name: "Entity-Service Anti-Pattern". This results in distributed monoliths where each service becomes a CRUD wrapper around a table, leading to excessive coupling and communication between services, causing performance issues.

Generally, this comes down to not following the microservices architecture as we defined it earlier.

In order to avoid those pitfalls and more correctly identify which microservices should make up Beep’s architecture in a way that fulfills all of the user stories and business capabilities of Beep, we have to examine the current architecture, database schema, levels of dependency between components of the current monolith, and more. This will allow us to consider, and better avoid past architectural truths while designing the new architecture, which is to be based on services revolving around business capabilities. In order to do that, we will apply some principles of Domain-Driven Development, such as identifying bounded contexts.

We may start by identifying the main domains Beep revolves around.

- Authentication of users and user management
- Real-time communication and instant messaging and file sharing
- Management and retrieval of large quantities of files and other stored media

This first repartition of domains us a basic idea of what domains Beep’s business capabilities, and the user stories they resolve, mainly revolve around. However, we need to break down and regroup these domains further, avoiding tight levels of coupling between them and their components.

Some parts of Beep are clearly more independent than others. These can already be set astide into their own "contexts", such as the authentication system, the real-time communication system, instant messaging system, media storage system, search system (which would search accross for resources across other systems), and the notification system (which would notify users' devices on events happening from other systems).

These couplings are starting to make more sense. There are features in Beep that we didn't mention yet, such as server lifecycle management: settings, invitations; as well as roles and permissions.
The lifecycle of servers in Beep is currently pretty simple, but that means it's bound to have many features added soon. That means it's important to future-proof this system as of now, by thinking about the possible user stories and business capabilities that could revolve around it in the future.
Currently, server lifecycle includes servers settings (names, etc.) as well as invitation links. But in the future, this could include communities, groupings of servers, academic communities available for students via their academic emails - for example.
Entirely future-proofing a system is not an easy task, and would require a document of its own. However, some basic principles can be applied to make most of the future-proofing have effect. Building a smart and common abstraction over the current models, and avoiding repetition are such core principles.

As such, all of this could be considered to constitute its own "server lifecycle management" context.

And while we're structuring our services, let's keep in mind the common pitfalls we may involuntarily fall into. Especially the "Entity-Service Anti-Pattern", defined above.

We have identifies contexts around which to define Beep's future microservices architecture. To further refine these contexts, we can draw a "context map" to clearly identify the contexts around which services should be grouped.

https://medium.com/@mike_7149/context-mapping-4b4909cf195a Context map draft

- Authentication context (authenticating any request, external (users, bots/webhooks) or maybe even internal (authenticate services) (that's hypothetical for now)
- Authorization context (Manage RBAC, ABAC and PBAC. Apply policies between/across services to users queries/requests, etc.)
- Chatting context (messages, files, "text" channels lifecycle management)
- Video/audio calls context (handles all the media, real-time audio/video/screen sharing logic/features, and "voice" channels lifecycle management)
- Notifications context (push notifications to devices. At least that would need a message queuing…)
- User context (users and their settings)
- Search context (search for servers (overview page), channels, users, messages, files… Anything)
- Communities/servers context (manage communities/servers lifecycle, crud, invitations/… Potential future features such as groupings of servers as communities/…)

→ Check how do these contexts organize around teams, features, and how they'd interact with each other ; to make sure it's correct.

Service separation proposal

In earlier sections, we identified the user stories and functionalities beep must provide. We then classified them into bounded contexts following domain-driven development principles. From these previous analyses, I will propose a separation of Beep into microservices.

Service name	Goal and provided features
Authentication	Authenticate users and other external requests (such as coming from bot accounts, webhooks, APIs/SDKs, etc.), and possibly also internal requests(?).
User	Manages the lifecycle of user information and related data (such as pfp, username/email/password…, preferences in language and authentication mechanisms, etc.)
Chat	Manages text channel conversation and lifecycle with styled messages, file previews, etc.
Call	Enable users to share voice/video feeds with minimal latency in voice channels.
Search	Provide results to search queries for servers, channels, users, messages, files, etc.
Notification	Send, and manage push notifications sent to devices.
Media	Manage files (CRUD, fast retrieval, long-term storage in iceberg or else…).

Should authorization be implemented as a separate service? Or sidecar proxy component? Central or distributed policy store? Or etc. Will be seen in chapter IV.

If we take future features into account, we would add a "search" service, taking care of the search for users, messages, files and other elements across storage systems.

We can be tempted to separate Beep into microservices following the current SQL database's schema and relations, that we saw in the initial chapter. But this would be a mistake, since as we defined in the initial chapter, microservices should be organized around business capabilities; and the current database schema is not a faithful representation of Beep's business capabilities, as opposed to the user stories, business capabilities and functionalities their represent, that we identified in the earlier sections.
If we had defined Beep's microservices architecture from its initial database schema, we would end up with a distributed monolithic architecture, with tightly coupled services, a lot of inter-service communication, and thus scalability, and other benefits of a microservices architecture would be impacted.

Also doesn't make sense since databases will be separated into their own services and can be designed completely independently of the rest of the system. They become not part of the business capabilities of Beep, but really just tools to store and retrieve data/state for each service.

Also also, how can we have a notification service… Without a message queuing system?? We'd have to use some other way like… Mongodb… To manage the push notifs state… Ugh. It's just a hacky

hack, compared to a message queue. Or we'd assume that every device in the world who runs Beep is always on and has a stable connection to Beep (so that there would be no state to manage). OH I KNOW!! We get rid of mobile push notifications (so no need for kafka, or third parties), and for the browser, we use a websocket. I suppose it would work for browser notifications into desktop notifications.. Maybe??

Component diagram for Beep's separation into microservices

These functional neighborhoods/business capabilities/... can be formally represented using a UML component diagram. Below is the resulting UML component diagram describing my proposal for Beep's separation into microservices.

WIP

> "Deciding what capability to decouple when and how to migrate incrementally are some of the architectural challenges of decomposing a monolith to an ecosystem of microservices."

Methods to break down the monolith:

Decompose services by business capabilities : reflect organization behavior. → component diagram.
En partant du domaine (communications temps réel), on va créer les différents services. Attention aux services qui communiquent beaucoup entre eux : **combiner les services**.
https://12factor.net/
"The philosophy of Twelve-Factor turned out to be surprisingly timeless. More than a decade later, people still find its insights valuable, and it's often cited as a solid set of best practices for application development. But while the concepts remain relevant, many of the details have started to show their age."
Domain-Driven Development. Application est construite sous forme d'abstractions au dessus de notre modèle.

Obstacles à la décomposition : network latency, data inconsistency/interfaces, god classes (fait tout, dure à décomposer, big) and reduced availability. Énormément de contextes sont impactés.

Architecture bien définie avec le DDD, bounded contexts. Architecture (enables org & proc), organization (enables proc) et process de développement ⇒ rapid, frequent & reliable delivery of software.
1 service = 1 responsabilité. Un service rendu. N'est responsable que d'une chose à faire.
Guidelines :
Dur à faire mais génial : signifie une bonne séparation des services : c'est le **Common Closure Pattern**.
En gros, une règle business n'affecte que 1 microservice, pas deux ou plus (si cette règle évolue il faudrait modifier plusieurs services...)
Disons que j'ai à modifier le mode de livraison : que ça ne soit fait que dans un seul service !!!
Open Closed Princple : on veut pouvoir intégrer facilement d'autres contrats d'API sans avoir

à changer le coeur de fonctionnalités !!! → Réelle abstraction. Logique non liée aux contrats d'APIs. ⇒ Des interfaces/contrats d'API génériques (comme un filesystem : create, read, update, delete, open d'un objet générique (fichier) ET NON PAS un type de fichier spécifique. C'est toujours que des fichiers ultra génériques avec leurs mêmes attributs communs : nom, taille, permissions, etc.)
Quality of a service: scalable, reliable, secure, maintainable, testable, etc.
⇒ Implementable functionalities are dependent on the quality of the architecture design. Future-proof design, stays easy to update with features, good abstractions (see how good filesystems are :3)
Liskov principle, ouvert en extension fermé en modification (en gros les bonnes abstractions type filesystem)
Je veux utiliser une autre bdd. Ou en utiliser plusieurs. Service and repository pattern. Hexagonal architecture. Des adaptateurs (que l'on branche sur un port) qui permettent de réaliser des opérations qui soient indépendantes de ce qu'il y a derrière (mongodb, postgres, filesystem...!! C'est dans l'adaptateur qu'on définit ça)

https://www.uml-diagrams.org/component-diagrams.html https://developer.ibm.com/articles/the-component-diagram/

Component diagram for Beep's separation into microservices. Made with draw.io

image::images/fig3.component/beep-uml-component-diagram-light.svg[Component diagram showing Beep's architecture separat ed into microservices.]

Reading a component diagram: components are strictly logical, design-time constructs. The idea is that you can easily reuse and/or substitute a different component implementation in your designs because a component encapsulates behavior and implements specified interfaces.

This diagram is a first draft of the separation of Beep into microservices. This draft will be refined and completed further in the following sections, where we will take into account multiple other issues such as authentication, authorization, observability and many more.

(Authentication and) authorization (centralized, or decentralized store/policy agent? will depend on tech used. See corresponding chapter) services do not appear in this diagram will be defined in later chapters.

This component diagram only illustrates components and their interactions. It is not a proposal for inter-services communication, which will be studied in the next chapter.

Also there isn't the frontend, surrounding load balancer and API gateway, which allows the frontend to reach API endpoints that lead to different services, and authenticate the requests via JWT.

4. How Beep can be migrated to a microservices architecture: breaking down the monolith

Earlier, we saw an overview of Beep's current architecture, of the microservices architecture, and why Beep should consider migrating to a microservices architecture.

In the last section, we went over my proposal to separate Beep into functional neighborhoods and microservices.
In this section, we will briefly go over how the Beep team can break down the application into microservices.

In this section, we consider that the earlier chapters have been read and understood by the team, including the microservices architecture, as well as the separation of Beep into bounded contexts.

There are two main ways the Beep team could break down the monolith:

- The incremental way,
- Or a complete redesign and reimplementation.

A complete redesign and reimplementation may seem like a sound idea at first, but when studied with a critical eye on resources management and common operational principles, it becomes obvious that this choice is soldem the right one.

Limits in resources such as time, effort or hands would need to be much more than sufficient to entirely replace the current architecture at once. Maintenance and the rate of delivery of new features and on the current system would suffer greatly.

On the contrary, incrementally redesigning and reimplementing groupings of features as new services, bounded inside their own contexts, and responsible of their own technology stack and business capabilities, would allow the Beep team to gradually replace parts of the current monolith with services, rendering the operational management of resources completely feasible across the team, with minimal impact on the current system's integrity, management, or feature delivery lifecycle.

Moreover, by concentrating the efforts of some on the implementation of a new service, this approach may also be just as fast, if not faster than the former approach.

Sources for the chapter:

https://microservices.io/refactoring/ https://microservices.io/patterns/ https://microservices.io/patterns/microservices.html https://microservices.io/patterns/decomposition/decompose-by-business-capability.html https://microservices.io/patterns/data/saga.html https://microservices.io/post/refactoring/2019/10/09/refactoring-to-microservices.html https://microservices.io/post/architecture/2024/08/27/architecting-microservices-for-fast-flow.html

Gérer les migrations de bdd, les insertions de bdd dans une architecture microservices ? → Trino !! (? Piste à explorer). https://trino.io/blog/2020/06/16/presto-summit-zuora.html PS: Trino s'appelait PrestoSQL avant. https://moduscreate.com/blog/microservices-databases-migrations/ Paraît que marche encore mieux avec les trucs datalake, Hive, Iceberg…

use kafka for inter-microservices communication? https://www.youtube.com/watch?v=Vz2DHAHn7OU

Was told this is a good tutorial to understand async await (in rust at least, but maybe in general!): https://tokio.rs/tokio/tutorial https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/ https://stackoverflow.com/questions/60457740/rest-endpoint-for-complex-actions

https://stackoverflow.com/a/60463179

when is microservice not a good pattern https://dzone.com/articles/10-microservices-anti-patterns-you-need-to-avoid

- https://microservices.io/articles/glossary#dora-metrics
- https://martinfowler.com/bliki/BoundedContext.html
- https://martinfowler.com/articles/break-monolith-into-microservices.html
- https://leofvo.me/articles/microservices-for-the-win
- https://www.geeksforgeeks.org/how-discord-scaled-to-15-million-users-on-one-server/
- https://samarthasthan.com/posts/building-a-scalable-e-commerce-empire-a-micro-services-system-design-approach/
- https://freedium.cfd/https://medium.com/@samarthasthan/from-zero-to-millions-crafting-a-scalable-discord-with-micro-services-0e55e65f2a16
- https://microservices.io/patterns/decomposition/decompose-by-business-capability.html
- https://microservices.io/patterns/microservices.html
- https://microservices.io/articles/glossary#dora-metrics
- https://microservices.io/patterns/data/database-per-service.html
- https://microservices.io/patterns/data/saga.html
- https://stackoverflow.com/questions/60071074/microservices-dependencies-in-uml-diagrams
- https://www.uml-diagrams.org/component-diagrams.html
- https://developer.ibm.com/articles/the-component-diagram/
- https://www.edrawsoft.com/fr/article/microservices-architecture-diagram.html

WIP

Listen to DIS https://www.youtube.com/watch?v=rv4LlmLmVWk

POUR LA BDD DE MESSAGES : MONGO!!! car relationnel déjà pour le search… Mais aussi pour quand ds le msg y'a une image/un msg vocal, ne pas avoir 90% de champs null dans la postgres pour tous les msgs qui en ont pas… Et si y'a, dans le mongo, mettre l'url du file service/object storage/CDN/FTP…

II. Managing the authentication system with OIDC

The following sections concern my proposal to integrate Open ID Connect authentication to Beep.

1. Authentication requirements

In this section, we will discuss the functionnalities and requirements we need from Beep's authentication mechanism, as well as possible constraints.

Authentication requirements

Beeps needs to provide the following authentication mechanisms:

- User can connect to Beep via their Polytech account.
- User can connect to Beep via their Google account.
- User can associate their Google and Beep accounts.
- Single-sign on (SSO), allowing users to connect via their Polytech account or Google account.

To provide these authentication services, Beep needs to integrate the Open ID Connect (OIDC) protocol, which standardizes authentication, identification and authorization processes between systems that provide authentication. Polytech uses the Lightweight Directory Access Protocol (LDAP), which is why we can't associate a Beep account to a Polytech account???? First of all, the authentication system will be managed by Keycloak, an identity and access management platform

---

OIDC will be Keycloak A user can log in with his Polytech account (⇒ Polytech LDAP access via OIDC - to be taken into account in your deployment scheme) A user can associate his user account with a Google account

Needs: link Google auth, Polytech auth, Beep auth (and associate accounts), manage identification and authentication and them give way to authorization. And take into account the fact that, iirc, keycloak can do all three. (verify)

Authentication constraints

Should Beep store user data, have access to user data? RGPD? Deleting an account and all associated data, as well as from backups?

Constraints/limits: - User data privacy: - User data security: - User data integrity: - User data availability:

2. OpenID Connect, OAuth2 and Beep

Very briefly present the What? and Why? of these protocols, as an introduction to the next section: How?

3. Comparing available technologies

Brief section
List of tech stacks for OIDC and identification+authentication system, with comparisons, benefits, drawbacks; if possible benchmarks (as pocs).

No benchmarks (as pocs) because no time + keycloak constraint makes dis double-dumdum.

4. Authentication system design and implementation with Keycloak

In this section, we will go over how OIDC can be integrated into Beep's microservices architecture as an authentication system.

Proposition (explanation of the approach, system design, how it fits into chapter I's component and diagrams), and sequence/activity diagram(s).
The PoC too.

4. Keycloak as an OIDC provider

In this section, we will go over Keycloak, the technology chosen to manage authentication in Beep.

[Keycloak](https://www.keycloak.org/) is an [open source](https://github.com/keycloak/keycloak) identity and access management server/service.

Keycloak provides the following features:

- (External ?) authentication manager thingy OIDC provider
- Identity Brokering and Social Login (setup through admin console, no code needed)
- User Federation: built-in support to connect to existing LDAP or Active Directory servers.
- Single-Sign On (SSO) which we need for apofenzioxnw
- OIDC provider
- RBAC (role based) authz AND "fine grained": If role based authorization doesn't cover your needs, Keycloak provides fine-grained authorization services as well. This allows you to manage permissions for all your services from the Keycloak admin console and gives you the power to define exactly the policies you need.
- Add authentication to applications and secure services with minimum effort.
- No need to deal with storing users or authenticating users.
- Keycloak provides user federation, strong authentication, user management, fine-grained authorization, and more.

POC: `docker run --name keycloak -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak start-dev` on proxmox VM Create a custom second (not the default) realm. The default should be used only for managing the keycloak server. Our custom realm will be for authentication and authorization purposes. https://www.youtube.com/watch?v=fvxQ8bW0vO8

The UI and backend are gonna be keycloak "clients" (create in keycloak admin UI) Public vs private client : public client cannot securely store a client's secret. Web UI is public client, private/confidential client is for server-to-server communication.

Tokens in OIDC flow: https://www.csharp.com/article/accesstoken-vs-id-token-vs-refresh-token-what-whywhen/ Access, refresh, ID tokens. Their individual scopes.

> Keycloak: once the user has authenticated via Keycloak, it provides tokens for the user, including ID, Access and Refresh tokens, which are JWTs. These are not shared with the frontend, but are stored on the gateway and associated with the user's session.

As for token contents, only ID so that data is not shared with clients/transmitted/stored there, and as such, always fresh.

OAuth is authz. OIDC is an authn, identity super-layer. I believe.

> With microservice architectures, one of the most used authentication type is OIDC (OpenID Connect), which adds authentication functionality to the OAuth2 protocol used for the authorization part with JWT Token. An open source popular identity provider used for this purpose is Keycloak. Other commercial OIDC alternatives include Okta, AUth0, Microsoft Azure Active Directory (AD), AWS Cognito and so on.

5. Microservices architecture re-design considering Keycloak integration

In this section Let us update our architecture diagram from part one to include Keycloak as an authentication service.

keycloak, api-gateway, services hidden behind in DMZ…

Keycloak replacing the user service? Or being its database? (what about preferences, etc?) → What can it store, what can it do.

API Gateway Considerations:

- The API Gateway can handle request authentication by verifying JWT tokens issued by Keycloak.
- It can also route requests to the appropriate microservices based on the user's identity and permissions.

Possible Architecture:

- Keycloak → Handles authentication (login, token issuance, session management).
- API Gateway → Enforces authentication & authorization before routing requests.
- User Service → Manages user-related data beyond authentication.
- Other Microservices (Chat, Call, Search, etc.) → Consume user identity from JWT.

About what Keycloak can store, it's authentication-related information (username, password, email; roles, groups; authentication settings such as MFA, login attemps, sessions; OIDC tokens, SSO and stuff) So the User service would still need a database to store user preferences, settings, in-apps stats or whatever, … → Decouple authentication logic from application logic. And more stuff.

Why an api gateway/why network zones/network zones separation

Why we need an api-gateway: (microservices architecture complex, poses pbs, api gateway fixes these) https://www.solo.io/topics/api-gateway/api-gateway-microservices

Manages request endpoints so that endpoint updates to services are internal only and not exposed to users? And stuff. See all that. Also is the only service that talks to keycloak, as an OAuth2 Client. User is a public client, api-gateway is a confidential client, maybe? The tokens it gets are used when communicating with a backend service. > API Gateway acts as an OAuth2 Client: in this case, any unauthenticated incoming request will initiate a flow to obtain a valid Token. Once the token is acquired by the gateway, it is then used when sending requests to a backend service. The API Gateway interacts with the authorization server to obtain access and refresh tokens and stores them securely.

Maybe the other services do communicate to keycloak, but differently than the api-gateway? So that the api gateway doesn't have a huge load to handle?

> At microservice level, authentication is not intended as the authentication used for identify users of an application. Authentication for a microservice means receive request authenticated and in this example means that all calls received from the gateway must be authenticated. A common example of authentication on microservices is JWT tokens. Endpoints exposed by microservices will only be accessible if a valid JWT token is provided.

disadvantage: keycloak/authentication server is our SPOF. If not the api-gateway.

2-3 zones? One (or two?) have to be publicly accessible: api-gateway (and keycloak?) or just keycloak? nah. Api-gateway is the endpoints, right?

Maybe use keycloak only for user management identity and authentication, and for authz use something else, and mTLS for inter-services communication security/authentication? To avoid having too much load on keycloak/SPOF? Or does that just lead to more SPOFs? Would that be better/more scalable anyways? And performance? (What are Beep's performance/speed requirements? In/from chapter 0.)

RBAC on user, then OPA/Permify/… for ABAC/PBAC/ReBAC/…? Or all authz via keycloak? Lelz. ? ?? > access tokens are also used by the underlying services for the role-based user authorization part. Roles are defined on Keycloak, assigned to users and included in the access tokens.

How Keyclaok handles identification+authentication+authorization

Explanations, diagrams, etc.

Connecting Keycloak with surrounding authentication mechanisms

PoC with current Beep app. Google + Polytech LDAP auth into an SSO or smth.

WIP and sources

For the poc, use react-oidc-context

Goatesque : https://medium.com/@a.zagarella/microservices-architecture-a-real-business-world-scenario-c77c31a957fb

Montrer les tokens reçus à la connection/création de compte, SSO, etc.

https://adil.medium.com/multi-container-patterns-in-kubernetes-adapter-ambassador-sidecar-40bddbe7c468 K8s containers-in-pod patterns identified : sidecar, adapter, ambassador. (more?)

list of technologies & concepts that can be used:

- OAuth2
- OpenID Connect
- SAML
- Ory
- Okta L.O.L.
- Keycloak ofc
- SSO
- See how GCP (and others) do IAM.
- more?

Lier les méthodes d'authentification aux comptes

https://developers.google.com/identity/protocols/oauth2

Oauth2: https://www.youtube.com/watch?v=ZV5yTm4pT8g OIDC (surcouche ?): https://www.youtube.com/watch?v=t18YB3xDfXI

à GCP, pour la comm entre CHAQUE service, y'a un système d'AUTHENT puis d'authorization !!! Pas juste authorization (0 trust approach). Est-ce que ça suffit d'avoir du chiffrement entre les services, ou faut-il un système d'auth complet ?

Faut des trucs en plus pour les microservices : Circuit breaking pattern. Important pour les microservices pour pas que ça call en continue avec les retry réseau. Retry exponentiel (1s, 10s, 1mn… et que ça bloque tout le service) → on arrête d'appeler le service (on ouvre le circuit) et pas mécanisme de fallback (réponse préfaite en cas de pb, genre "ah dsl jpp afficher ça en fait" alors que ça chargeait) → En gros gestion d'erreur réseau en fait. Pour éviter surcharge réseau + jamais de réponse. Aussi circuit breaking, fault tolerance, latency… Problématiques de microservices entre eux. → Quota (peut faire 1M d'appels à service X sur un

mois, etc) + Rate limiting (même chose mais sur une période très courte, genre secondes ou 1mn).

Et logging, metrics (métriques techniques, CPU, etc. Ou plus fonctionnelles rédigées par le développeur genre nb de requêtes, etc.), distributed tracings (suivre l'appel de son entrée et toutes ses transitations de svc en svc. Permet d'identifier dans quel svc y'a des pbs quand y'a un pb sur la requete, genre latence ou erreurs) et topology.

Security, observability, network resilience (genre trucs de circuit breaking etc), policies. En sidecar containers (envoy??) en PLUS du service logique !! dans le pod. Donc un container app et un container proxy qui a les 4 trucs secu, obs, netw resi et poli. Qui intercepte en premier tous les calls puis retransmet.

Inscrire les services et leurs endpoints dans une bibliothèque de services, un "service discovery". Pour gérer leur scalabilité et des trucs.

Dans un service mesh : Il y a un control plane : api/interface pour donner des instructions pour configurer le control plane, ses proxy qui vont appliquer les configs (d'auth, de traffic management, de sécu réseau type ntls?mtls? c'était mTLS etc. Certaines traitées en inbound ou outbound du proxy), etc; Pour les microservices, le service mesh permet de gérer facilement le traffic entrant, sortant et intérieur aux services (traffic splitting, canary, blue-green, mirroring...), sécuriser l'accès et comms (mTLS etc.), et visibilité complète sur etc.

mTLS avec Istio ou HCP Consul

Tout ça est implémenté dans Istio !!!!

https://istio.io/latest/docs/tasks/ Exemples de comment mettre en oeuvre ces fonctionnalités !!!!!!!!!!

https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/
https://www.youtube.com/watch?v=uWmZZyaHFEY

OAUTH OIDC SSO SAML

https://samarthasthan.com/posts/building-a-scalable-e-commerce-empire-a-micro-services-system-design-approach/ https://www.geeksforgeeks.org/how-discord-scaled-to-15-million-users-on-one-server/

Random important stuff: event driven architecture and aggregates saga pattern service that redirects microservices service and repository pattern http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod https://medium.com/microsoftazure/certificate-pinning-for-mtls-authentication-at-the-istio-ingress-gateway-978ed31699ab https://www.youtube.com/watch?v=vJweuU6Qrgo https://www.geeksforgeeks.org/how-discord-scaled-to-15-million-users-on-one-server/ activity diagram for authentication keycloak activity diagram activity vs sequence diagram

https://discord.com/developers/docs/topics/oauth2 Find Discord's OAuth flow of authentication?

https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc OAuth 2.0 is

designed only for authorization, for granting access to data and features from one application to another. OpenID Connect (OIDC) is a thin layer that sits on top of OAuth 2.0 that adds login and profile information about the person who is logged in. Establishing a login session is often referred to as authentication, and information about the person logged in (i.e. the Resource Owner) is called identity. When an Authorization Server supports OIDC, it is sometimes called an identity provider, since it provides information about the Resource Owner back to the Client. https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc#learn-more-about-oauth-and-oidc

Oauth, OIDC, identity ***and access** managers: provide authentication (oauth), user identity (surcouche oidc de oauth), et parfois aussi gère l'authorization/perms! Genre okta : "and access" manager.

JWT: bien pour avoir des sessions (date d'expiration) Ne pas mettre de data sensible PAS DE SYSTÈME DE RÉVOCATION!! On les révoque dans le système manuellement en les supprimant… C pas dans la spec. (biscuit y a pensé, voir plus bas chap IV) Le chiffrement du jwt dépend du SDK qui l'a créé.. Peut être passoir, si pas config, etc. Et si tu connais le SDK, tu peux crack. → Alternative : PASETO (Platform Agnostic Secure Token) : version specific protocols, up-to-date, secure cryptographic algorithms Purpose explicit: public and private/local(server-side) tokens. Plus possible de changer l'algo de chiffrement sur le haeder déchiffré du token, et signature obligatoire. Meilleures règles d'inmplémentation/fiabilité en gros. + système de révocation. + le fait que t'ai la partie locale sur le service = on sait/est sûr de quel service l'a créé, nativement un peu en gros. https://permify.co/post/jwt-paseto/

Oauth : juste la clé pour la serrure OIDC : toute l'identité raccrochée qui va avec.

III. Enabling communication between services

The following sections concern my proposal of design and implementations for the communication between the microservices of Beep.

Pour les diagrammes, reprendre la figure 3 et mettre le gRPC / interfaces RPCs(?) / repo protobuff(?). Puis refaire une 5e figure qui, par dessus cette figure 4, rajoute aussi l'api gateway/+/LB, le frontend devant, et derrière, les requêtes REST aux endpoints APIs offerts par chacun des services. C'est en json? P'tet.

1. Inter-services communication in a microservices architecture

Briefly present the What? and Why? of inter-services communication. This should be done as a natural introduction to the next chapter (Identifying needs and constraints).

2. Identifying needs and constraints

Talk about the needs in latency etc (see chapter I. Maybe move these to chapter 0? to be more globally referenced) and that we need a proper solution for that (so no http1.1)
Also talk about the fact that we'd have teams working separately and independtly ⇒ Proper API definitions. So RPCs more than an OpenAPI spec (common/shared API definitions over viewable API specifications)
Talk about CQRS/event-sourcing etc (their use, why they are used typically) and say that we won't use them (and why it's not considered? Find the reason?)
Talk/warn about the very important constraint that is that communication between microservices is something that needs to be designed well, and future proofed very well, since it's very very hard to change later (and why).

Accounting for considerations in inter-services communication

Talk about the fact that since we're not doing event-driven architecture, we need API gateway interfaces (services, if not LBs) in front of each service family, to load balance/state-aware round-robin the requests. And that if requests are lost, they'd have to be resent, they are not queued - and services don't queue to event source either.

Maybe briefly talk abt tracing for the next chapters too?

Needs: link Google auth, Polytech auth, Beep auth (and associate accounts), manage identification and authentication and them give way to authorization. And take into account the fact that, iirc, keycloak can do all three. (verify)

Dedicated central repository/location for the .proto files. And service discovery (or even a service that manages only that?)

3. Comparing available technologies

List and compare technologies, protocols, frameworks for communication.

Mainly:

RPC and REST

HTTP1.1 (json) and HTTP2/QUIC/3 (binary+compression+more)

Deliberate the best one under those constraints. From all I've read until now, I'm choosing gRPC, mainly over GraphQL.
From what I'm seeing, GraphQL and gRPC are opposites when compared on one of my favorites software development philosophies to follow: smart data structures for simple code over dumb data structures and complex code.
GraphQL is the dumb data structure for complex code, and gRPC is the smart data structure for simple code. GraphQL is a query language that allows the client to request only the fields it needs, and nothing more. This means that the client has to filter the data it receives, and that the server has to send all the data it has. Blahblahblah.
Plus extremely heavy and intertwined configuration… What the fuck, graphql??

What do you mean I gotta rewrite my schema in yaml for you?? AND THEN IN JSON?? IN ADDITION TO YOUR SDL??? Deduplicate even that??

graphql needs a LOT of work to be proper. And then to grow it. It may offer some bandwidth advantages MAYBE. If done right that is. But disadvantages in other ways such as confidentiality? Idk. I think performance issues. Idk.

I want to do benchmarks… At least give numbers and cite them from existing, relevant benchmarks… (You probably won't have time to benchmark yourself. Maybe if the poc is easily interchangeable, but that's very unlikely)

4. Communication framework implementation with gRPC

Improve earlier diagrams with gRPC interfaces, RPCs and shared/common API definitions, inter-services interactions; and present the POC.

In this section, we will go over how OIDC can be integrated into Beep's microservices architecture as an authentication system.

Proposition (explanation of the approach, system design, how it fits into chapter I's component and diagrams), and sequence/activity diagram(s).
The PoC too.

Link to poc:
https://github.com/theotchlx/inter-services-communication
Should it be deployed too?? No I don't think that's smart. But readme should be clear and consise steps. + excalidraw small archi diagram in readme.

WIP and sources

List of technologies that can be used:

Per directives: → No message queue, CQRS or event-sourcing (so no Kafka/etc.). Sadge.

- REST: http 1.1, slow, heavy
- gRPC: RPC, common defined API interfaces, interface is sent with message (verify), http2, faster than http. Oh and interface attributes are numbered!! Very important difference. Why is it even like that? Verify. Just against name uniqueness by position? What advantages does this really have to offer?
- GraphQL: lots of formats possible (including binary formats), but DX is not very scalable I believe + security&cie concern (whole data schema is sent, it's up to the client service to filter what it wants to see)
- Apache Thrift:
- Avro: interface is sent with message, binary format, can be decoded to json natively/easily(verify), natively easily integrated with Kafka

- more?

Benchmarks, or at least real numbers, then comparisons,
Advantages|benefis/disadvantages|drawbacks of each

https://devopedia.org/inter-service-communication-for-microservices https://github.com/Netflix/Hystrix fault tolerant capable framework n more

Communication. Message-driven architecture.
Comm synchrone : http de l'un à l'autre, si l'autre tombe, la comm passe pas.
Comm asynchrone : Envoie de mail. On l'envoie. il sera stocké et reçu à un moment dès que possible quand le service de réception/envoi sera good.

API composition pattern for microservices : un service a la connaissance des autres services, connaît les contrats API qui permettent de les faire parler entre eux : fait de la composition.
Tu as Cours et Etudiant : le machin map les deux. Et en plus peut enrichir la donnée ! Avec dans quel service elle est passée, etc. I thnik.

GraphQL!!! Format binaire underlying est interchangeable !! Par contre faut build soit-même les APIs? à voir. Pas grave en vrai. Mais faut les modifier soi-même ??

Si je comprends p'tet, avec graphql tu renvoies masse de données et tu filtres ce groc bloc côté client. Donc faut modif ton code client généré. Et faire bien attention à l'aspect sécurité… ? À tester si c'est bien ça la différence. Noter les différences. Pour plus tard les comparer. Ou graphql pour server-client final et grpc pour service - service ? When to use gRPC or graphql? Do they even compare? If so, how? GraphQL for microservices? "[GraphQL] permet notamment aux consommateurs de l'API de demander seulement les champs nécessaires à l'inverse d'une API REST qui expose un schéma prédéfini." https://affluences.com/blog/optimiser-architecture-micro-services/ Ah oui en effet niveau sécurité ça a l'air dur à gérer, si ton service toi exposer tout à tous les autres services et que c'est à eux de choisir… Ou à l'inverse… ?? J'ai juste l'impression que ça ne suit PAS DU TOUT le principe de "smart data struct for simple code vs dumb data structure for complex code". Donc pour l'instant c'est un non. Le code serait dur à maintenir/scaler, et dur à sécuriser, il me semble. Et c'est aussi deux retours que j'ai lu. Un peu biaisé, mais aussi en partie logique. !! ⇒ Answer to "Why [gRPC and] not GraphQL?"

In rust, with grpc? (Contribute to Tonic's doc cuz it's shit, on build.rs setup mostly?? idk. Maybe it's a skill issue) grpc cuz kube, google etc? Real motives. Why other are not better choices. USE REAL NUMBERS like estimations to say why they're not better!!! See kafka cours .md obsidian

Quelle architecutre ? Saga pattern ? Kube avec apiserver et etcd centralisé ? Juste etcd centralisé ? Ou tout state et api distribué ? Ou juste API centralisé et state distribué ?? Comment dissocier le storage ? Rajouter des questions sur le TAD ? Lel

Poc : 2 services Rust. Chacun une BDD : postgres et l'autre mysql ou autre. Pour montrer que peut séparer ainsi les systèmes (mongodb, sqlite auraient pû être choisis aussi !) scylladb, etc. Serveur / user ? Ou un truc du genre. Ou channel / message. et un docker compose. deux dockerfile.

Et même poc mais avec autre chose que grpc.

3 dossiers, 1 .git. 1 dossier common / interfaces / whatever avec les .protos ou autre, 1 pr le premier service (cargo new) et un autre pr le second service (cargo new). Ou le faire en Go. Dépend de ce que veut poccer.

Pourquoi Rust? Car <avantages du Rust> + désavantages du rust : plus gros binaires. Mais pas important dans le use case de beep, car (etc C +petit mais on fait pas de l'IOT et etc etc). Voir même bénéfique car bien plus petit que environnement typescript anyways car (etc. nodemodules frameworks node deno pnpm npm etc) Y-a-t-l un site qui recense les avantages et désavantages comparés du Rust? ptet! Ou un blog post idk. Le citer, dater sa lecture, et sortir la citation datée !

Faire un joli schéma du poc.

Schémas : UML ? Séquence, composants, useCase (avec le bonhomme) + des plus classiques, architecturaux à la mano non-standards compréhensibles sans app des règles ? Ou c'est kaka ? :X Je crois que c kk..

gRPC/Protobuf / Avro / Apache thrift (RPC), ultra modulaire. Par contre prise en main pas facile… Configs, etc. Mais fine-tunable. Avro plus utilisé avec Kafka. Décodable du binaire au json! Tu as deux fichiers envoyés : metadonnées (défini le type, le champ que ça remplit etc). Et l'autre c'est de la donnée pure. gRPC l'ordre des champs compte. gRPC envoie aussi le schéma supposément vu que gRPCurl peut curl comme ça. Il me semble. gRPC plein de styles d'intéraction, stream bidirectionnel, etc.

Service registries pour qu'ils sachent qu'il y a eu une modif d'api ou quoi.

Sozu vs nginx, vs sozu? benchmarking!! (Dockerfiles) vs rpxy

=== Communication framework proposal

In this section, I will propose a communication framework for Beep's microservices.

In this proof of concept, I implement a communication framework between mock services. I used gRPC as the communication framework.

gRPC is… (what)

I chose gRPC because… Some numbers… (why) Compared to….

I implemented using Tonic… (how)

https://github.com/hyperium/tonic/blob/master/examples/helloworld-tutorial.md
https://github.com/theotchlx/inter-services-communication

on vm in proxmox.

SAY DTN AND BP FOR SMTH OTHER THAN HTTP!!! :3333333333

microservice communication best practices microservice communication protocols microservice communication patterns microservice communication pitfalls

TALK ABT asynchronous/synchronous comms. between microservices https://youtu.be/uprdxlQ1U5g?t=589 How to do async. comms. without kafka/RabbitMQ or the other things forbidden?

https://www.youtube.com/watch?v=16fgzklcF7Y omg nana https://www.youtube.com/watch?v=voAyroDb6xk NYAHHAHAHAHAH

IV. Authorization system design and implementation

WIP and sources Utiliser les concepts de mon talk en cours de Go sur la résistance aux failles !!!!!!!!

Permify, Zanzibar, AuthZed (zanzibar, permify en plus mature). https://authzed.com/, https://zanzibar.academy/, https://research.google/pubs/zanzibar-googles-consistent-global-authorization-system/, https://permify.co/post/google-zanzibar-in-a-nutshell/.

Permify/OPA/Kyverno. ATTENTION policies infra/network VS policies applicatives !!! ATTENTION!!! Gérer les policies applicatives (mon user dans mon serveur) fait peut être partie de la logique service!!! Voir D'ABORD des vidéos/sites ou autre qui expliquent comment on gère les permissions d'une application/applicatives dans une architecture microservices.

https://zanzibar.academy/ Google zanzibar.

ReBAC (relation-based access control)

Permify has a playground: https://play.permify.co/

Keto/permify/opa

list of technologies that can be used:

- OPA
- Oso
- Keycloak
- Permify
- Ory (keto?)
- See how GCP (and others) do IAM.
- more?

Maybe take inspiration from K8s' authorization system, which first goes through an RBAC check, then checks requests validity through admission controllers.

permissions ultra atomiques regroupables héritables attachables à n'importe quelle ressource

équipe contient gens équipe à des droits les gens ont des droits aussi les gens héritent les droits de l'équipe, mais sous forme d'un groupe de permissions qui porte le nom de cette équipe (comme ça paf on retire le gars de l'équipe → ça màj les droits automatiquement - ou on ajoute un gars à l'équipe et paf il a ses droits màj) les gens peuvent override les droits (comment gérer ça ?) → un "yes" par défaut override ? Un "no" par défaut override ? Ou si la perm / ensemble de perms est placé avant, elles override ? (= rôles discord) Ces groupes/ensembles de perms (=rôles beep) seraient donc attachés à une ressource, et héritables, et overridables.

what does google cloud handle authorization, permissions and policies? Not only via IAM, but in itself, in organizations/projects/... IAM : *IDENTITY*!!!! and *access/authorization*!!! management!!!

principle of least privilege

Gérer les permissions par groupe de permissions

Les permissions sont le truc le plus atomique, qu'on verra toujours partout ! Il sera handle partout : service(s) pour le gérer ! (et pas ds chaque service sinon kk hihih)

Si jamais j'applique une modif des droits/perms alors que je peux pas, ou à l'inverse une modif ne s'applique pas (fait à la main en call api manuel, ou un service qui a foiré) du coup y'a un état transitoire à régler. Donc avoir un controller qui monitor ces états transitoires - ou plutôt monitor un etcd pour savoir si un truc est fait ou pas et s'il y a un truc à faire ? centralisé (comme dans kube), pas distribué. Mal ou bien ? Quelle architecture ? Les comparer sur le TAD!!

> Note: The deny-all-ingress and allow-all-egress rules are also displayed, but you cannot check or uncheck them as they are implied. These two rules have a lower Priority (higher integers indicate lower priorities) so that the allow ICMP, custom, RDP and SSH rules are considered first. "PRIORITY"

Service mesh

https://en.wikipedia.org/wiki/Attribute-based_access_control#API_and_microservices_security

Authorization / permissions services must be external from all other services, and they all need to use it. So... Why not integrate it directly inside of K8s ? Kubernetes service meshes have proxies. Service mesh proxies that stand in front of services and handle the filtering, other stuff, etc. A lot. And OPA (Open Policy Agent) stands with the proxy, and handles the authorization policies. https://www.openpolicyagent.org/docs/latest/ https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/ https://sysdig.com/blog/kubernetes-admission-controllers/

Is this how it can be done? Can OPA be configured to handle Beep's authorization service, or do we have to write it ourselves, or is there a protocol or standard to implement, or a config to set and then something (OPA maybe) handles the authorization for us?

FAIRE DU BENCHMARK GRPC (http/2 ?) VS HTTP REST ETC!!!!

https://istio.io/

Istio + OPA

Istio vs linkerd (both cncf btw) kiali: console for istio

Keycloak (and why not authentik)

OpenTelemetry https://opentelemetry.io/docs/what-is-opentelemetry/ Jaeger, OTLP (otel line protocol), Prometheus → OpenTelemetry collector OTEL : très bien pour les traces. Attention Beta pour Rust. Metrics pas mal, logs bof. "Profiles" : juste annoncé. "Zero code instrumentations" pour Go, Python, JS, Java ! Signifie + facile pr récup les logs (quasi pas de modifs à faire, se branche au runtime etc. Je peux aussi récupérer des données spécifiques à mon application)

https://prometheus.io/docs/introduction/overview/

Elasticsearch pr données à bcp de cardinalité. Kibana Sinon Loki (entendu à conf cncf grafana sur OTEL)

https://opentelemetry.io/docs/specs/otlp/

https://cloud.google.com/iam/docs/roles-overview

https://medium.com/@sadoksmine8/understanding-identity-and-access-management-iam-in-gcp-a-detailed-exploration-57030ec37609

permify vs keto

Permify : authorization for microservices. + patterns !! https://play.permify.com Ory / Krong microservices OPA Google Zanzibar https://www.youtube.com/watch?v=5GG-VUvruzE

Chaos mesh

Oso https://www.osohq.com/ https://github.com/osohq/oso Mieux que OPA supposément. Niveau config et architecture du truc. Tester, poccer, prouver, comparer.

oso vs opa

https://github.com/Permify Permify !!

https://getsops.io/ SOPS: encrypts data client-side(?verify), + sealedsecrets encrypts server-side. https://getsops.io/docs/ It's CNCF-sandboxed. But I think it's a bit too much for now, and also this particular software doesn't seem extremely well defined/developed yet? See others. But it's not really a current concern in Beep. I think. I dunno! https://github.com/getsops/sops

Service mesh: Istio, Linkerd Mutual TLS (mTLS) encryption for secure inter-service communication

fully externalized authorization: pros: - no workload on the targeted service - no authori to handle in the service cons: - Scattered logic : too much logic on the authorization software... - Heavy workload on authori software = even more of a SPOF!! Duplique de la logique : check

d'auth en amont mais svc doit redemander si a le droit au sys d'auth etc etc.... → Alors qu'on aurait pu juste check l'authorization au moment de la logique ds le service. Éviter aussi que le truc d'auth doive faire de la recherche en bdd.. Pas fait pour souvent en plus.. (plutôt fait pour renvoyer oui/non mdr)

On a le mapping des ressources et les auth qui en découle. Pas avoir ça dans les bdd de chaque service (pr préserver leur indépendeance). Donc le svc d'auth va requeter SA bdd. et tu as la logique d'auth dans TON service. Donc en gros ce qu'il faut (car c'est génial un syst d'authorization externe centralisé) c'est d'avoir la logique d'authorization dans chaque service, mais les règles/mappings entre ressources et les auths qui en découlent DANS le service d'authorization.

Biscuit : système d'authorization décentralisé Datalog (basé sur prolog) Token/JWT attenuation (When making the JWT, remake it from the request to minimally scope the authorizations). Donc au lieu de taper un système, tout est déjà scopé dans le JWT. la logique d'authorization est dans ton service. https://www.biscuitsec.org/ https://github.com/eclipse-biscuit/biscuit Les SDKs/CLI/... créent le token de manière à ce qu'il https://www.youtube.com/watch?v=v7JkOxSG4gI https://www.clever-cloud.com/blog/engineering/2021/04/12/introduction-to-biscuit/

V. Tracing logs and queries

Easy peasy. Sidecars + OTEL + LGTM + kumas. paf.

handling traces in the new distributed architecture traces, logs, queries, metrics, observability, monitoring

snowflake UUIDs (UUIDv7 = snowflakes?) to sort chronologically and etc

how does tracing work microservices https://www.youtube.com/watch?v=XYvQHjWJJTE

Tracer les DENYs... Mais aussi les accès.

VI. Setting up a production ready system

FAULT TOLERANCE HEEHETEEHEEEEHEE

cia triad principle of privilege (including in service mesh)

Migrate infrastructure to a (or 3???) proxmox cluster. With a high-availability Kube on top. Separate etcd or not? Postgres as etcd or not? Proxmox vs apache cloudstack vs openstack.

Apache Mesos: Program against your datacenter like it's a single pool of resources. Kubernetes pour l'infra ou qq chose comme ça.

"Mesos propose deux modèles de fédération. Une première approche place toute l'infrastructure sous une couche de contrôle et crée une abstraction des ressources du datacenter, d'un cloud public, d'un déploiement de VM par exemple. Cette couche de contrôle forme une abstraction

uniforme pour l'hébergement. Avec le deuxième modèle, la technologie rassemble des déploiements Mesos distincts de manière à ce qu'aucun ne soit relié à un autre, mais coopèrent tous de manière totalement distribuée et tolérante aux pannes."

https://www.baeldung.com/apache-mesos https://mesos.apache.org/documentation/latest/ https://agenda.infn.it/event/29701/sessions/21750/attachments/88134/117909/Apache%20Mesos.pdf

CNI plugins: Flannel, cilium, calico, … There are more good ones! To read to understand stuff: https://mvallim.github.io/kubernetes-under-the-hood/documentation/kube-flannel.html https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/ https://kubernetes.io/docs/concepts/cluster-administration/networking/ Flannel has basic features when compared to cilium/calico. Supposedly. I haven't tested it yet.

MinIO vs seaweedfs vs deuxfleurs's garage

Pour le load balancer à self hoster : https://geek-cookbook.funkypenguin.co.nz/kubernetes/loadbalancer/ https://medium.com/@ferdinandklr/creating-a-production-ready-self-hosted-kubernetes-cluster-from-scratch-on-a-vps-ipv6-compatible-660aa5018feb MetalLB(?)

"Target diagram" == "Diagramme de l'architecture cible" !!

How? to put this in place

Like google: Dev measured in agility, ops measured in stability of product? No, makes quarrels (cuz change breaks stability) → SRE approach = budget d'availability (25.9s à 99.999% par exemple). Availability : client's metrics ! Product/service/site must be …(use cases/scenarios/critical user journey) (available, fast, provide good stuff, etc.). → dev push push push… Until unavailability time is épuisé → mains levées du clavier et ops et dev se concentrent sur availability.

Service mesh: Istio, Linkerd Mutual TLS (mTLS) encryption for secure inter-service communication

Each team works on its service's logic. And a networks team + … team works on the sidecar proxies (fault tolerance/circuit breakers: link my poc; auth; etc…)

cockroachdb

Mettre des UUIDs v4 ou v7 en bdd plutôt que incrémentaux, pour éviter que les attaquants puissent juste incrémenter l'ID pour trouver les autres users et voler toutes les infos des users ds la bdd. Aussi UUID v4 a plus de chances de clasher quand on en génère beaucoup d'un coup. UUIDv7 ajoute un timestamp. Donc en plus d'être plus unique, ça peut être trié, optimiser l'indexation, etc. https://www.uuidgenerator.net/ Les UUIDv4 étaient trop rapides à faire : plus faciles à bruteforce. Les UUIDv7 sont un peu plus longs à générer… Mais exprès. Pour que ça soit plus dur à brute force. Genre ont rajouté des calculs de courbes elliptiques exprès pour que ça soit plus complexe en calcul et hasard, et que ça prenne plus de temps.

TLS : va que dans un sens client-serveur. Le client sait qui est le serveur mais le serveur sait pas.

QUIC (Quick UDP Internet Connections) : TLS + UDP. Gagne du temps/vitesse sur les handshakes TCP (et/ou TLS?). Et gestion de stream. Streaming dans la spec de TCP est rarement/efficiement bien implémenté. Rapidifie de bcp le first content load dûes aux handshakes TLS?

mTLS : davantage confiance au client. Mutual authentication: improve trust.

TLS : le serveur est vérifé et le client peut lui faire confiance. mTLS : le serveur peut aussi faire confiance au client! Peut plus sniffer. Il faut aussi des règles de sécurité. Pour complètement éviter le MITM.

RGPDR : quand user demande à ce que ses data soit del, cela inclue aussi les backups de bd ! Taille max de data chiffrable par clé = dépend de la taille de la clé. C'est aussi une des raisons pour laquelle on utilise des clés symmétriques.

Toutes les données utilisateurs sont chiffrées : email, nom, mdp.

Tradeoff qualité chiffrement et temps de création du secret. Par contre plus de temps = beaucoup plus dur de déchiffrer pour l'attaquant.

Bcrypt: itère sur un nb de tour. Utilise du sel (aléatoire) GPUs pour paralléliser calculs : se prévenir d'attaques de crackage psswd par GPU : Argon2. Restreint l'usage mémoire. Optimisé pour la résistance contre les GPUs. ⇒ cryptomonnaies, applis avec moins de risques de side channel attacks. Plus lent que bcrypt tho. Donc pour un usecase anti-GPU.

Les clés qui chiffrent les data en bdd… Sont aussi en bdd. Si je supprime la clé on peut faire un système qui clean les data user dans toutes les bdd. Avec les saga pattern. Mais les clés sont dans une autre bd du coup? Ou dans la bd de chaque service, clés diffs chiffrent données diffs?

Ou les clés sont côté client. D'autres PBs (perdre sa clé, voler, …) Mais plus de confidentialité.

Ou split la clé privée en morceaux, et il les faut tous !! (Horcrux) https://github.com/jesseduffield/horcrux Vault fait ça ?

VII. Infrastructure security

design, implementation, automation and handling

cia triad principle of privilege (including in service mesh)

VIII. Search engine design and implementation

Separate service

Search = indexing…

Figma/Drawio(?) UI mockup/frame

Sequence diagram of the indexing and search mechanism.

Can search whats? All/most elements of Beep?

- Servers (from server discovery pages, side page, etc.)

- Users (from channels, from private messages, etc.)
- Messages (from channels, from private messages, etc.)
- Files (from channels, from private messages, etc.)
- more? What else. Channels? meh. It's a bit stupid. Maybe channels but across servers? Like in message transfer in Discord, you can choose a channel across servers.

for files:

Files have multiple lifecycle stages (short term, for previews etc. Long term, for storage etc. Iceberg & similar). Multiple object storages. Multiple file storages. Many different types and sizes. How to manage and distribute all that, and also search and index it?

IX. Managing platform security issues

X. Integrating UI applications

microfrontends?? https://micro-frontends.org/