Chapter

Middleware Patterns for Cloud Platforms

Gary S.D. Farrow

Abstract

This chapter explores how traditional system architectures are being affected by the emergence of 'Uber' style platform models that provide business services with huge global reach. The specific demands and characteristics of such platforms are discussed which in turn dictate their technical requirements. The chapter will explain how middleware technologies have evolved to support today's requirements for such massively scalable platform solutions. The latest preferred architectural paradigms dictate the use of micro-services and APIs are central to the design of such platforms. Similarly, event based architectures are another key paradigm that must be supported. The role of modern middleware and cloud technologies to support these newly dominant paradigms will be explained. Key architectural patterns pertinent to global platform solutions are illustrated. The role of modern middleware in fulfilling these patterns is highlighted using real-world examples from the field of open finance.

Keywords: Cloud architecture, migration patterns, API ecosystem, event-based architecture, microservices, cloud migration, PSD2, open banking

1. Introduction

This Chapter describes advanced patterns relating to the use of cloud platforms in hosting IT solutions. The context for the patterns is the evolution towards open information ecosystems, mandated to a large degree by regulatory initiatives such as PSD2 [1], but also by competitive necessity. In this future business environment, there is an expectation of a significant increase in transaction volumes as new and innovative services become available for consumers.

The Chapter describes why cloud platform are the essential technology to provide cost effective scalability for IT solutions. The patterns highlight how new application components in the cloud are used in conjunction with existing on premises applications in a hybrid approach to deployment. Further, the Chapter also highlights how the patterns can then be used as part of a phased, but fully complete, migration to the cloud. Finally, specific real world usage scenarios for the patterns are highlighted.

The patterns are presented in a cloud vendor agnostic way and can be implemented in any of the key cloud provider technologies; Amazon Web Services (AWS), Google Cloud Platform (GCP) or Microsoft Azure.

The Chapter is structured as follows; Section 2 provides the background in terms of the business environment and the associated business drivers that necessitate the move to cloud. It further explores the technology perspectives of cloud that provide the specific advantages over conventional infrastructure technologies to support

the emerging business environment. Section 3 introduces the definition of a cloud platform in the specific business context outlined.

Section 4 provides the underpinnings of the key cloud platform patterns in the form of relevant established architecture patterns and outlines the essential building blocks for cloud solutions. Section 5 then uses these underpinnings to describe the proposed cloud platform architecture patterns. Section 6 illustrates the use of the cloud patterns to achieve a migration from conventional IT solution deployment via a multi-phased approach. Finally, in Section 7, real-world scenarios are described to which the cloud patterns are directly applicable.

2. Background

Cloud computing has revolutionised the provisioning of infrastructure for IT services. As the maturity of the cloud offerings has increased, the richness of the of capability has progress from, initially, Infrastructure as a Service (IaaS), through to Platform as a Service (PaaS) and then Software as a Service (SAAS). The latest generation of cloud services relate to the availability of cloud platforms; domain specific applications connecting users of a particular services with the service providers via the concept of a cloud platform. There are numerous examples now appearing but, one of the earliest and a classic example of such is Uber.

The Chapter introduces some key business drivers for the use of cloud platforms. Specifically, the context of regulatory changes relating to open banking are used to illustrate trends in financial services domain. Its consequences in terms of impacts to IT system non-functional requirements, particularly those relating to the ability to scale on demand and cost effectively, are highlighted. This creates a problem for an organisation's IT function in supporting these trends.

The use of cloud technology and cloud platforms is now ubiquitous in most organisations IT architectural thinking, with the promise of providing:

- On demand and self-service characteristics; hence being suitable for agile delivery lifecycles
- Highly scalable architectures supporting platforms having huge global reach
- Capability to store of huge volumes of data and derive useful insights to inform a variety of downstream services

Various patterns for the migration of components to the cloud have been identified previously [2]. These have focussed on basic technology migration patterns such as:

- Re-Deployment
- Cloudification
- Relocation

This focus of this Chapter is in defining advanced, application migration patterns that exploit the advantages of cloud computing for use in emerging and future open information ecosystems. The patterns highlight the essential adoption and use of cloud platforms through:

- Enabling the caching and aggregation of customer data from which insights can be determined and further support downstream customer services

- Catering initially for a hybrid co-existence with 'on premises' IT systems and services such that these can be retained in the short term and provided cost effectively

- Ultimately supporting the complete migration of IT systems from on premises deployment to a cloud platform

2.1 Business drivers

The introduction of new financial market regulation, notably the revised Payments Services Directive [1] (or PSD2), has mandated banks to open up account information and payment services to third parties. The regulation is considered an important enabler for the creation of new and innovative customer propositions. PSD2 is recognised as a trigger for the wider concepts of 'open banking' and, beyond this, 'open finance' in which ultimately a rich variety of financial services are accessible to an ecosystem of third parties comprising third parties, business partners and industry bodies.

Open banking has led to a corresponding rise in financial technology organisations – namely the "*fintechs*". Indeed, information pertaining to the take up of open banking services confirms that 94% of fintechs view open banking as a major area of opportunity [3].

The net effect of this is that there is likely to be a growth in financial transactions accessing customer accounts as new services, founded on the open access regulation, are brought to market. This presents a challenge for financial institutions; that of scaling their IT systems cost effectively to support the new market dynamics with increased transaction volumes.

To summarise:

- Within financial services, open banking is recognised a key area for driving business growth

- To enable rapid pace of change and innovation, businesses must adopt technology that scales effectively and enables an engaging customer experience

- Cloud technology is considered essential to achieve scalability of the banks underlying systems to meet the demand for future services

Organisations are therefore faced with a key problem to address of how to combine the value of their existing mature core applications with the advantages that cloud technology provides. Superficially, they are faced with a number of high-level architecture challenges:

- Do they lift and shift applications to the cloud?

- Do they invest in rewriting applications to take advantage of the cloud technologies?

- Do they migrate to greenfield cloud platforms providing new implementations of core services?

This Chapter helps organisations to address these key issues by describing a variety of patterns highlighting relevant cloud platform usage scenarios including hybrid deployments and patterns to support, ultimately, the full migration of services to a cloud platform.

2.2 Cloud technology drivers

In this Section, a precis of the features of cloud technologies is provided. These reinforce what makes a cloud platform suitable for supporting the provision of scalable information services in general and the specific emerging trends in banking. The notable technology features are:

- **Elastic scaling.** As transaction volumes increase or decrease, cloud autoscaling technologies scale the computing resource required automatically.

- **Compact Data Notation using Java Script Object Notation (JSON)**. Compact data representation standard based on name-value pairs. Again this infers requires less bandwidth than other data formats such as XML when transmitting data, making it more suitable for internet usage.

- **RESTful API standards**. A RESTful API uses standard HTTP requests to invoke remote processing on data. REST is the preferred API technology of choice as it is based on open standards and the use of a 'lightweight' stateless HTTP protocol for its requests and responses. This again reduces computing and networking resource requirements and makes a cloud solution inherently more scalable.

- **'No SQL' database technology**. 'Document' style storage databases allow for the storage of data is the same format as which it transmitted, specifically JSON. This approach requires no, or minimal, format translation from data store through to payload, further reducing the computing resources required in supporting a transaction from data deserialization through its transmission and presentation to a consumer.

- **Use of Open Source middleware**. Open Source middleware is prevalent for cloud deployments. Such software licencing models are much more scalable as the software is free, or at the very least the pricing models are better geared to the highly elastic solutions enabled by the cloud. Thus, cloud solution runtime costs are close to that of a linear 'utility' model, rather than having the incremental costs breaks associated with traditional middleware and vendor software. Hence this ultimately more cost efficient.

Consider now a traditional IT architecture that supports banking and other information systems. The underlying customer information will typically reside in a 'system of record', the implementation of which typically fall into one of two categories:

1. A bespoke legacy system, such as a mainframe, developed over many years; often difficult to maintain with rigid release cycles for enhancements

2. A vendor package, providing a complete or modular domain solution. e.g. a core banking platform or payment engine.

In the context of open information access, the ability to scale on demand and at a cost that is linear to the transaction load becomes a key requirement. However,

both of the above implementation options present challenges in meeting the non-functional characteristics of a new open information ecosystem outlined since the ability to scale cost effectively becomes difficult.

One reason for this is that both legacy and vendor product pricing are typically very dependent on supporting hardware and the number of CPUs required. Hence cost breaks relating to hardware and vendor product licencing tend to be highly non-linear. To accommodate the open information ecosystem and expected transaction growth via a traditional IT architecture typically means over-engineering to allow for sufficient headroom in the capacity. Thus, mitigating the scalability risk in a traditional IT architecture is likely to be highly cost inefficient given the wide range of loads that could be experienced.

2.3 Open information ecosystem requirements

The difference in usage profiles between open banking and traditional banking are now explored. In general, traditional banking is subject to highly predictable loads based on:

- A finite customer base for a given banks
- Online usage patterns that are well understood and predictable
- System processing that is based on periodic cycles
    - Daily processing cycles such as overnight batch processing
    - Monthly processing cycles, such a billing

It is reasonable to assume that net transaction volumes will inevitably increase substantially as third parties develop their propositions and these gain maturity in the marketplace. This alone will result in customers interacting with their bank more frequently, albeit indirectly via the third parties applications in '*customer present*' scenarios. Also, there will be an increase in transaction volume driven from the third parties directly. Third parties, having obtained consent from the customer for specific account information, will exercise their right under PSD2 to access that information up to four times daily in '*customer not present*' scenarios.

However, with open banking, transactional loads are likely to be significantly less predictable. The open banking transaction volumes have a more complex and less deterministic relationship with existing customer volumes and their access patterns:

- Customers may employ the services of several third parties and thus a multiplier will apply to the volume of transactions normally associated with a given customer base. This multiplier is difficult to quantify as:
    i. The percentage of account holders that subscribe to use PSD2 services is not yet known
    ii. The number of PSD2 services that customers subscribe to is likely to be highly variable
- third parties will undoubtedly access account information and transaction history without the customer being present up to the limit defined by the regulation.

- Information access patterns are less predictable and determined by the third party rather than via predictable customer access patterns that are well understood by the banks'.

These characteristics translate to specific IT issues for the account information provider, notably:

- How to achieve scalability of the mandated services to meet a, potentially huge, increase in transactions volume
- How to accommodate peak loads at non predictable times
- How to ensure performance and availability of the regulatory interface to support the open information services

3. Cloud platform approach

Given the challenges highlighted for open information access, the role of a cloud platforms in the providing solutions to this problem have previously been identified [4].

3.1 Platform definition

In brief, a platform is a business based on enabling value-creating interactions between external producers and consumers. The platform provides an open, participative infrastructure for these interactions and operates within governance conditions set for them. The platform's overarching purpose: to consummate matches among users and facilitate the exchange of goods, services, or social currency, thereby enabling value creation for all participants.

3.2 Technical service provider platforms

A Technical Service Provider (TSP) is a non-regulated participant in the PSD2 ecosystem. They provide services on behalf of a regulated entity and provide the necessary IT components to implement the required PSD2 services, intermediating between an Account Provider and a Third Party Provider via their platform, as illustrated in **Figure 1**. Standards for PSD2 access to account services (e.g. from the Berlin Group [5]) universally employ application programming interfaces (APIs), these being the de facto standard for B2B interfaces over the Internet. Further, as the ecosystem expands to accommodate broader open banking services, there is an expectation that additional, non-regulatory, services will also be implemented using APIs.

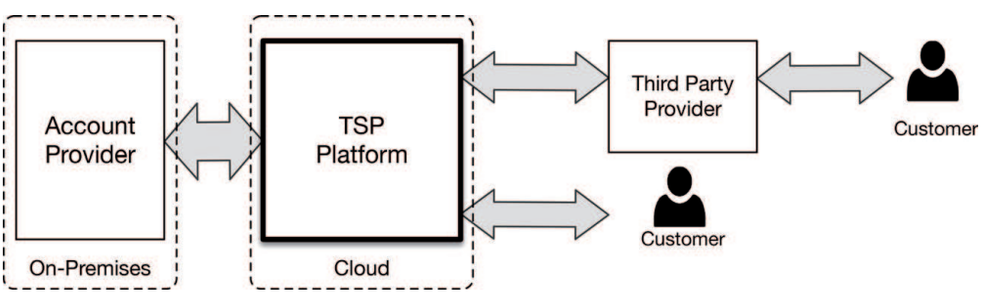

Figure 1.
Cloud platform context.

TSP platforms can accommodate such open banking services on behalf of an account provider.

3.3 Summary

The concept of a cloud platform has been introduced and the associated advantages highlighted. In practice such a platform can either be provided by a third party, known as a TSP, or by the bank themselves in the form of a private cloud. For the purposes of the patterns now described below and their rationale, this distinction is not significant.

4. Pattern building blocks

4.1 Command query response segregation

Command Query Response Segregation (CQRS) is a fundamental design pattern identified by Young [6] and Fowler [7]. Up until recently, the use of this pattern was quite limited and, furthermore, its usage came with caveats about the additional implementation complexity required. Through an implementation of this pattern,

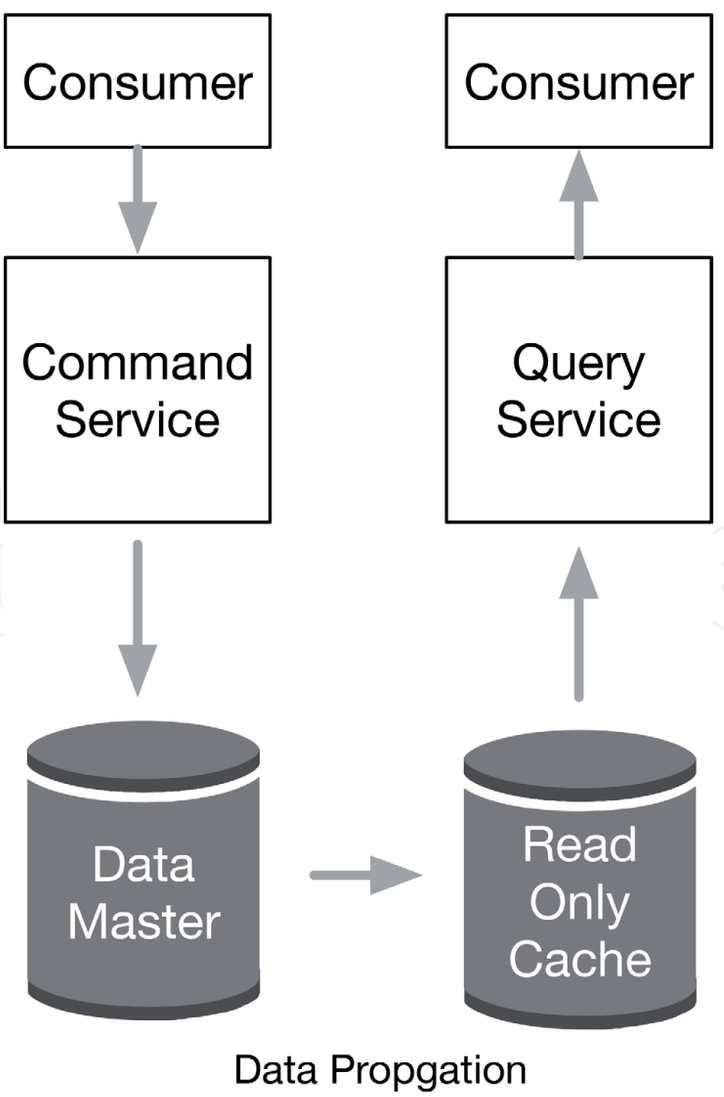

Figure 2.
Abstract CQRS pattern.

it will be shown that certain key benefits of a cloud platform can be realised. The pattern is shown conceptually in **Figure 2**.

In its abstract form, the pattern is very simple:

- One mechanism is used to read data – the *Query* element of the pattern
- A different mechanism is used to write data – the *Command* element of the pattern
- Data subject to an update in the *Data Master* (illustrated) is propagated to the *Read Only Cache* once an update has occurred occur.

The pattern is unspecific regarding implementation.

The significant feature of this pattern is that is reduces loading on the Data Master, since only write operations are performed on this data store. In the context of financial services this is highly significant. Consider the Data Master as supporting a system of record such as an accounting application. Since the majority of transactions on accounts are in fact read operations (typically 80%), by having a separate data cache for read only transactions, this approach becomes highly effective in reducing the net load on the system of record.

In the cloud patterns described in this Chapter, it will be shown how the query service and the command service can be implemented independently and deployed to a cloud platform in a phased approach if necessary. This enables scalability and performance and ultimately can facilitate a complete migration of a system to a cloud platform.

4.2 Publish-subscribe architecture

Publish-Subscribe is an architectural pattern that is exploited in the proposed patterns for cloud platforms. The components of the pattern are illustrated in **Figure 3**. The pattern is fundamentally about message distribution:

- An Event Publisher creates and sends a message
- An Event Subscriber receives and processes messages
- The messages delivery is facilitated by a Publish/Subscribe Engine.

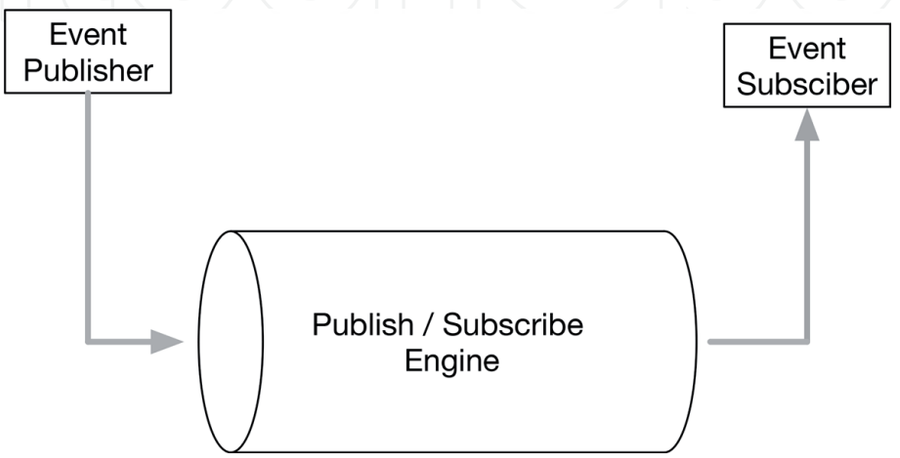

Figure 3.
Publish - subscribe architecture.

The Publish/Subscribe engine manages the distribution of messages based on the set of subscriptions. When an event is published, the engine matches the subscribers, typically based on the assignment of 'topics' and transports the message to the destination accordingly.

4.3 API gateway

A brief description of an API Gateway is provided here for the purpose solely of illustrating its role in the cloud patterns. In short, an API Gateway provides services for the management of APIs. These services broadly equate to a set of policy driven capabilities that dictate the characteristics and behaviour of an API. Typical policies relate to:

- Security management of the API
- Performance management, viz. throttling of endpoint connections

In addition, an API Gateway acts as an audit point and the logging of API usage. A number of commercial and open source API Gateway offerings are available.

4.4 Micro services architecture

The Service Oriented Architecture (SOA) paradigm has previously dominated architectural thinking. This paradigm relied on the constructs of a layered hierarchy of web services to fulfil a request. The services were specified and implemented via strongly typed interface definitions using XML. Similarly the invocation protocol, SOAP, was a verbose XML implementation.

Microservices have a similar concept of an interface definition but this is specified and implemented using a much simpler data typing language, namely JSON with services invocation via a 'lighter', stateless protocol, denoted Representational State Transfer (ReST).

However, whilst there are technical differences in the way that web services and microservices are specified and invoked, the difference in architectural style goes much deeper that the underlying technologies. Specifically, a microservice architecture has the following characteristics:

- It is not a 'layered' architecture in that each microservice should be designed to perform a specific function through from data presentation to the data persistence
- Each microservice should therefore encapsulate all the functionality to support:
  - Presentation of data, irrespective of whether presentation layer is a graphical or 'headless' data payload.
  - Business logic associated with the service. e.g. business validation logic.
  - Data retrieval and update services.
- They employ a 'lightweight' ReST protocol for the invocation of each microservice.

It is useful to emphasise the difference between this and the traditional service-oriented architecture paradigm as this is key to the effectiveness of the cloud platform patterns presented here. In order to compare, **Figure 4** illustrates the typical layering of a SOA architecture. This is shown alongside the concept of a microservices architecture, with each microservice encapsulating a vertical 'slice' through this layering. In its simplest form, the microservice architecture is a series of such vertical slices with each microservice being a completely independent construct and having zero coupling to other microservices.

4.5 Event based architecture

An event-based architecture is another mature architecture paradigm that complements perfectly a microservices architecture by supporting the communications between them. The design principle for this style of communication, again relates to ensuring decoupling between microservices. Independence of each microservice supports the ability to design, build and deploy microservices without impact to other microservices supporting the concept of Domain Driven Design [8].

Thus, rather than create dependencies between microservices using point to point connections between them (i.e. one microservice explicitly invoking another microservices via it interface), using an event driven architecture pattern, a microservice will publish data via an event construct. Microservices that are potentially impacted by the event, subscribe to the event and receive the event and its associated data.

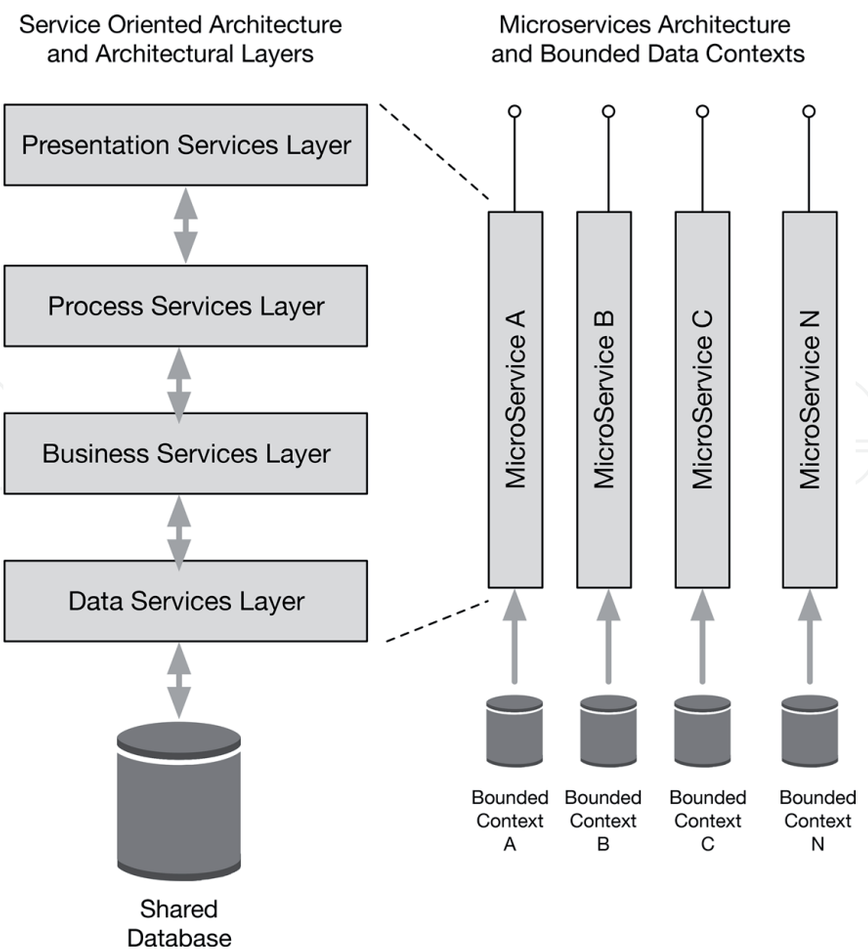

Figure 4.
SOA versus microservice architecture paradigm.

To implement this architecture pattern a Publish-Subscribe Engine component is required. This component manages the publication of events, typically via the definition of topics. Consumers of the events are defined by their subscriptions to the variety of topics.

4.6 Bounded data context

In its general from, a bounded data context defines the necessary data entities and attributes to support a given business domain. This is another idea that is integral to the concept of Domain Driven Design [8]. In simple terms, a bounded data context defines a key domain entity such as a Customer, an Order or an Account. The data attributes for the bounded data context contains foreign keys that allow linkages to other domains. For example, a 'Customer' bounded context may contain a list or collection of 'Account IDs' to define linkages to the Account domain entities that equate to the Account bounded context. Limited hierarchical nesting of the data enables implementation using JSON data type definitions rather than a stronger typed data structure implementation such as XML.

By constraining the data set in this way, a relatively simple data structure can thus support the microservice in respect of its CRUD services. Since the bounded contexts each define a self-supporting and independent data set, this in turn supports low coupling between the microservices allowing for independence of design, through to packaging and deployment.

4.7 No SQL databases

SQL databases relate to a very specific data entity relationship model and associated query model based on tables. NoSQL databases are a technology that provide an alternative to traditional data entity relationship models and storage and support data retrieval mechanism that are different to the traditional entity relationship model. There are a several fundamentally different types: columnar, document (aka object) databases, key-value pair and graph NoSQL databases.

The bounded data context approach is well suited to an implementation using a NoSQL database, specifically the document style. Data can be serialised and de-serialised efficiently without any paradigm shift in the data representation. In this respect, JSON microservice payloads can translate directly to serialised document objects and vice versa.

5. Platform patterns

A prime focus of the patterns for cloud platforms presented in this Chapter is the notion that you for a specific functional service, you use a different approach to update information than the approach you use to read information. The original idea stems from a pattern known as Command Query Responsibility Segregation outlined in Section 6 above.

Consider now the business context attributed to open banking described in Section 2. When deconstructed into its constituent parts, the CQRS the pattern can be seen to be highly useful in supporting organisations in meeting a number of the business drivers that have been identified, specifically:

- To meet their regulatory requirements for open access to their customer's data

- To scale ageing legacy systems cost effectively to meet growth needs
- To ultimately support migration of complete legacy platforms to a new cloud-based platform, helping to meeting the requirements for an agile IT organisation supporting IT changes with a high cadence.

In this respect this Section highlights the following patterns each using elements of the original CQRS pattern to meet a specific use case. The following cloud platform patterns are identified:

- Cloud Data Cache
- API Façade
- Data Hydration

Further, it is shown that, through the sequencing of these new patterns, they can be used to facilitate the complete migration of a on premises legacy system of record to a new cloud platform. The cloud patterns are now described in detail.

5.1 Data cache

This pattern relates to the provision of read only services via a cloud platform. The business context of this pattern is that information services, traditionally provided by an organisations' core systems, such as a system of record, can now be invoked indirectly via a third party, such as is mandated by the open banking regulation. In such a scenario there arises an increase in demand for services. This in turn places demands of increased transactions and additional load on the core system of record.

Consider now that, for applications such as core banking systems, read transactions typically account for 80% of transaction volume. In these circumstances, to alleviate transaction load on the core system, the pattern provides a read-only data cache of data derived from the system of record. As described in Section 2.2, this solution therefore provides a highly scalable solution to this particular business scenario by using the cloud platform pattern, notably through:

- Use of low-cost, open-source licencing models for the cloud component technologies and middleware.
- Avoidance of high cost, monolithic scaling of the underlying system of record having high cost-breaks.

The key advantage of this pattern is that, in response to such increased demand for read only information services, the organisations information services can be scaled in a far more cost-effective way than by scaling the underlying system of record.

The components of this pattern are illustrated in **Figure 5** and their role described in **Table 1**.

5.2 Hydration engine

This pattern relates to population of the data stores cache to support the Data Cache Pattern described above. Each of the populated data stores represents a bounded data context for the microservices in the Data Cache pattern above.

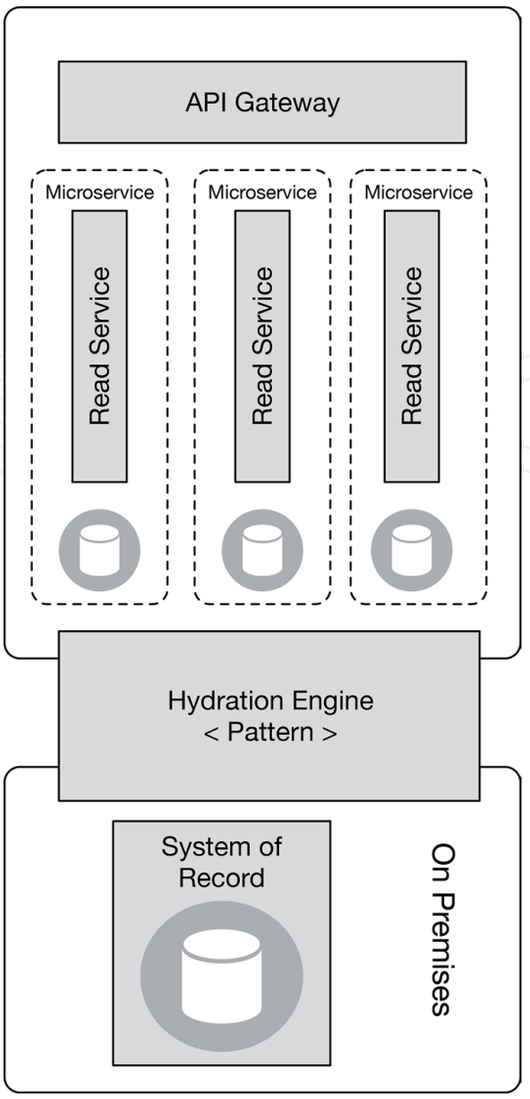

Figure 5.
Data cache cloud pattern.

Component	Role
API Gateway	Hosts proxy APIs for each microservice and acts as the Policy Enforcement Point (PEP) for access to the services.
Microservice	A microservice is associated with each API offered via the Gateway and provides read access to a given data cache.
Data Cache	A number of data stores are provided each relating to a single bounded data context
Hydration Engine	Its purpose is to populate the data caches and keep them synchronised with the system of record. This component is itself a pattern and may have a number of implementation variants.

Table 1.
Data cache pattern components and role.

The population of the data stores relies on the implementation of the publish-subscribe pattern described in Section 4.2 To populate the data stores, a single subscriber microservice is defined for each bounded data context identified.

The components of this pattern are illustrated in **Figure 6** and their role described in **Table 2**.

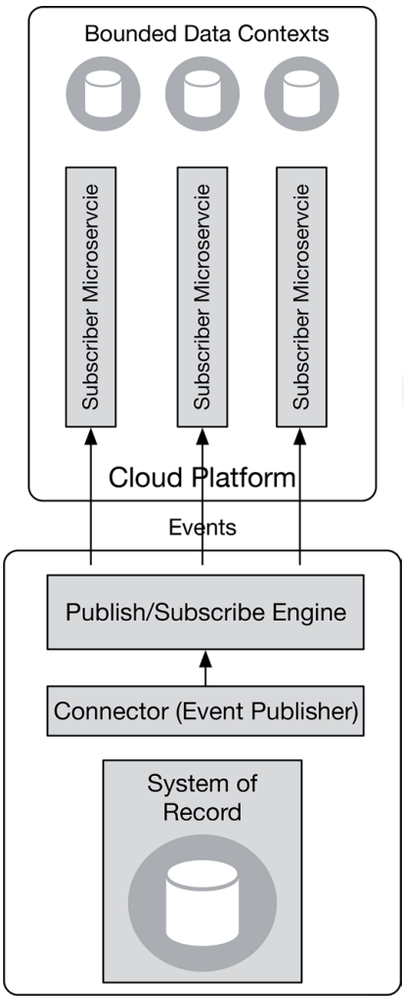

Figure 6.
Hydration engine cloud pattern.

Component	Role
Data Cache	A number of data stores are provided each relating to a bounded data context.
Subscriber Microservice	Each subscriber microservice subscribes to a set of event services sufficient to populate the bounded data context of the data store.
Publish/Subscribe Engine	This component maintains the set of events for publishing data and maintains the set of subscribers to the events.
Connector	The connector provides integration with the source system. The connector detects changes in the underlying data in the system of record and translates these into events for processing by the Publish/Subscribe Engine.
System of Record	This component represents the master application that is the source of the data being cached.

Table 2.
Hydration engine pattern components and roles.

5.3 API Façade

This pattern is a cloud specific implementation of the well-known Bridge pattern [9]. The pattern assumes that there are an existing set of services that provide integration with the system of record. The pattern implementation provisions a set of modern API interfaces, functionally equivalent to those provided by the combination of the system of record overlayed by its existing integration services. Such existing integration services could be implemented by a number of technologies, including:

- SOAP web services
- Messaging services, such as MQ Series, Rabbit MQ, AWS Simple Queuing System
- CICS transaction processing technology

In this respect, the APIs represent a new interface definition and constitute a 'façade' for the existing integration services. The APIs are hosted within the cloud platform, fronted by an API Gateway that provides API management controlled through policies as described in Section 4.3.

The components of this pattern are illustrated in **Figure 7** and their role described in **Table 3**.

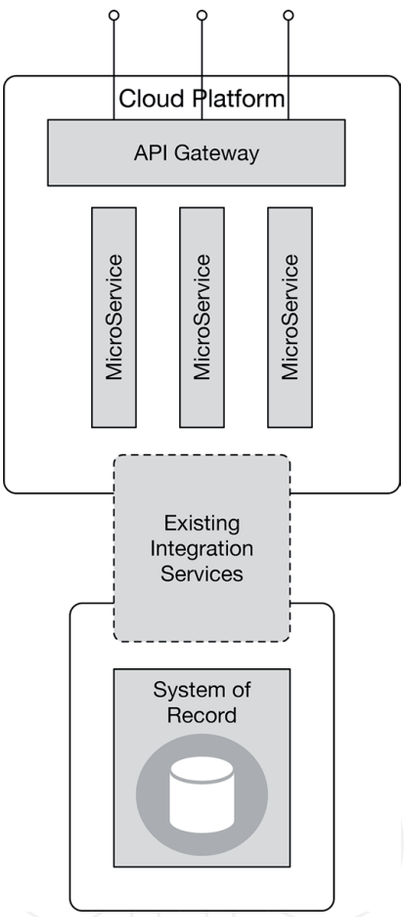

Figure 7.
API facade cloud pattern.

Component	Role
API Gateway	Providing policy based access to the APIs.
Facade Microservice	Provides the implementation of the API interface by consuming the existing legacy integration services.
Existing Integration Services	This component represents existing integration services that are consumed by the new microservices to access the underlying systems of record.
System of Record	This component represents the master application that is the source of the data and the target of data updates.

Table 3.
API facade pattern components and role.

6. Cloud platform migration

This Section provides an illustration of how the three patterns outlined previously can be used to achieve a migration of an on-premises legacy application, such as a system of record, to a modern cloud platform. Three potential phases of the migration are identified, fulfilling gradual 'strangulation' [10] of the legacy platform as shown in **Figure 8(a)-(c)**.

Phase 1 of the migration provides a set of selected read services via the cloud platform using the *Data Cache* and the *Hydration Engine* patterns. This migration step itself can be a phased approach, gradually incrementing the number of the bounded contexts that are supported in the cloud platform.

Phase 2 of the migration then provides complementary write services for the read services and their bounded data contexts. To affect this migration the *API Façade Pattern* is used to support the write services. Having both read and write services for a given bounded data context allows integration in the form of update via events between the system of record and the cloud platform to be switched off.

A caveat to this happening is that the consuming applications must be migrated to use the new cloud platform services and not continue their use of the legacy services. Without this occurring, the architecture becomes complicated by the fact that any updates made via the cloud platform must also be propagated back to system of record. To support this, the system of record must also be a subscriber to events derived from updates via the write microservices. Similarly, any updates made to the system of record must be propagated to the cloud platform. To support the latter, the hydration engine must be retained in the architectural solution at this stage.

To avoid such a complication requires the coordination of:

- Provision of the write services to complement the read services within the cloud platform for each of the bounded contexts.
- Migration of the service consumers to the new cloud platform services as they become available.
- Discontinuing use of the equivalent legacy services.

Achieving a clean separation of write services can be difficult, particularly if there is not a simple correspondence between the legacy and cloud platform services. Similarly, Phase 2 can represent the target state architecture where a residual set of legacy services are retained that existing consumers still continue to use. This is very much a realistic scenario in the cases where it not feasible to change the legacy consuming applications to use the new API based cloud services. A strategy of maintaining the legacy system of record for legacy consuming clients may therefore be necessary. Alternatively, new consumer applications may be built to complement the existing legacy client applications and the legacy clients may ultimately be deprecated.

If the intention is to fully deprecate the system of record, then the migration process can proceed on a per bounded context basis until all the data that was originally managed by the system of record is represented in the cloud platform. The consumers of the legacy services must be migrated to consume the new services offered by the cloud platform as the bounded contexts are gradually

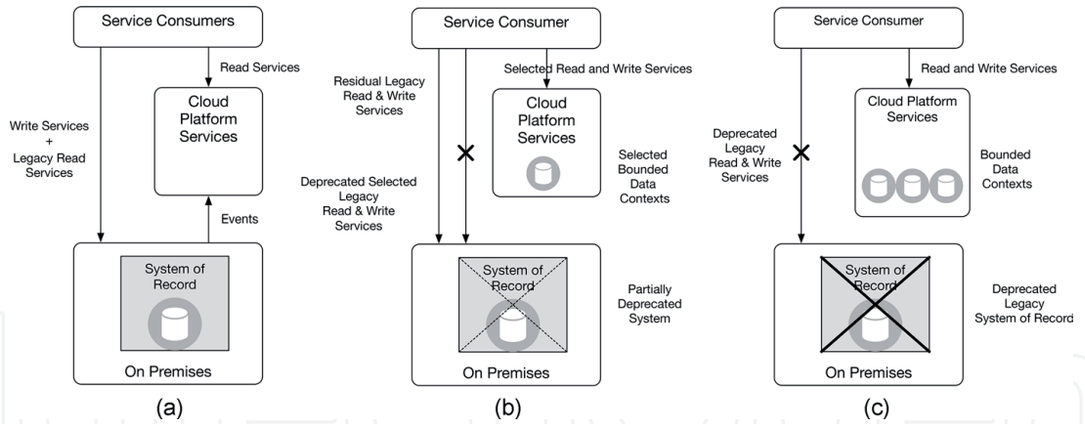

Figure 8.
(a)-(c). Migration to a Cloud Platform via Strangulation.

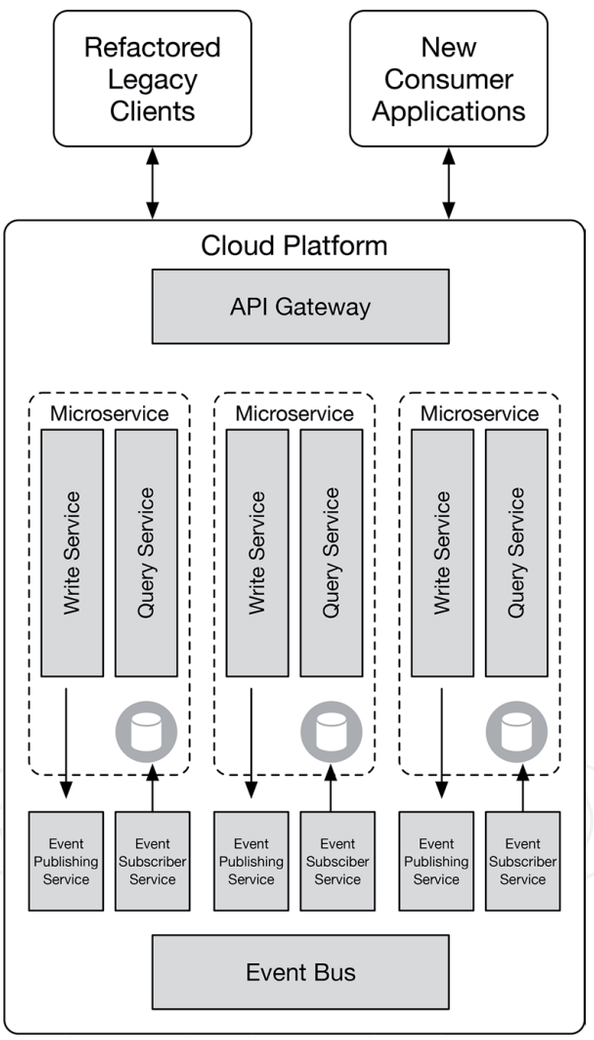

Figure 9.
End state architecture post migration.

migrated. Once the migration has completed, the legacy system of record can then be deprecated. In these circumstances, to replace the legacy client application, refactored or completely new client applications must be built on top of the new cloud platform services. The end state architecture in post migration is shown in **Figure 9**.

7. Pattern usage scenarios

This Section describes four specific example usage scenarios for the cloud patterns introduced here.

7.1 Open banking account and transaction data

Regulatory initiatives such as the PSD2 [1] in Europe and the Competition and Market Authority Order [11] in the UK dictated banks must provide information services relating to customer's account details and their historical transactions to approved third parties, subject to customer consent.

Using the data they obtain about the customer from their banks, the third parties are thus empowered to create innovative, value add, services that entice the bank's customers. These new services create a demand profile for information from the banks that is significantly different to existing customer behaviours; these being typically highly predictable and with a tendency to be based on a point in time transactional need e.g., to check their account balance, to make a transfer. Given that third parties are permitted to access customer information multiple times per day, this will result in a significant increase in transaction frequency from the banks perspective as third parties will take advantage of this to keep their data up to date to reflect a given customer's intra-day transactions.

As explained in Section 2.2, faced with a choice of scaling their existing accounting systems of record to accommodate this increased transaction volume, the bank should implement the Data Cache Pattern described in Section 5.1 and the supporting pattern in Section 5.2 to achieve cost effective scaling to support the increased transaction volumes.

7.2 Public and private API hosting

Once again, in response to the landmark regulatory initiatives for open information access previously described, financial institutions, notably banks, are mandated to provide access to account services to third parties. In terms of the technology to offer these services the de facto architectural style for implementation of these services is that of ReSTful APIs. Similarly, to complement the regulatory services, banks may also choose to offer their own services for consumption by their partner organisations or to monetise additional, non-regulatory services for consumption by third parties.

The net effect of this is that banks need to offer a wide range of ReSTful API services for consumption by external parties. To support these services bank should implement the API Façade cloud pattern of Section 5.3, enabling controlled policy based access to the set of APIs implementing the functional services and leveraging existing integrations to the systems of record where appropriate.

7.3 Customer data aggregation

A third usage scenario relates to the aggregation of customer data. Third parties access and accrue account information for a given customer from multiple financial institutions. This data should be captured according to the Data Cache pattern and serves to support the third party provider in obtaining a convenient and full picture of the customer's financial position. This data supports their provision of value add services to their customers. A key observation is that the cloud platform implementation is provided by the third party provider, not by the account provider, resulting in a demand side cloud platform [5].

A variant of this is that, within a given financial institution, data about a customer may be aggregated from a number of different account systems of record (e.g., current account savings account, credit card account) via the same Data Cache Cloud Pattern. Conversely, this usage scenario represents a supply side platform. The two styles of platform are illustrated in **Figure 10** below.

7.4 Legacy system remediation

A common problem for banks and other financial institutions is that of vendor lock-in to legacy technologies caused by a variety of circumstances:

- Low risk appetite of the organisation to undertake a complex migration to a new replacement system of record
- Sheer effort to refactor the legacy application using a modern IT architecture

At the same time, drivers to move from the legacy platform have increasing immediacy:

- Scarcity of resources to maintain and enhance the legacy system
- Correspondingly high maintenance costs
- Inability to support business resiliency due to slow development timescales and long delivery cycles for changes

In this context, the phased migration using the strangulation pattern outlined in Section 6 offers a viable solution to the vendor lock-in problem. By allowing for a phased migration the approach this significantly de-risks the migration to a new system of record.

- New services are introduced in a controlled manner, rather than one 'big bang'
- The approach has low initial complexity, focusing on read services for new consumers
- It has the advantage that legacy application service consumers are not initially impacted by introduction of new services.

8. Summary

This Chapter has highlighted the key business and technical drivers to leverage cloud platforms in an era of open information services. Specific examples and scenarios from the financial services domain have been provided, but these are considered readily able to generalise to other business domains.

As open access to information becomes more prevalent, either though regulation or competitive necessity, there will be a need to support increased volume of transactions to access information. In these circumstances, to support scalability of the underlying information systems, it is considered vital to leverage the properties of cloud infrastructure. To do this effectively the key architecture patterns have been identified to support this business prerogative.

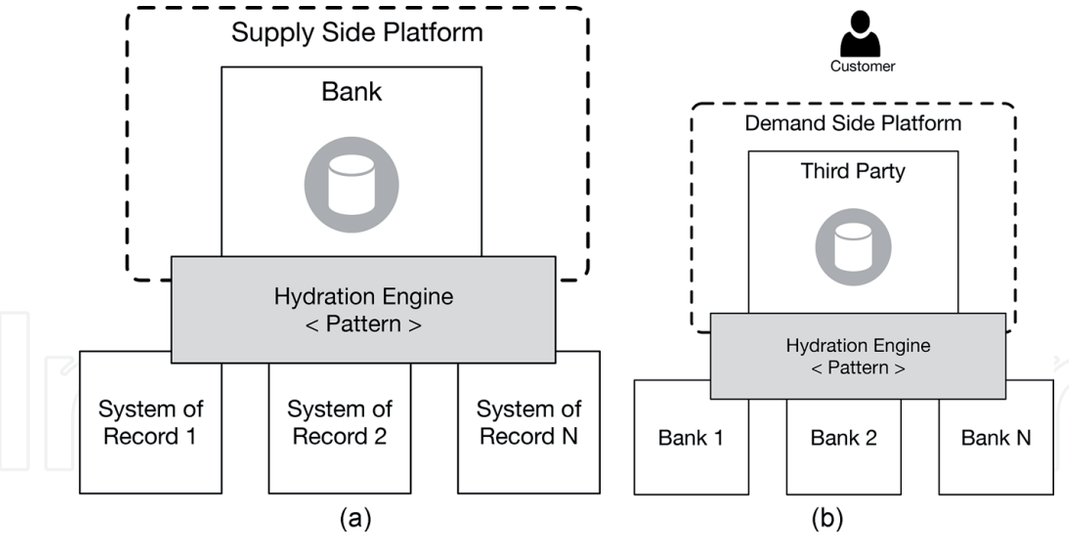

Figure 10.
(a) Supply side and (b) demand side cloud platforms.

The patterns accommodate both:

- A hybrid approach, leveraging existing infrastructure, co-existing with a cloud platform and;

- A phased, but ultimately complete, migration from a conventional infrastructure deployment to that of a cloud platform

The patterns have been presented in a cloud provider agnostic manner and there are a significant number of technology implementations that can be considered that map to the middleware capabilities that have been highlighted. This makes them highly realisable with current cloud middleware technologies and the key global cloud providers; AWS, GCP and Azure.

Glossary

API	Application Programming Interface
AWS	Amazon Web Services
B2B	Business to Business
CQRS	Command Query Response Segregation
CICS	Customer Information Control System
CRUD	Create Read Update Delete
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
JSON	Java Object Notation
MQ	Message Queue
PaaS	Platform as a Service
PEP	Policy Enforcement Point
PSD2	Payment Services Directive 2
ReST	Representational State Transfer
SaaS	Software as a Service
SOA	Service Oriented Architecture

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TSP	Technical Service Provider
XML	eXtensible Markup Language

Author details

Gary S.D. Farrow
Triari Consulting Ltd, Manchester, UK

*Address all correspondence to: gary.farrow@triari.co.uk

References

[1] Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC (Text with EEA relevance), available at: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32015L2366, last accessed on 7 Feb 2020

[2] *"Pattern-based Multi-Cloud Architecture Migration",* Pooyan Jamshidi1, Claus Pahl2, Nabor C. Mendonc , Software Practice and. Experience. 2016; 00:1-25

[3] Ernst & Young (2018) 'FinTech Open Banking Snapshot', available at: https://assets.ey.com/content/ dam/ey-sites/ey-com/en_gl/topics/banking-and-capital-markets/ey-FinTech-open-banking-snapshot. pdf Last accessed 6 June, 2021.

[4] The Berlin Group (2020) 'NextGenPSD2 XS2A Framework — Implementation Guidelines, Version 1.3.6', available at: https://www.berlin-group.org/nextgenpsd2-downloads. Last accessed 8 Jun 2021

[5] "*Open Banking: The Rise of the Cloud Platform",* G. S. D Farrow, Journal of Payments Strategy & Systems, Volume 14 Number 2, 2020.

[6] *Young, Greg. "CQRS Documents" (PDF)* http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf. *Last accessed 7 May 2021*

[7] *Fowler, Martin. "CQRS".* https://martinfowler.com/bliki/CQRS.html. *Last accessed on 7 May 2021*

[8] *Domain Driven Design: Tackling Complexity in the Heart of Software. Eric Evans, Sept 2003, ISBN-10032112515*

[9] *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley. pp. 151ff. ISBN 0-201-63361-2.*

[10] An Agile Approach to a Legacy System, Chris Stevenson and Andy Pols, http://cdn.pols.co.uk/papers/agile-approach-to-legacy-systems.pdf. Last accessed 10 May, 2021

[11] 'Competition and Marketing Authority Report - RETAIL BANKING MARKET INVESTIGATION, "The Retail Banking Market Investigation Order"', 2017.