Building a digital bank

with Thought Machine and Vacuumlabs



Contents

3

The clear need for a new banking tech stack

5

Sealing the technology gap with tape and hope

8

Cloud-Native modernises the core

12

Vault by Thought Machine

21

Conclusion

The clear need for a new banking tech stack

Banking technology has evolved significantly over the past century and we have seen several major shifts in the core tech stacks used to power banks. Crude technology such as paper ledgers have been in use for centuries. It was only in the 20th century that banks began digitising these records.

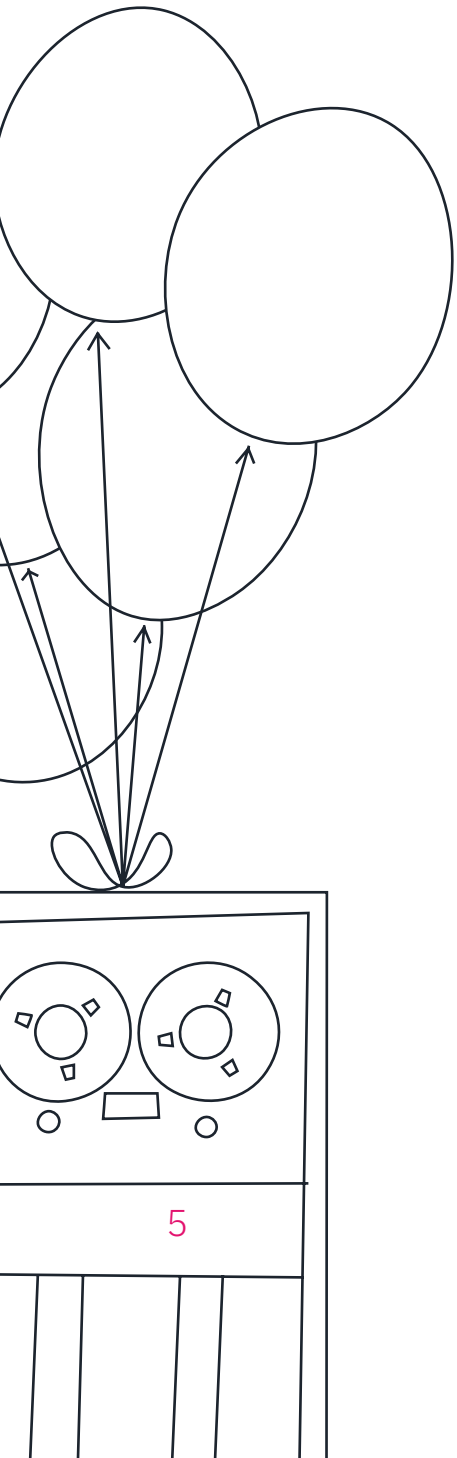
This typically involved recording changes in branches, that close at 5pm, at which point end-of-day positions are calculated and collected in a 'general ledger', which acts as the centralised repository for accounting data in the bank. Server communication was slow and unreliable. These early core banking systems were monolithic in their architecture and designed to run on expensive mainframes; the only machines capable of handling the number of client-server sessions required to service all branches belonging to a given bank. As banks began to develop new products and services, they found that their early core banking systems were rigid and hard to change.

As we moved into the 1980s, consumer banking evolved to be less branch-centric, and the first-generation systems were extended to support emerging channels such as ATMs and call centres. Over time, the core banking system evolved into an architectural mess that just about worked. Of course, this meant that every change was becoming more complex, more time consuming and more costly with ever larger teams required to support and maintain the systems.

Compounding the problem, core banking systems are often designed to only run when the core ledger is closed, outside the hours of 9am and 5pm. As consumer demands changed, and 24/7 banking became the norm, third generation systems generally adapted by shortening the time the core was shut for, and by giving the system the ability to run in 'stand-in' mode so that payments were buffered as the bank transitioned over the end-of-day process.

Riding the wave of emerging web and online banking technologies, another generation of core banking systems emerged, which featured customisable product engines, making change cheaper and less risky. However, since these were often tightly coupled to their respective UIs it was often challenging to compose this newfound flexibility in novel ways. They were generally written in more modern programming languages and deployable into application servers which paved the way for the banks to ditch their expensive mainframes. These applications were generally stateful, and relied on session management, making it difficult to scale in any way other than vertically. These systems still largely used batch-based processing and remained monolithic in style. Since they were no longer tied to expensive yet effective mainframes, they were typically less resilient when compared to their predecessors.

Sealing the technology gap with tape and hope



With multiple generations of core banking platforms on the table, banks started to plan their digital transformation with three equally problematic strategies in mind:

STRATEGY 1

Shimming the mainframe

Some banks have gone down the route of installing software directly onto the mainframe that can expose previously difficult to extract data via modern APIs. However, this is a dangerous trade-off since the underlying systems lack the elastic scaling properties to handle the resulting volatile workload volumes. Banks could inadvertently make their services unavailable by triggering a denial-of-service (DDoS) attack on their core systems.

STRATEGY 2

Hollow out the core

This is rapidly becoming the de facto strategy, and involves pulling the product engine, along with other key capabilities, out of the core. Banks can rely on more modern products to solve some of the shortcomings of the legacy core, however the downside is that there is an increased operational complexity and integration challenges. Additionally, the proliferation of these tactical systems causes data silos, which lead to additional overheads as data provenance and reconciliation concerns.

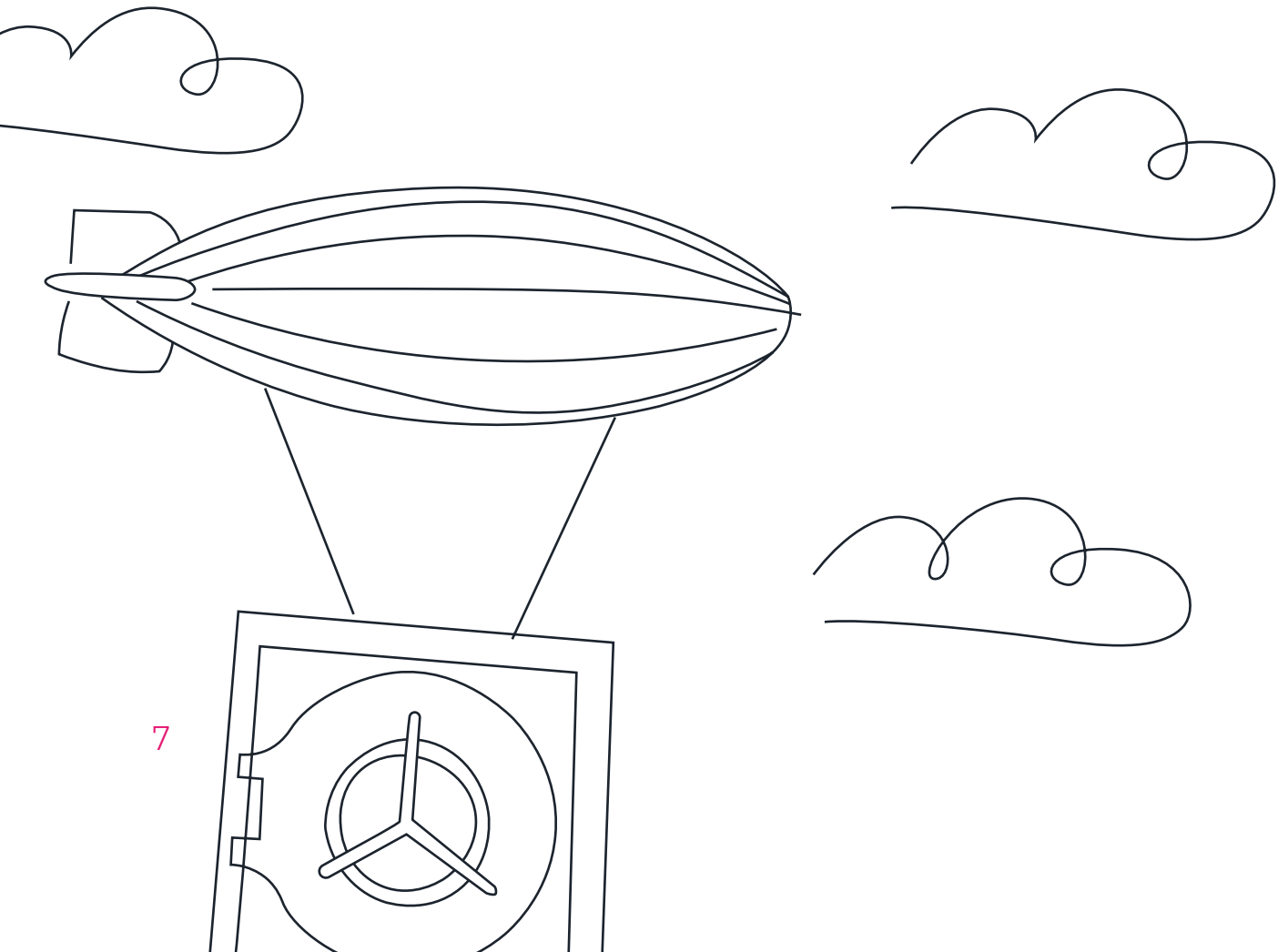
Some might be asking, why do these cores simply not refactor their existing systems, splitting up the monolith into a microservices-based architecture?

A monolith has a single physical clock, and therefore easy access to a single global (total) ordering. As the monolith is split up, we lose the common clock, and as result, the ordering. This means correctness becomes difficult to uphold. Application logic may have to be considerably refactored to handle race conditions and bottlenecks caused by network latency. In addition to this, it can be easy to fall into the trap of simply creating a 'distributed monolith', and as a result being burdened by the worst attributes of both paradigms. Here is where truly cloud-native events stream-based messaging will become helpful, but more on that in the next chapter.

STRATEGY 3

Lift and shift on to the cloud

In the first decade of the 21st century, facing increasing pressure from the market to 'move to the cloud', banks and their vendors often employed a lift and shift strategy taking their monolithic app server, containerising it and deploying it on the cloud. Whereas this technically qualifies as a cloud strategy, we could argue it isn't a very effective one, as we haven't leveraged any of the benefits of being on the cloud, notably, elastic scalability - we are essentially dealing with the same monolith, but now in a more expensive data centre!



Cloud-Native modernises the core

The mainframe systems suffer from having their core ‘hollowed out’ making them too passive. Later systems suffer from being too tightly coupled to their interfaces, making them a closed box and a result, difficult to change. An emerging term for latest generation core banking systems is the “headless core”. A system no longer bound to a green screen terminal window or custom app. It interacts with other parts of the bank through streaming APIs, always up, always accurate, and designed for the 24/7/365 world of the 21st century.

	MAINFRAME MONOLITH	HYBRID MONOLITH	CLOUD-NATIVE
Emergence	1970 - 1990s	Late 1990s	Late 2010s
Product engine	Static	Parameterisable	Hyper-flexible
Messaging	Batch	Batch + Events	Stream
Interface	Terminal	Specific channels	Headless
Connectivity	Server/Client + RPC	API + File transfer	API only
Portability	Mainframe	App server	Cloud/Agnostic
Availability	Single datacentre	Multi-zonal	Multi-regional
Architecture	Monolith	Monolith	Microservices

Running cloud-native software is significantly more efficient, both in terms of performance and staffing costs. Maintenance duties are reduced to near zero: a core principle of cloud-native is the ability to "self-heal". Malfunctioning instances are spun-down automatically and replaced by new ones. Many of the tasks traditionally associated with running IT infrastructure are redundant. IT staff are left to focus on innovation.

Cloud-Native software uses microservices. The application is split into autonomous chunks, which communicate via APIs. Microservices are inherently more robust: a problem arising in one is isolated and contained. The others are unaffected, so the system can keep running despite a single failure.

Cloud-Native software is extensible. This means it can expand and contract on demand. The cost of integrating with third-party services is reduced with cloud-native software. APIs make connectivity easy. Adding a new payment service, or artificial intelligence engine, can be done in weeks - impossible with traditional software architectures.

Continuous integration and delivery

And let's not forget the advantages of continuous integration and continuous delivery methodologies in a fully automated environment. In the age of monolithic design, continuous integration and delivery was a challenge because everyone was working on a single entity and breaking one part of the monolith would often result in the whole thing falling apart, especially if there was a lot of spaghetti code all over the place. With a microservices infrastructure, breaking one part of the chain with continuous integration resulted in a partial temporary problem rather than a catastrophic collapse:

- Applications evolve through incremental improvement. No longer will a bank risk being marooned on out-of-date software, unable to upgrade because the gap has grown too large.

- Innovation is made easy. Product designers can make valuable changes to the customer experience on the fly. A/B test. Launch new products in weeks, not months.

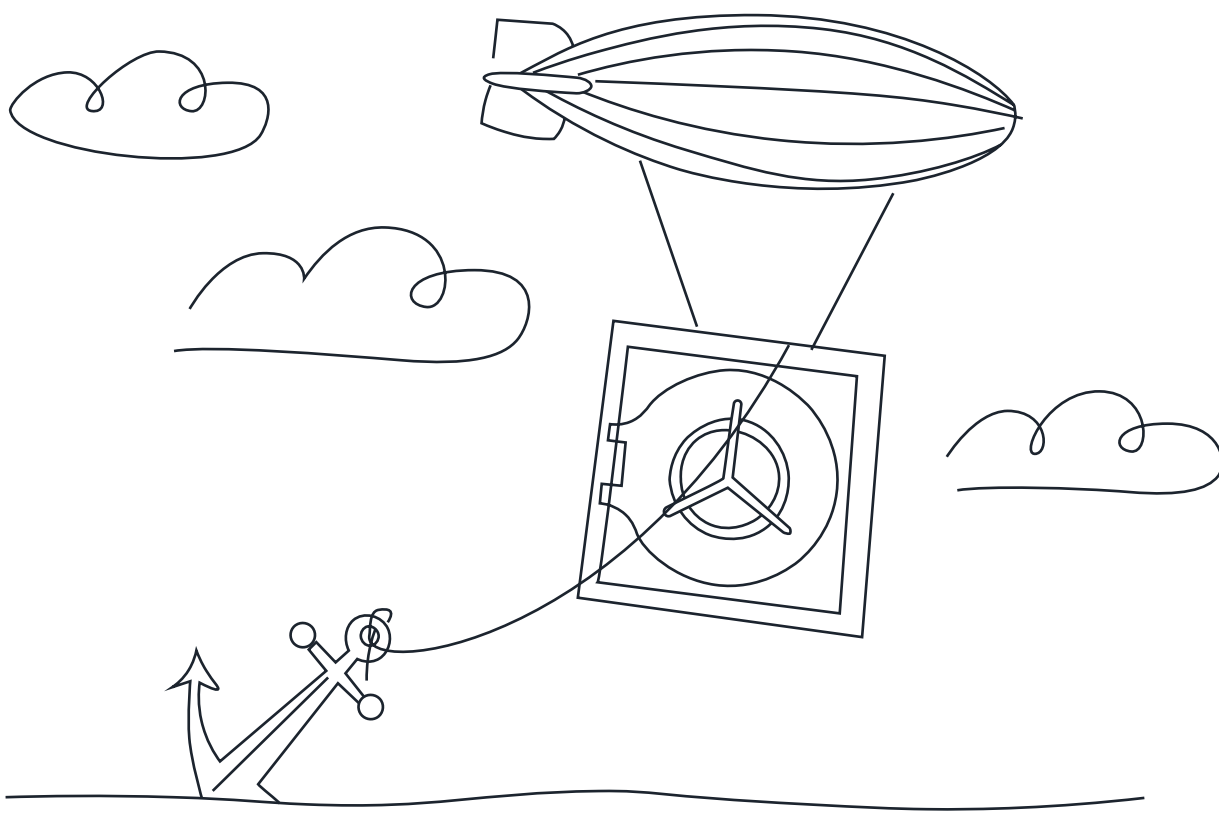
- Improved stability. Introducing small, frequent changes is far more reliable than Big Bang upgrades once every six months. We all know banks' track records on these mega-releases – they have the potential to ruin their reputation.

- Through continuous delivery, features can be delivered daily, if not multiple times a day. Any bugs can be easily identified and changes rolled back. Banks undertaking Big Bang upgrades will add many features at once. If one feature breaks the system, it becomes very difficult to identify which caused the break.

Reliable enough for regulators

Only until recently, cloud and microservice architectures were considered “cutting edge”. Now, more cloud vendors are working closely with regulators and partnering with banks. They are also doubling down on their infrastructure as the networks become faster and more reliable. Many banks have already deployed production systems into cloud environments. Tooling and frameworks are emerging that bring the once theoretical side of distributed computing into everyday practice, such as Kubernetes, Istio and Kafka. Databases can now scale with the application, paving the way for true linear scalability.

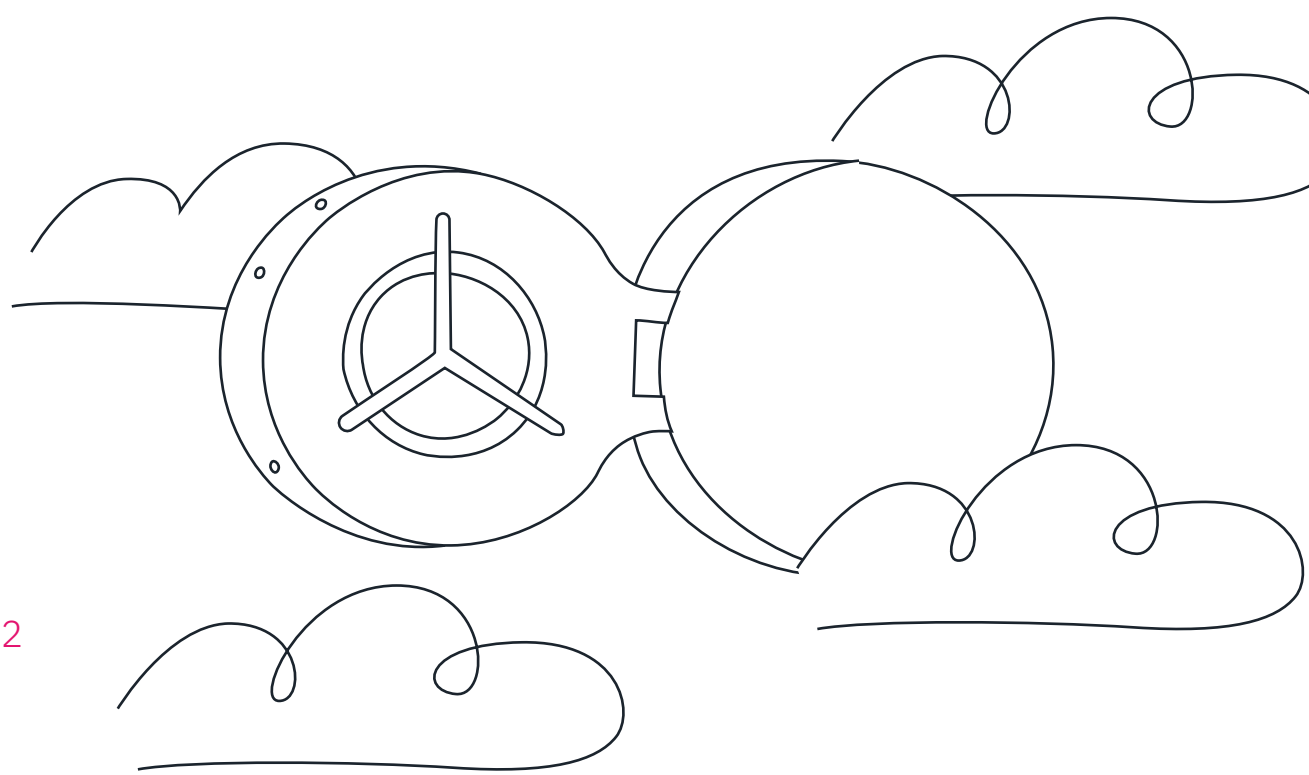
The cumulative result is that we are undoubtedly at an inflection point. It is at this inflection that Thought Machine was founded to build a cloud-native core banking engine with all the requirements discussed so far. Say hello to **Vault**.



Vault by Thought Machine

From its inception, Vault has been written purposefully as a cloud-native core engine by Thought Machine. Vault is built around APIs using a microservice architecture. The services within Vault constitute a significant portion of all the functionality required to run a bank.

At the core of any bank are its products as these define how the bank operates for the financial management of a customer's assets and liabilities. Through its **Smart Contracts** system, Vault can be configured to run any type of retail bank product, no matter the complexity. Vault includes all the key products of a retail bank, such as current accounts, savings, loans, credit cards, and mortgages. The additional services, provided by the bank or other vendors, connect through Vault's APIs to complete a seamless, frictionless bank.



What makes Vault different?

1. **Security**: Security is baked in with fine-grained role-based access control to the APIs, database keys stored in a hashi vault and even distroless linux. Naturally, all data in the Vault is encrypted at rest (in the database) and in transit between the services and within browsers and mobile apps.

2. **Scalability**: Vault can efficiently run a bank of any size. It can handle any amount of customers and can handle transaction volumes in the tens of thousands per second.

3. **Run cost**: Using the cloud means a bank only pays for the exact amount of hardware it uses.

4. **Change cost**: Vault's architecture means that new products can launch quickly, and changes in products can take place with minimal disruption. Upgrades can happen with zero downtime and much simpler rollback if any issues arise.

5. **Flexibility**: Vault has been designed to run any type of bank. It does this through a configuration layer which allows banks to program precisely the functionality they require.

6. **Lack of friction**: Vault offers a simple, comprehensive set of APIs for its key services. Plugging into new Vault services often takes days, not months or years.

7. **High availability**: Through the Kubernetes orchestration layer, Vault is robust against hardware and software failure. If a server goes down or crashes, Kubernetes simply routes traffic to other servers until a new server is brought online.

Flexible to the core

Vault's Configuration Layer enables a bank to achieve a wide scale of customisation without changing anything in the underlying platform. This is highly advantageous and a key part of how Vault's architecture is a counterweight to the "spaghetti" that arises in other systems when customisation and platform functionality are not separated. Vault also comes with an extensive developer SDK and developer portal, which shows how developers can build products using Smart Contracts, build workflows, add extra services, and, in general, complete the bank's build:

- **Smart Contracts**: These lie at the heart of Vault and define all the products in the bank. Banks are free to adopt the existing retail banking product library or custom build new products using Vault's integrated smart contract builder tools.

- **Workflows**: These define the operating model of the bank. Processes such as KYC, product opening, credit scoring, fraud, refunds, collections etc. are defined by the workflow engine.

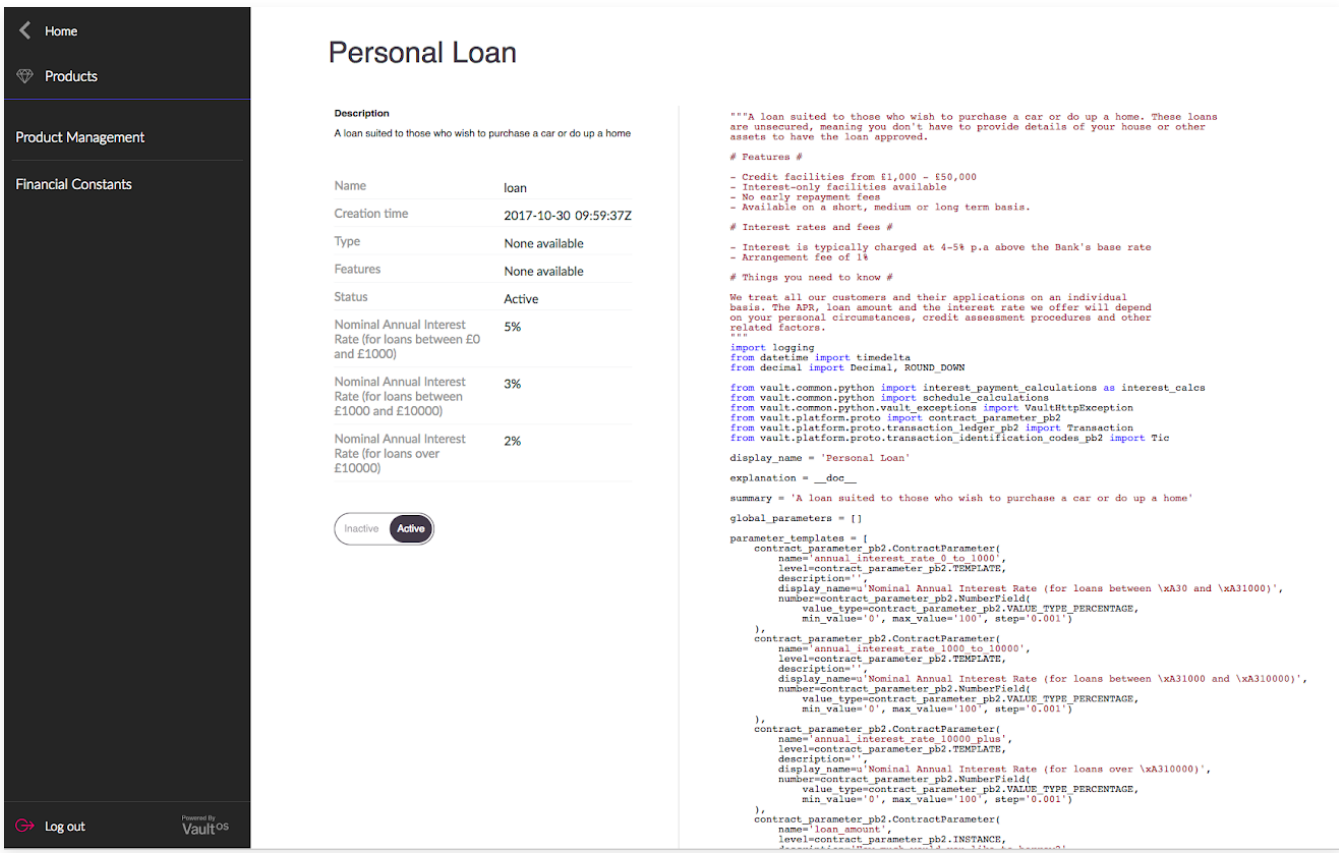
- **APIs**: Allow the bank to connect its own services, and to build customer apps and websites that connect to Vault and offer the whole range of Vault customer journeys.

Vault is resilient, containerised core banking software with zero or minimal dependencies on other applications. A microservices-based architecture, where each service is cohesive, loosely coupled, and self-healing. It guarantees a level of resilience, availability, and upgradeability that the system architectures of previous years do not have.

Beyond the core with Vacuumlabs

Vault is a “headless core”, which means it is agnostic about what apps and interfaces are used to access it. Whether a client is launching a new challenger bank, building a new banking product, or completely replatforming their existing bank stack, they will need a guide by their side to help them build out all the necessary interfaces and applications needed to achieve the experience they want for their customers.

Vacuumlabs specialises in building digital products and tools for fintechs and banks. We have been doing this for challengers like Twisto, Cledara, Railsbank, and even worked with Thought Machine to help integrate its core platform with various third party providers into one cohesive app experience.



Vault smart contracts, written in code, are used to define the business logic and financial operation of the bank's products on a single platform, enabling centralised product configuration

This example image above demonstrates how Smart Contracts can be used to build a personal loan product. Banks create a set of Smart Contracts to represent its product catalogue. All instantiated contracts are recorded on a contracts ledger, which serves as the single source of truth for all active products in the bank. By interfacing with Thought Machine's signing service and posting service, customers can access a variety of products, all written as code.

What exactly can and should be integrated on top of a core engine for a full customer experience? It can be just some of the following, which we have already done in conjunction with Vault:

KEY RESPONSIBILITIES	KEY INTEGRATIONS	ENGINEERING
Payment devices & cards	Card processor	Kotlin
Customer profiles	Identity Verification	Kafka
Accounts and saving rules	Name screening	GraphQL
Install and monitoring	Messaging and email	React Native
AWS and Kubernetes setup	Data analysis and PFM	TypeScript

When building the next generation of cloud-first banking and fin-tech, one must first start to think like a technology company. At Vacuumlabs, we help clients enter this new world. Technology Companies, the ones that have dominated their markets, operate from a robust cloud-native basis. They harness all the benefits of speed, resilience and innovation that cloud technology provides. They do not suffer from downtime, restrictions in innovation, or lack of computational power.

Vacuumlabs provides highly-skilled cloud-native engineers for clients to build on top of core systems. We understand the benefits of this new paradigm shift and have considerable experience working with financial services technology to get to production rapidly and seamlessly. In a recent project, Thought Machine and Vacuumlabs facilitated a live card payment in a physical retail store. The process took less than nine months to deliver.

Our joint signature approach

Vacuumlabs works with clients to build banks from inception to production. We deliver world-class software development expertise and strategy for those clients. Together, the companies deliver on design, requirements and partnership programs.

Day one: Client's own instance

At the very start of the engagement, clients get their own Vault instance. This allows them to start creating their bank by adding apps, connecting other systems, building products, etc. This is a cornerstone of how Thought Machine and Vacuumlabs work – by having a working instance at project inception, the bank can get an in-depth feel as to how everything works and build the bank in a truly agile fashion.

In pre-production we assign a delivery team specific to the bank. This team will stay with the bank from project inception through to production and beyond. The delivery team will help the client set up instances and test environments, set up products and workflows, decide on vendors, and support them throughout the process.

Configuration

Vault is highly configurable, enabling banks to build exactly the proposition they want. Vacuumlabs' team can write smart contracts, workflows, and configure the bank through Vault's developer SDK:

GREENFIELD DIGITAL BANK	Adopt existing retail banking products out of the box and add network partners to complete the bank. All that remains is to build the customer proposition and build the digital front end. This is when Vacuumlabs comes in and works with the client from design to production build.
NEW PRODUCTS IN A BANK	Banks can use Vault to launch new products without the need to launch an entirely new bank. Vault can be connected to existing systems within a bank such as payments, KYC, credit scoring etc. Vacuumlabs delivers the integration and custom development on the project.
REPLATFORMING THE BANK	Vault's Smart Contracts can exactly replicate the financial operation of a bank's current products. All customer data can be moved, including transactional history using the Vault API. Often, improvements in the operating model are desirable or new apps need to be custom-built or modified. Vacuumlabs is here to help the client through the process.

Ongoing support

As the first instances are deployed and prepared for production, Thought Machine and Vacuumlabs work together to test out whether everything is working correctly and are immediately able to give feedback about any inconsistencies. We have daily and weekly sync calls with clients and our team members speak via Slack or video calls regularly. As new features become available on Vault, we align our sprints to maximise collaboration and prevent overlaps or conflicts. We even track issues in the same JIRA instance to link directly to the source of problems and maintain issue resolution flow.

With Vacuumlabs, the client has a continuing partnership to help the client build a suite of integrations and apps on top of the Vault. And as new features become available or market demands change, we continue this ongoing support and journey together.

Conclusion

Banks used to be pioneers in technology and this is a root cause of the current problems. By being early they never had the chance to start again with fresh thinking and new designs. The cost of legacy systems is astonishing; tier 1 banks spend billions of dollars just to keep their systems working, many employ tens of thousands of developers and support staff with too little to show in innovative, customer pleasing experiences.

Only a purposeful move to the cloud-native technology can solve these problems.

As the digitisation of banking continues and the desire for traditional banks to escape their legacy systems in the quest for innovation grows, it's vital to find a partnership that can deliver a fast, scalable solution that mitigates the risks that are usually associated with these projects. Vacuumlabs engineering combined with the power of the Vault by Thought Machine is already driving success for a number of banks.

Advocates of archaic core banking systems maintain the argument that there's simply no option to migrate over to modern, cloud-native systems. They argue that legacy code is inexorably linked to critical systems, and a migration effort would cause untold damage. This couldn't be further from the truth. Forward-thinking, long-term organisations and pionering challengers are already making the move.

Will you join them?



WRITTEN BY MARCEL KLIMO

Tech Evangelist at Vacuumlabs
marcel.klimo@vacuumlabs.com

With thanks to Peter Dudbridge (Dudders), Director of Architecture, Thought Machine, for his expert views in this blog which contributed to the content of this report.

vacuumlabs

Vacuumlabs provides teams of agile, remote by nature, software engineers, and designers with years of experience building fintech, cryptocurrency, online marketplaces, and digital products.

Every step of the way, our experts guide our clients, ensuring they can most fast, learn, and iterate. When it comes to world-class product development, we are here to be your guide and partner.

Thought Machine

Thought Machine builds core banking technology enabling banks to deploy modern, cloud-native, systems. We solve the banking industry's primary problem: reliance on outdated IT infrastructure. This cripples the industry's ability to provide its customers a service they deserve. Thought Machine's clients include Lloyds Bank, SEB, Atom Bank and Standard Chartered.


vacuumlabs


Thought
Machine