Topic: Cloud Computing

Date: July 2011

Author: Lawrence Wilkes

Application Migration Patterns for the Service Oriented Cloud

Abstract: As well as deploying new applications to the cloud, many organizations will also be considering the opportunities to migrate current applications to the cloud in search of reduced costs or SLA improvements. In this research note we consider several migration alternatives, expressed as a set of patterns.

The patterns can also be seen as a sequence of activities, through which the current application is gradually modernized.

Public or Private?

A fundamental question will be the extent to which a pattern applies to the migration to a public or private cloud, or both.

Architecturally, there should be no difference. But from a capital or operational expenditure perspective, an organization seeking to reduce costs will not want to invest in a private cloud to just improve the SLA of applications running on a niche platform. That said, if an organization invests in a private cloud for its core platforms, but it is also one that can purpose instances of the niche platform on-demand, then that may be a viable option.

The use of a public IaaS provider will be dependent on their ability to support the platform. They may provide

- Configurable IaaS resources providing required OS, database and necessary licenses.
- Or support a "bring your own licenses" approach, when provisioning an empty server.

That is not to suggest that a private cloud doesn't face licensing issues. Issues of multi-tenancy and virtualization may not be well dealt with by the niche or legacy platform on a license basis. But that is beyond the scope of this note.

Application Re-Hosting Pattern

Organizations can start by considering whether the application is suitable for simply migrating "as is".

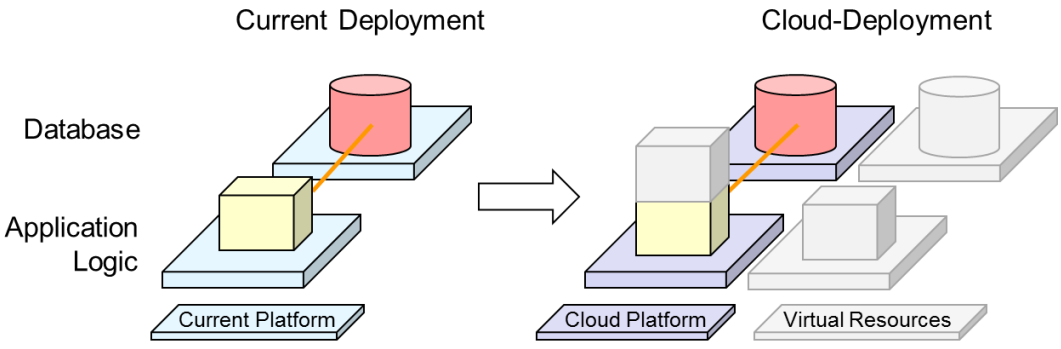

Figure 1 – Application Re-Hosting Pattern

As illustrated in figure 1, the current architecture is simply mirrored in the cloud deployment, but can take advantage of virtualization to not only reduce operational expenditure (OpEx), but also to create multiple

instances of the application to improve the SLA with scalability and failover without increasing the capital expenditure (CapEx).

The key risk is that underlying architecture issues not addressed. A monolithic legacy app in the cloud is still a monolithic legacy app. Hence scalability is on a coarse-grained basis and may not be easy to achieve if for example the internal architecture doesn't lend itself to the database being updated by multiple instances of the application.

Title	**Application Re-Hosting in Cloud**
Definition	An application is re-hosted as-is on the cloud computing infrastructure.
Problem	Current application requires re-hosting for one or more of the following reasons, • Resource constraints limit scalability • Need to improve the SLA without Capital Expenditure (CapEx) • Single point of failure • Runs on niche platform (in comparison to other in-house apps) that the organization wishes to retire to reduce Operational Expenditure (OpEx) • Investment in additional resources hard to justify for niche platform. For example investment in backup or additional resources for unpredictable scalability • General strategy to outsource hosting to cloud platform
Synonym	Re-deployment, Forklift

Solution	Re-host on Cloud Computing infrastructure.
	Make use of elastic resources, and the provision of multiple replicated instances for failover and scalability.
	Benefits:
	• Virtualization
	  ○ Improved Backup and Failover
	  ○ Coarse-grained scalability at application level.
	• Simple coarse-grained re-deployment

Risk	Underlying architecture issues not addressed. A monolithic legacy app in the cloud is still a monolithic legacy app. Existing architecture constrains portability, deployment time and cost, scalability. Integration requirements may introduce greater complexity. Direct cost savings may be limited to storage and compute costs.

A variation on this is illustrated in figure 2, where only the database component of the application is re-hosted. The general benefits of re-hosting still apply.

The key determinant for this is the separation of application logic and database components in the current application. For many client/server style applications already using a database server this should pose little problem. The most obvious scenario for this is for "thick client" applications where the application logic is deployed to multiple desktop and laptop clients, and not to an application server.

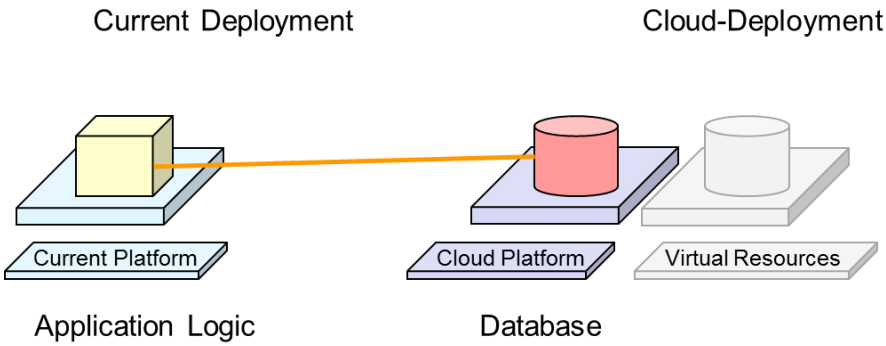

Figure 2 – Database Re-Hosting Pattern

Service Façade Pattern

Simply re-hosting the current application to the cloud does not make it service-oriented, even though the underlying cloud platform itself might be. Existing APIs do not become REST or SOAP services just because it has been re-hosted. Hence it may be difficult to integrate with other applications.

The well-known service façade pattern is not specific to application migration, but is likely to be used together with the application re-hosting pattern in order to provide REST or SOAP service interfaces that allow programmatic access to the re-hosted application. Here the pattern is given a cloud perspective.

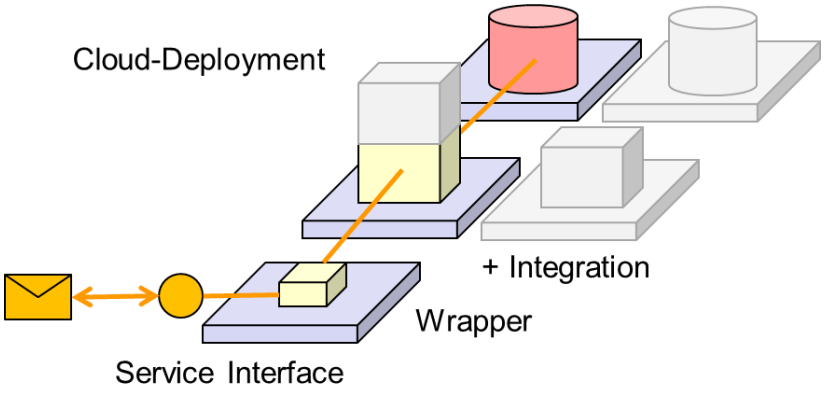

Figure 3 – Service Facade Pattern

As figure 3 shows, this requires a wrapper around the native API of the existing application that does the necessary schema and protocol conversion to provide a service interface. This helps to decouple the application from consumers (loose coupling) and provided platform independent interoperability. This pattern can also apply to database re-hosting.

Title	**Service Façade for Cloud Application**
Definition	A service façade is implemented to provide loose-coupled access to applications re-hosted on cloud computing infrastructure.
Problem	Application re-hosted as-is lacks appropriate service interfaces for integration.
Synonym	Wrapper
Solution	Build a service façade, hosted in the cloud deployment. Benefits: • Loose Coupling • Platform Independent interoperability

Title	Service Façade for Cloud Application
Risk	Lack of suitable API on existing application. Existing application process not amenable to message-based interaction. Wrapper may have to 'simulate' UI or batch interaction. Service Façade is not provided as part of a well-formed service architecture.

Re-Host and Optimize

If the goal is specifically to improve the performance aspects of the SLA, then there may be steps that can be taken to achieve that, which don't necessarily require the complete re-architecting of the application.

For example, as figure 4 illustrates, throughput may be better managed by adopting the principle of writing to a queue and reading from a cache. The queue front-ends the application and helps to smooth out peaks in transactions, whilst the cache takes the load off the application for simple reads.

This isn't going to help where the application is accessed most often via its embedded UI. However, in addition to the above, data could be replicated to a simpler table or flat file structure that optimizes reads. Or the database could be partitioned, or non-relational data segregated onto a separate instance. This would help in either UI or service interface based access.

Whilst these same actions could be taken for the current in-house deployment, leveraging the capabilities provided by IaaS and PaaS make these more viable without the associated CapEx required. A combination of the capabilities offered by Amazon AWS for example, such as the Simple Queue, Simple Storage (S3) or CloudFront may make these enhancements relatively straightforward.

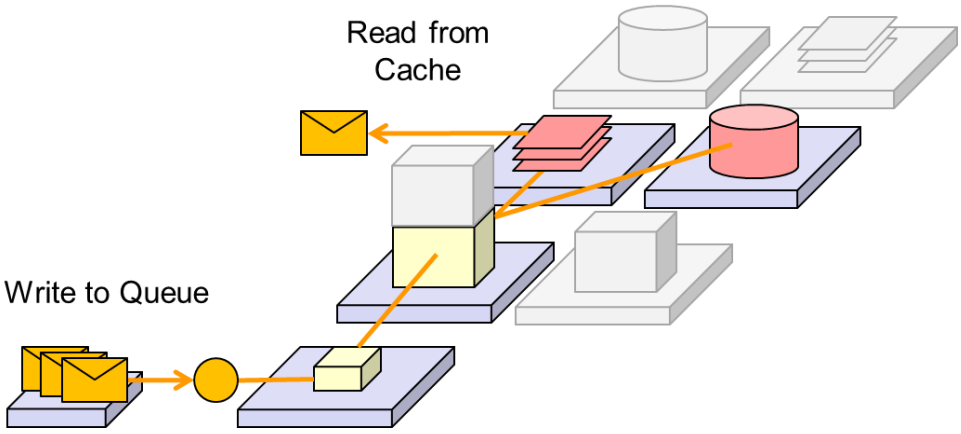

Figure 4 – Re-host and Optimize

Title	Re-host and Optimize
Definition	An application re-hosted on cloud computing infrastructure is optimized but without being re-architected.
Problem	The performance of an application needs to be improved without the significant effort of re-architecting it, and without incurring capital expenditure.
Solution	Leverage IaaS or PaaS capabilities to improve throughput, • Write messages to Queue • Partition Database • Segregate non-relational data • Cache data for fast access

Title	Re-host and Optimize
	Benefits: As Application Re-Hosting in Cloud, plus optimized performance.
Risk	Read:Write ratio changes for unplanned business reasons and proportion of read only calls reduces IaaS or PaaS provider doesn't provide the necessary capabilities to wrap the optimizations around the application without re-architecting.

Re-Architect

There is however, a limit to how much can be achieved through the optimization pattern. As stated earlier, a monolithic legacy app migrated to the cloud is still a monolithic legacy app.

Moreover, the application migration to the cloud may be under consideration as part of a broader application modernization initiative, where the goal is not just to re-host the application in the cloud but to address new business and IT requirements that demand a more agile, fine-grained architecture of services and software that is not provided by the as-is implementation.

Hence figure 5 illustrates that the application is re-architected into a set of independent services and automation units that encapsulate their own data – we will refer to these as integrity units. Each integrity unit can be independently deployed and its SLA optimized to its unique profile. The componentized implementation improves scalability – with individual automation units for each service. The deployment of high-usage components can be optimized independently of low-usage ones. Whilst a parallel design can provide better throughput.

The major risks here are that an application is modernized in isolation, and not as part of a portfolio that ensures consistent service and information architecture. Or that modernization is done primarily for technical reasons resulting in continued sub-optimal response to business change.

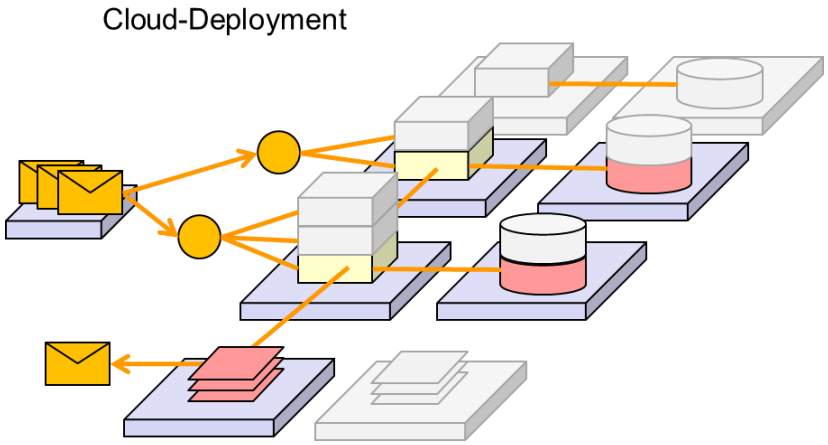

Figure 5 – Re-Architect

Title	Re-Architect
Definition	An application is re-architected for deployment on cloud computing infrastructure to provide greater agility.
Problem	Monolithic or coarse-grained applications are not agile enough to respond to changes in business and IT requirements or variations in workload, and cannot take full advantage of the SLA improvements that can be offered by cloud computing infrastructure.
Synonym	Application Modernization

Title	Re-Architect

Solution	The application is re-architected as a set of fine-grained services and automation units (implementation),
	• Componentized implementation for scalability – individual automation units for each service. Deployment of high-usage components can be optimized independently of low-usage ones.
	• Parallel Design for better throughput
	• Services and their automation units designed as independent integrity units to reduce dependencies (tight coupling) and enable replacement
	  ○ Encapsulate own Data
	  ○ Separation of sensitive data into separate integrity units
	Benefits: Fine-grained architecture enables
	• optimal scalability and performance
	• wider range of deployment options
	• agility to respond to business and IT change

Risk	Application is modernized in isolation, and not as part of a portfolio that ensures consistent service and information architecture. Modernization is done primarily for technical reasons resulting in continued sub-optimal response to business change. Service architecture is only determined bottom-up from existing APIs. Transaction and data integrity approaches may need to be re-evaluated. Cannot be left to single database in RDBMS.

Re-Architected Hybrid

An outcome of re-architecting an application is that it can utilize a hybrid cloud deployment. This is a likely scenario, where components of the application are deployed independently to both public and private clouds.

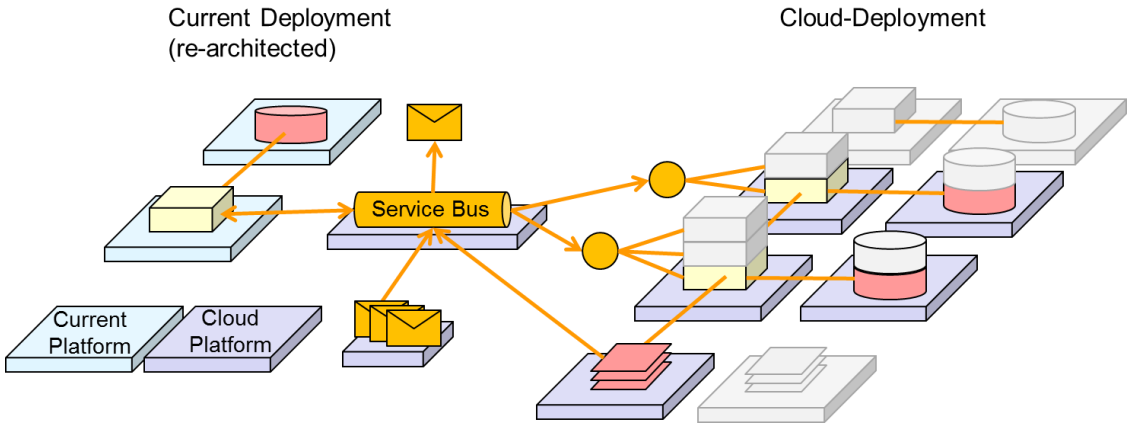

Figure 6 – Re-Architected Hybrid

A further likely scenario is illustrated in figure 6, where components of the re-architected application remain deployed on their current platform, whilst the remainder is deployed to the cloud.

Either of these scenarios might be triggered for example by a requirement to keep sensitive data in-house. Or perhaps where integration requirements demand a component remains deployed on its current platform. Or some feature of the current app cannot be replicated on the cloud platform.

If this is the case, then of course there is no option other than to re-architect the application as simply re-hosting it will not suffice.

In this scenario some form of 'service bus' is used as a mechanism to both integrate the different components of the application regardless of their deployment location, and also to further decouple the service consumers from the complexity of the deployment architecture. Moreover, it enables the provider to continue to refine the architecture without impacting the consumer. For example, there may be an orderly migration of the components from the currently platform to the cloud platform rather than a big bang approach.

The service bus might be the existing Enterprise Service Bus (ESB) that an organization already has in-house. Or it could be a capability that is part of the cloud platform, for example by Microsoft Azure's AppFabric

Title	**Re-Architected Hybrid**
Definition	A re-architected application is deployed partially on cloud computing infrastructure and partially to its current platform, or is deployed to a public/private cloud hybrid.
Problem	Not all components of the re-architected application are suitable for deployment on cloud computing infrastructure, or for deployment to a public cloud. For example due to, • sensitivity of data • lack of cloud capability to support current feature of application • license restrictions Or to support a gradual migration approach. However, as they are not co-located, some mechanism is required to integrate the components of the application.
Synonym	Heterogeneous Cloud Components
Solution	A service bus is used to provide integration of the components in different locations and to shield the service consumer from the complexity of the hybrid deployment. Benefits: • Integration is not dependent on co-location • Sensitive data remains isolated, in-house • Deployment locations for individual components can be changed over time with minimal impact on other components or service consumers.
Risk	Integrity of relationships between distributed data, and complexity of transaction integrityas a consequence of re-architecting, not just because of hybrid. However, the hybrid aspects may magnify the issue.

Portfolio Modernization and Rationalization

As mentioned in the re-architect pattern, one risk is that an application is modernized in isolation. Whilst the architecture of the new application might be greatly improved, the opportunities to improve consistency and reduce cost through consolidation and sharing across a portfolio are missed.

As figure 7 shows, Applications A and B are re-architected and migrated as a component-based portfolio offering shared services or capabilities common to both.

It is likely this will happen in stages. The business users of Application A may be unwilling to sit patiently by waiting until the migration of Application B is also completed. There are ways to mitigate this that are beyond the scope of this research note. For example, the service architecture may be developed first to act as a façade

across the current applications. New solutions can then be assembled using these services, whilst the underlying applications are re-architected and migrated behind the scenes.

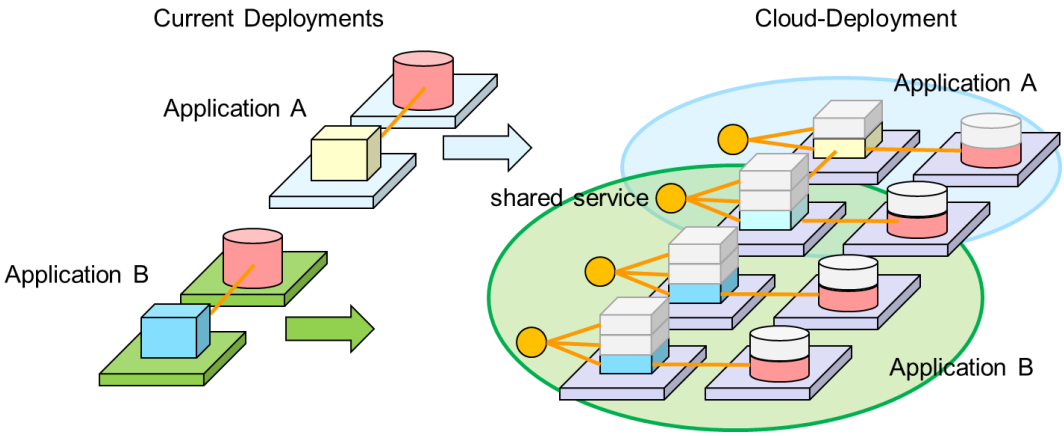

Figure 7 – Portfolio Migration

For more on this see "The Agile Application Modernization Project", and the case study "Application Modernization - Portfolio Pathfinders".

Title	**Portfolio Modernization**
Definition	Applications are re-architected as a portfolio and deployed on cloud computing infrastructure.
Problem	The re-architecting of current applications in isolation does not remove inconsistencies in data or rules between duplicated capabilities, nor reduce the cost of their combined operation or maintenance.
Synonym	Portfolio rationalization
Solution	Current applications are analysed as a portfolio to identify opportunities for rationalization, consolidation and sharing. The separation of the service architecture and the solution architecture enables the identification of services (capabilities) that are shared by more than one solution. Current applications are then re-architected and migrated to a cloud computing infrastructure as a portfolio, rather than in isolation. Benefits: • Consistent information and rules in shared services • Reduced OpEx and maintenance costs for shared services • Foundation for more agile delivery of subsequent new applications
Risk	Lack of business commitment to shared capabilities. Individual business users unwilling to wait for shared capabilities if it takes longer or costs more than a capability delivered in isolation that meets their individual needs.

Re-Provisioning

Re-architecting an application provides an opportunity to re-evaluate provisioning decisions for each capability contained in the application. The opportunity is greater when modernization is undertaken on a portfolio basis.

Analysis of business requirements should identify a set of required capabilities. Current systems analysis on the existing application then identifies which of these capabilities could be supported by the current system in its re-architected state.

However, it should not be a given that where there is a match that the existing capabilities are re-engineered. Rather, each capability should be evaluated to see if some alternative provision can be made – for example by use of a Cloud Service, or a COTS component that can be deployed to the cloud. Clearly this decision would be made on the assumption that the alternative provided some additional benefit, such as reduced cost compared to re-engineering, reduced time to solution, encapsulation of best practice, better SLA, etc.

Traditionally, organizations provision and deploy the business capabilities they require on a coarse-grained whole application basis. They purchase an ERP, an HR, or CRM application for example. However, this typically results in the tight coupling of these capabilities within these coarse-grained applications, which leads to inflexibility, and as explained earlier their deployment to the cloud cannot be fully optimized. Moreover, with a purchased application there is no opportunity for the end-user organization to re-architect.

Cloud Services presents an opportunity for the capabilities to be provisioned on a more fine-grained basis. However, it is also true that many Cloud Services are still implemented as a monolith behind the service façade which may lead to dependencies between services that requires they are provisioned on a "suite" basis.

Title	**Re-Provisioning**
Definition	Individual capabilities in a re-architected solution are re-provisioned rather than re-engineered
Problem	Existing capabilities provided by the current application are not the best alternative to meet business requirements
Solution	Analysis of business requirements should identify a set of required capabilities. The provisioning of each capability is assessed by considering • Current systems analysis on the existing application to identify which of these capabilities could be supported by the current system in its re-architected state. • Alternative provisioning sources that provide a benefit over the re-engineering of the current capability Benefits: • The solution is improved though best-in-class capabilities • Re-engineering costs and effort are saved
Risk	Cloud Service implementations are just as tightly coupled as the current application they are replacing.

Recommendations

In large organizations there may be thousands of different applications in use. Many of them are non-core applications that are quite independent, serving some specific niche business or 'utility' need. These may be obvious candidates for straightforward re-hosting.

For more integrated applications and or those considered core to the business, then re-architecting is a more likely requirement. Core business applications should best be considered as a part of a wider application portfolio modernization strategy, and not re-architected and migrated to the cloud in isolation.

In these cases, it is important that the service architecture is considered top-down to match business requirements, not just bottom-up based on the existing applications. This is necessary to better provisioning decisions.

Figure 8 shows that as suggested in the introduction, these patterns might be viewed as a sequence of activities by which an application is gradually migrated to the cloud and refined. For the reasons given throughout this research note, in many situations the initial steps of re-hosting may only be possible in a private cloud scenario. Only later once the application has been re-architected can a hybrid public/private deployment be considered.

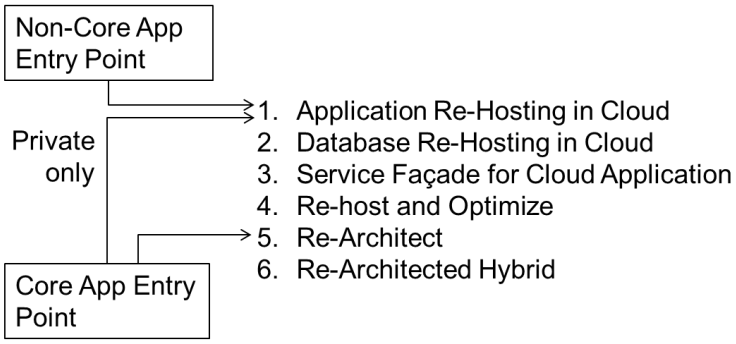

Figure 8 – Sequence of Migration Patterns

However, there is no simple rule here and each application will have to be evaluated on its own merits.

KeyWords:

Links:

CBDI Journal Report - CBDI-SAE Application Modernization Process

CBDI Journal Report extract - The Agile Application Modernization Project

Everware-CBDI Case Study - Application Modernization - Portfolio Pathfinders.

CBDI Journal Report - Service Portfolio Planning and Architecture for Cloud Services

