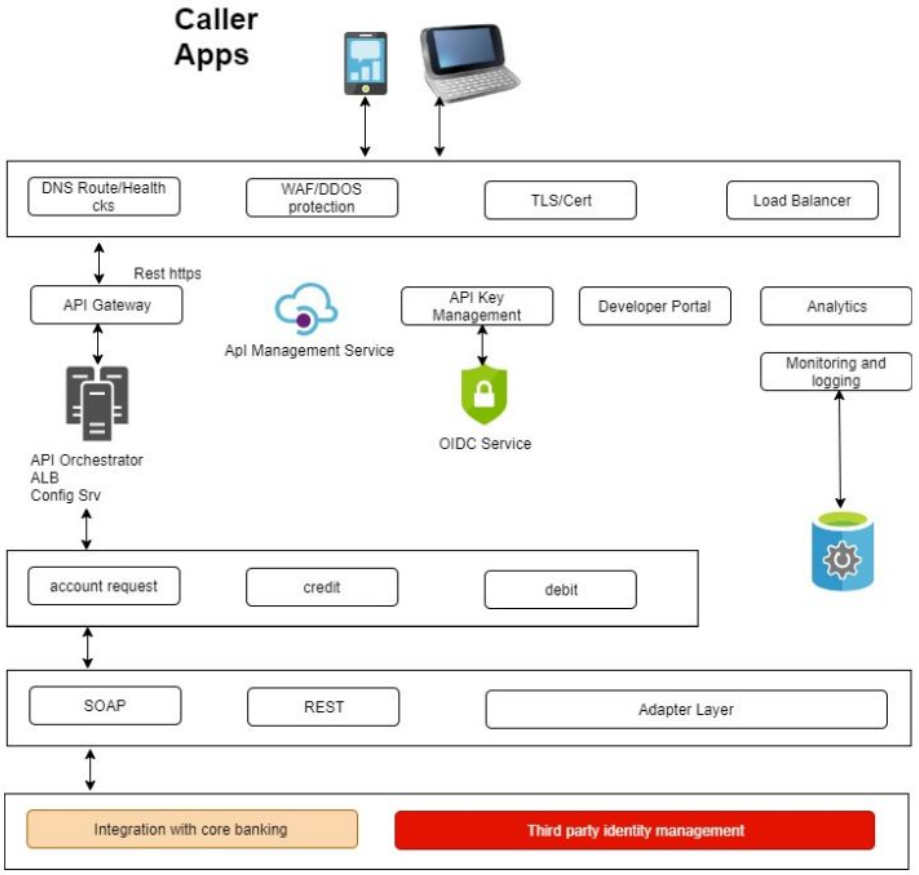
Hi Archana,

As discussed, I have been very busy with my hand over KT session in my current organisation today. Please do let me know if I missed to include/mention any specific context/topic here.

Case study: CS20180505



A common trend across many organisation to modernize applications is to use APIs and decompose them into smaller units that typically live in containers(Docker containers and later kubernetes are there to scale and manage the container scaling and monitoring problems).

We know that the overhead of managing and maintaining our own IT infrastructure is very tedious and so organisations are going with cloud providers due to amazing benefits provided by them.

a) With cloud providers, it's easy to spin various env such as dev, qat for development, testing and integration which results in a lot of cost saving.

b) By using the serverless technologies we can start at low cost and scale as and when needed.

c) By Using the WAF and DDOS shield along with firewall, IAM, we can achieve the all the security standards

d) Build tomorrow an auto scaled platform for future.

For example the resources are :

/custNotifyMgr : Customer notification service: customers should receive a notification in the app about their latest tx.

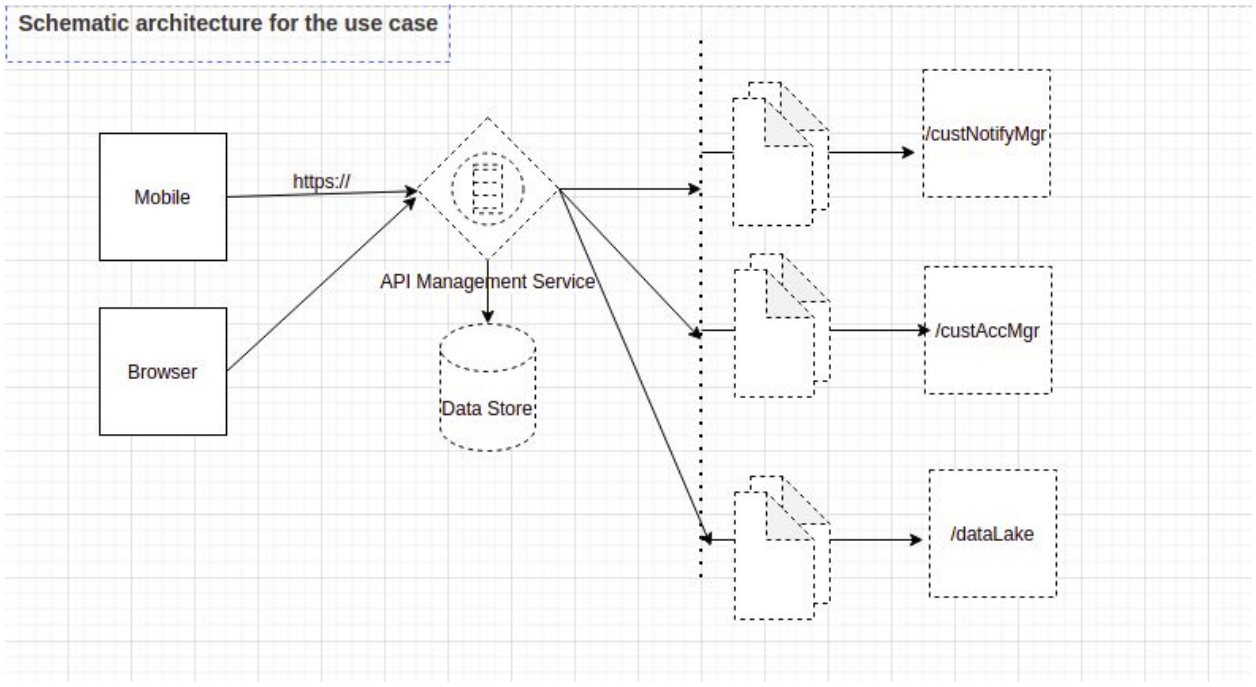
/custAccMgr : This service is responsible for collecting the account details from the customer.

/dataLake: Danske Bank wants to ingest all data generated from any process into their data lake for arbitrary analytics.

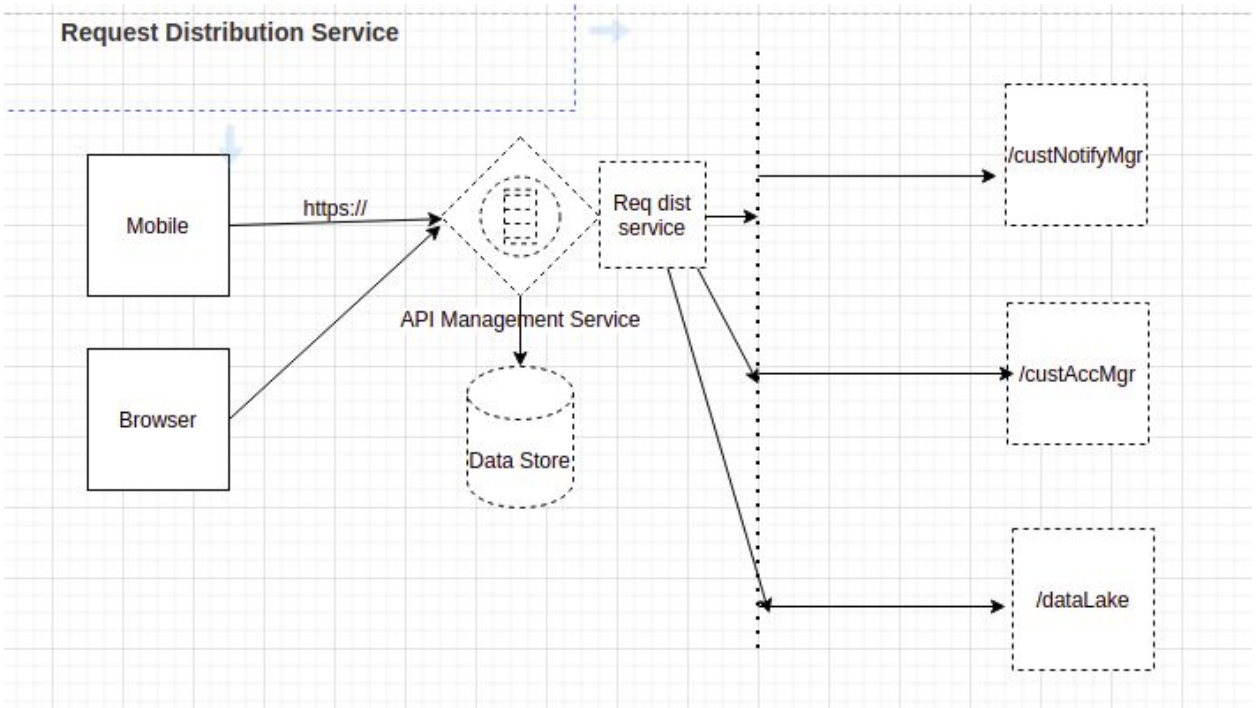
Integration options

Integration via database: Sharing the database also restricts your flexibility to scale and evolve your services. So this is not recommended mostly.

Integration via REST APIs: Most recommended but the problem with this design is the additional complexity to the API management service and coupling on both sides. Although there are self-registration and discovery approaches, managing a recipient list is not the core use case of the API management service.



A better approach would be to externalize the recipient list into a separate Request Distribution Service. Again, this heavy lifting is not the main task of this service.For this use case, integration via REST APIs potentially adds coupling to the services. And it adds heavy lifting to the services that are beyond their actual domain.



Integration via messaging: The API management service can now send the **messages** into the topic created. All interested services on the right-hand side can subscribe to this topic.

Its really a good design called topic-queue-chaining. That means that you add a queue, between the msg broker topic and each of the subscriber services. As messages are buffered persistently in an queue, it prevents lost messages if a subscriber process run into problems for many hours or days.In contrast to REST APIs, a messaging system takes care of message delivery outside of your service code.