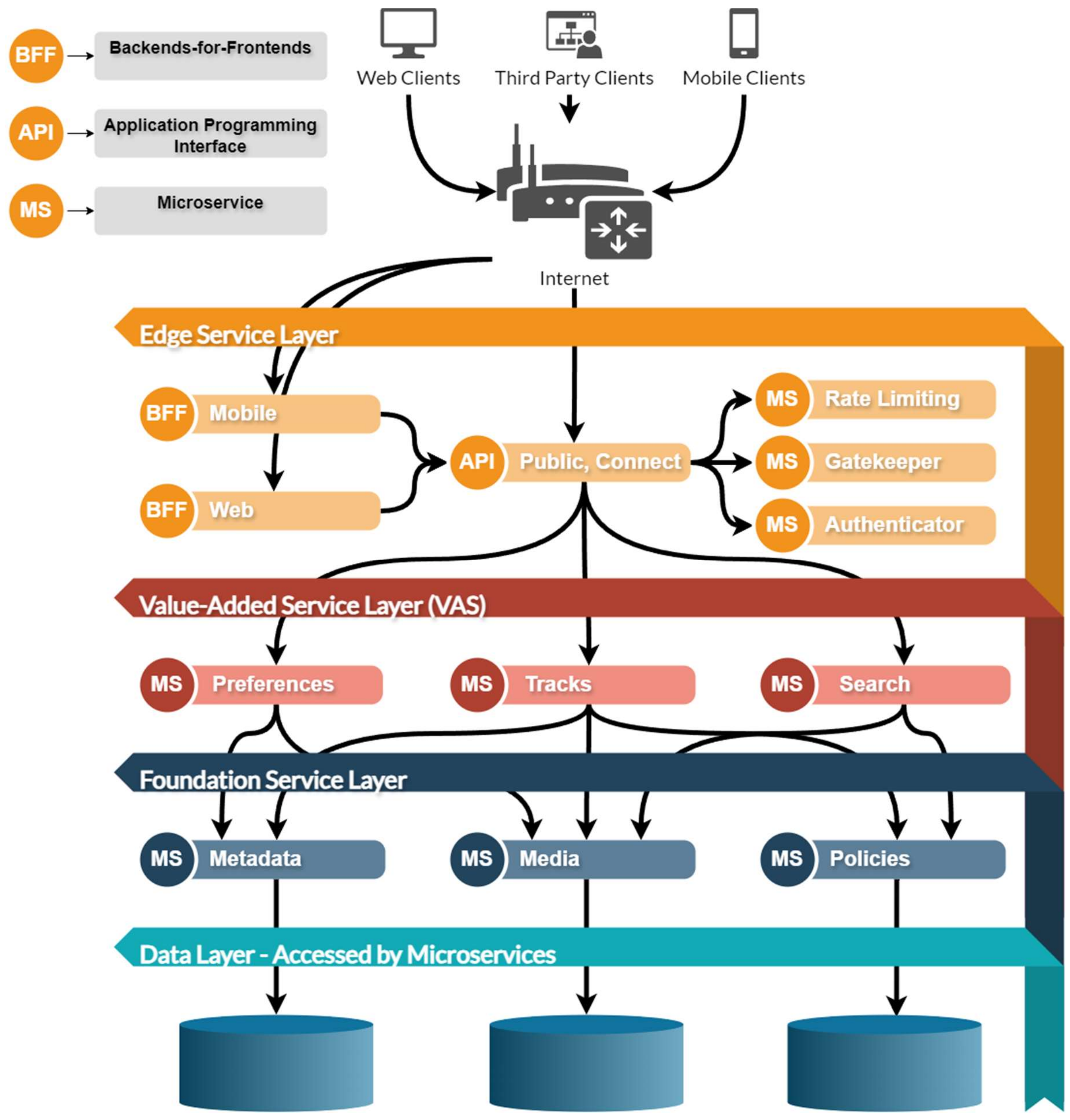
Catherine Queenan

Lab 2, Microservices at Soundcloud

SoundCloud originated as a monolithic application, but its user popularity meant it needed to scale into a social network that also handles media distribution. Understanding the benefits of microservices for their application, they became one of the first to make the transfer [2].

As SoundCloud made the transition, they developed the microservice systems to function in conjunction with the monolith, referred to as the Mothership [7]. That way, they could keep the existing system running while testing the new services [4].

Between the data layer and the clients, SoundCloud consists of three microservice layers; Edge Service Layer, Value-added Service Layer, and Foundation Service Layer [6], [7]. Within the Edge Service Layer, SoundCloud utilizes Backends-for-Frontends (BFF) to handle different client types (for example, web or mobile) [5] and services such as authentication and gatekeeping [3]. Below is the Value-Added Services (VAS) that act to prevent BFFs from directly calling individual microservices [5]. They group logic to link the two layers [5]. The final layer is Foundational Services, where the lowest level microservices exist [5].



Examples of SoundCloud Microservices: [7]

1. Media Service (Foundation)
2. Metadata Service (Foundation)
3. Policies Service (Foundation)
4. Search Service (VAS)
5. Preferences Service (VAS)
6. Tracks Service: regional visibility filter (VAS)
7. Rate Limiting Service (Edge)
8. Gatekeeper Service (Edge)
9. Authenticator Service (Edge)

Pros of Making the Switch to Microservices for SoundCloud:

1. No single point of failure, allowing for a simplified deployment of the app and higher application availability [2]
2. Code reusability and the freedom to use a wider array of technology to create a higher-quality streaming service [2]
3. Event sourcing is enabled to deal with shared data thanks to microservices [2]
4. Able to keep the existing system running, while components are replaced, called the Strangler pattern. Functionality is maintained even during transition [4]
5. The work and costs involved in the migration were not outweighed by the benefits for cost in the long term [4]
6. Development teams are specialized for individual features, allowing expertise of the given system [6]

Cons of Making the Switch to Microservices for SoundCloud:

1. Originally built as a monolith and functioned as such for a long time, so transferring to microservices happened in steps, which costs time and money [1], [4]
2. Transferring included decoupling, which is time-consuming and introduces bugs [2], [4]
3. Aspects that the monolith did itself, had to now be manually implemented [4]
4. Added significant complexity to the system [4]
5. As more microservices are added, it becomes harder for developers to track dependencies [6]

References:

[1] https://diceus.com/microservices-vs-monolith/
[2] https://samaratungajs.medium.com/microservices-architecture-used-by-soundcloud-58200ac192e8
[3] https://developers.soundcloud.com/blog/service-architecture-1
[4] https://nordicapis.com/case-study-soundclouds-multi-year-journey-breaking-the-monolith/
[5] https://blog.bytebytego.com/p/millions-of-requests-per-hour-soundclouds
[6] https://www.infoq.com/articles/microservices-evolution-soundcloud/
[7] https://developers.soundcloud.com/blog/service-architecture-2