Microservices

So to start us off on this, we would have to start things off with some of the benefits you'd find with using microservices. So comparing a microservice to a monolithic one, would be given that the size of the application isn't very small to begin with. It would be that we gain an increase in release velocity over a monolithic version. The services themselves will be smaller, more agile, you will be having multiple smaller APIs rather than just a single larger API.

Another benefit over a monolithic architecture is the fact that we could have a part of the service, be faulty or we could with relative ease replace a service or integrate a third party service in that spot instead. To keep with the modularity of microservices is that we can scale independent parts of the overall system, to avoid the wrong parts of the system taking the resources you wanted for a specific part. In a monolithic architecture there's a risk that the wrong parts of the system taking the resources.

There aren't just benefits associated with microservices though and one of them is that we are moving towards a microservice world is that we have an increase in complexity to manage the system. As we split the services apart from each other to increase the modularity we also run into the problem, how do we make the services communicate with each other? Communication between services is part of what makes microservices difficult. Well to do it safely and securely is difficult when it comes to handling the communication.

As the microservices becomes more spread out the risk of someone listening on the communication or even pretending to be a service becomes a real risk factor to take into consideration rather than having the communication happening in memory.. Thus as the inter-service communication rises we become more vulnerable to issues or problems with our network, compared to a monolithic architecture.

A good use case for microservices could be found in many places, but one in particular that I find a bit interesting would be within machine learning or just analytics in general. Here we can use the modularity and scalability quite well, and accessing it through an API. As we also want the ability to compare to previous changes and the ability to deploy new models alongside older models. To this point I find the flexibility given by microservices even at the added complexity involved, as the tasks themselves can be rather complex to begin with.

Circuit Breaker Pattern

So the first design pattern that came to my mind was a 'Circuit breaker', which means that we have a few services that rely on each other, let's call them A, B and C. So since they rely on each other we don't want to waste resources or have a bad user experience by just waiting around for a long period and then just timing out anyway. So to avoid this we introduce a circuit breaker, which essentially has the job of checking in with our services that we have behind it, so that if service B goes down, then it steps in and stops us from accessing any of these services, for a period of time until the circuit can be completed again.

API Gateway Pattern

So a problem that might arise, especially if we convert a monolithic application is the need for communication between the client and multiple services. We could choose to make a direct connection between the client and each service but to an extent we will be coupling our client app to the services, meaning if we want to make changes to the services it would require updates to the client application as well. Another thing to keep in mind is the amount of network traffic we would be using, we could be getting some increased latency when we would need to make multiple calls to the services. Another thing to keep in mind is that as all the services would have to be reachable by the client, meaning they would all be exposed publicly and would increase the risk of security breaches as our footprint is larger in this way.

So what we could do is introduce an intermediary the 'API gateway' which would then be exposed instead of each individual service that our clients would need to talk to. Through this new gateway we could handle multiple different clients and we would also have more ability to make changes or redirect traffic of what is behind the gateway. And yes as we increase and keep the gateway running as we increase the functionalities and such we might just end up getting quite bloated at this stage as well, it is important to keep in mind that we might have to end up splitting our platforms apart, one for iPhones and one for Android and so on to keep them at a maintainable and manageable size.

This also has drawbacks in the form of creating a single point of failure in the sense that our services relies on our gateway to function and thus if the gateway goes down, our apps wouldn't function. It might just be moving the issue of coupling your services not to the client apps like before, but to the gateway instead, so we should keep it finely grained and not build up massive gateways as we are back at the point of a much more monolithic structure at that point, which is something we wanted to avoid by taking this route in development.