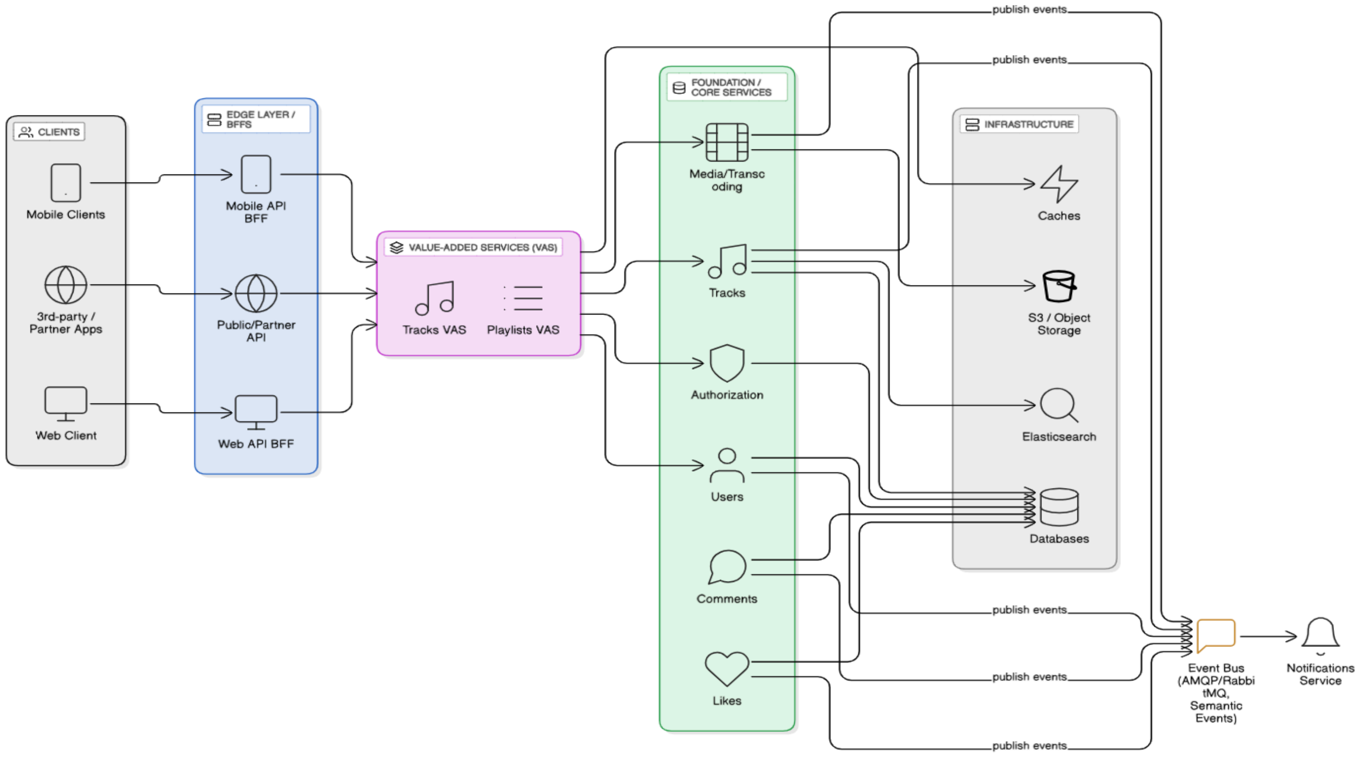
Lab 2 – Microservices at SoundCloud

Logan Robert Dutton-Anderson



Architecture Overview

SoundCloud originally relied on a single **Ruby on Rails monolith** (“Mothership”) that included controllers, views, and business logic, backed by MySQL and memcached. While simple, this design became difficult to scale and slowed feature delivery.

To address this, SoundCloud adopted a layered **microservices architecture**:

- **Clients (Web, Mobile, 3rd-Party):** All user requests first pass through the Edge Layer.
- **Edge Layer (BFFs):** Dedicated APIs per client type — *Mobile API BFF*, *Web API BFF*, and *Public/Partner API*. These handle authentication, rate limiting, and caching.
- **Value-Added Services (VAS):** Middle layer that aggregates data and applies authorization rules. Some examples include the *Tracks VAS* and *Playlists VAS*. These simplify BFF logic by consolidating calls to multiple backend services.
- **Foundation Services:** Core building-block microservices such as *Users*, *Tracks*, *Comments*, *Likes*, *Authorization*, and *Media/Transcoding*. Each owns its data and

exposes a clear API.

- **Infrastructure & Event Bus:** Services rely on databases, caches, object storage, and Elasticsearch. An event bus (RabbitMQ/AMQP) broadcasts *semantic events* (e.g., “new track uploaded”), enabling asynchronous workflows like notifications.

Example Microservices

1. **Tracks VAS** – aggregates track metadata, transcoding, and access control.
2. **Playlists VAS** – centralizes playlist logic and authorization.
3. **Users Service** – manages user profiles and accounts.

(Some others: Comments, Likes, Authorization, Media/Transcoding.)

Pros and Cons

Pros

- Independent deployments enable faster iteration.
- Services can be scaled individually.
- Fault isolation prevents one failure from crashing the entire platform.
- Client-specific APIs (BFFs) improve performance for mobile vs web.

Cons

- Operational complexity (monitoring, testing, versioning many services).
- Added latency from inter-service calls; VAS layer helps reduce this issue.
- Synchronizing shared logic (like authorization) across services is challenging.
- Migration from the monolith required years of incremental work.

References

[1] Phil Calçado, *Building Products at SoundCloud — Part I: Dealing with the Monolith*, SoundCloud Backstage Blog (2014).
[2] Phil Calçado, *Building Products at SoundCloud — Part II: Breaking the Monolith*, SoundCloud Backstage Blog (2014).
[3] Jorge Creixell, *Service Architecture at SoundCloud — Part 1: Backends for Frontends*, SoundCloud Backstage Blog (2021).
[4] Bejal Lewis & Marc Tuduri, *Service Architecture at SoundCloud — Part 2: Value-Added Services*, SoundCloud Backstage Blog (2021).
[5] Marc Tuduri et al., *Service Architecture at SoundCloud — Part 3: Domain Gateways*, SoundCloud Backstage Blog (2021).