Decision Point for Choosing a Cloud Application Migration Strategy

ARCHIVED **Published:** 29 March 2016 **ID:** G00299768

Analyst(s): *Eric Knipp* / *Richard Watson*

Summary

Organizations are moving a growing share of application workloads to public cloud services. This Decision Point advises solution architects about how to select the most appropriate strategy for migrating an application to a public cloud provider.

ARCHIVE
This research is provided for historical perspective; portions of this document may not reflect current conditions.

More on This Topic

This is part of an in-depth collection of research. See the collection:

SERIES OVERVIEW

Modernize Application Development to Succeed as a Digital Business (https://www.gartner.com/document/code/302823?ref=grbody&refval=3266634)

Overview

Key Findings

There are five distinct strategies for migrating applications to the cloud: rehost, refactor, rearchitect, rebuild and replace. Each requires trade-offs and is appropriate for specific migration goals and priorities.

The rehost strategy is the most direct path to the cloud, but it is also the least ambitious. It offers a limited opportunity to improve the application's runtime characteristics and no opportunity to improve its underlying architecture in order to leverage cloud capabilities.

Greater benefits accrue to applications that undergo refactoring or rebuilding, which places the architecture firmly on a track toward cloud-optimized or cloud-native characteristics, but requires greater commitment on the part of the development team to master both general patterns of cloud architecture and the provider's specific support for them.

Rebuilding or replacing applications is the best way to transition immediately to a fully cloud-native architecture, but can come at a high price in terms of proprietary provider lock-in, particularly in the replace option, which shifts the strategy toward SaaS.

Recommendations

Choose a migration strategy on an application-by-application basis rather than defining a strict policy for migration across the application portfolio. While you should have clear guidance on your organization's overall principles with respect to cloud computing, you need the flexibility to make the best decision for each application.

Before you choose a migration strategy for a particular application, collect the information required. Taking shortcuts based on prior experience will cost you time and money.

Choose a rearchitect or rebuild project when you know that the lifetime of your application stretches into the foreseeable future and you have the budget to pull it off. For a legacy application, cloud computing is a modernization opportunity that must not be ignored.

View promises of SaaS portability with skepticism when you consider the replace alternative. SaaS applications will be no easier to move away from than COTS products.

Decision Point Question

How do I choose a cloud migration strategy for my application?

Cloud computing has gone mainstream. Organizations of all sizes have cloud strategies and are selecting and prioritizing applications for cloud migration. The key remaining question is: What is the best way to move them? This Decision Point makes crucial assumptions:

This is not a decision of *whether* to migrate applications to the cloud. Your organization has already made that choice. This is a "how do I do it" execution decision, not a "should I do it" strategy decision.

Your organization has already analyzed and can manage the security, availability and trust issues that accompany the move to cloud computing.

You wish to move an application that already exists in some form: either custom in-house software or commercial off-the-shelf (COTS). This is not a "greenfield" project. Several of the alternatives involve abandoning existing systems, but you already know the requirements for the applications.

This application is for production use. You're not moving development and/or testing to the cloud while maintaining production in a legacy hosting arrangement.

Your organization has already assessed the application in question, and you have determined your goals for the migration.

The decision in scope for this Decision Point is not solely an issue of migration but is truly one of optimization. Another way to state the question: Which cloud platform and migration techniques offer the chance to optimize the application's contribution to stated and implied business and IT goals?

Decision Overview

Migrating an application to the cloud requires you to consider myriad factors in order to select the best fit from five alternatives:

Rehost: In this alternative, you "lift and shift" your application from its current physical or virtual environment onto a cloud infrastructure as a service (IaaS) platform while avoiding any modifications to the system other than those required to adapt to the hosting environment itself. Although this is the least ambitious alternative, it is also the quickest to implement.

Refactor: In this alternative, you modify your application so that it can begin to take advantage of cloud capabilities for elasticity and minimized resource use. You might opt to use IaaS for the application, but at this stage, you could also choose certain platform as a service (PaaS) capabilities.

Rearchitect: In this alternative, you materially alter the application so that you can shift it to a cloud-optimized architecture, making heavy use of cloud-native capabilities. An ambitious undertaking to be sure, the rearchitect alternative is appropriate for migration projects targeting cloud platforms with a choice of IaaS and/or PaaS capabilities.

Rebuild: The rebuild alternative proposes that you start from scratch, jettisoning any old code that you've accumulated over the years. Why would you want to do this? Because new, cloud-native environments confer productivity opportunities with point-and-click styles of development. This alternative prescribes PaaS, possibly of the high-productivity variety (model-driven or rapid application development).

Replace: In this alternative, you're giving up on a custom application altogether and instead opting for commodity SaaS. While this is a common pattern for replacement of aging on-premises COTS solutions, you shouldn't immediately rule it out for your custom software applications; the low barrier to entry of the SaaS business makes it entirely possible that alternatives for your application are available off-the-shelf.

Table 1 outlines some of the key differences affecting level of effort between the alternatives. The balance of this document advises you how to make the decision.

Table 1. Which Elements of the Migrated Application Change, Need to Be Updated or Are Entirely Replaced With Each Alternative?

	Rehost	Refactor	Rearchitect	Rebuild	Replace
Programming language	No change	No change	No change	No change/new	N/A

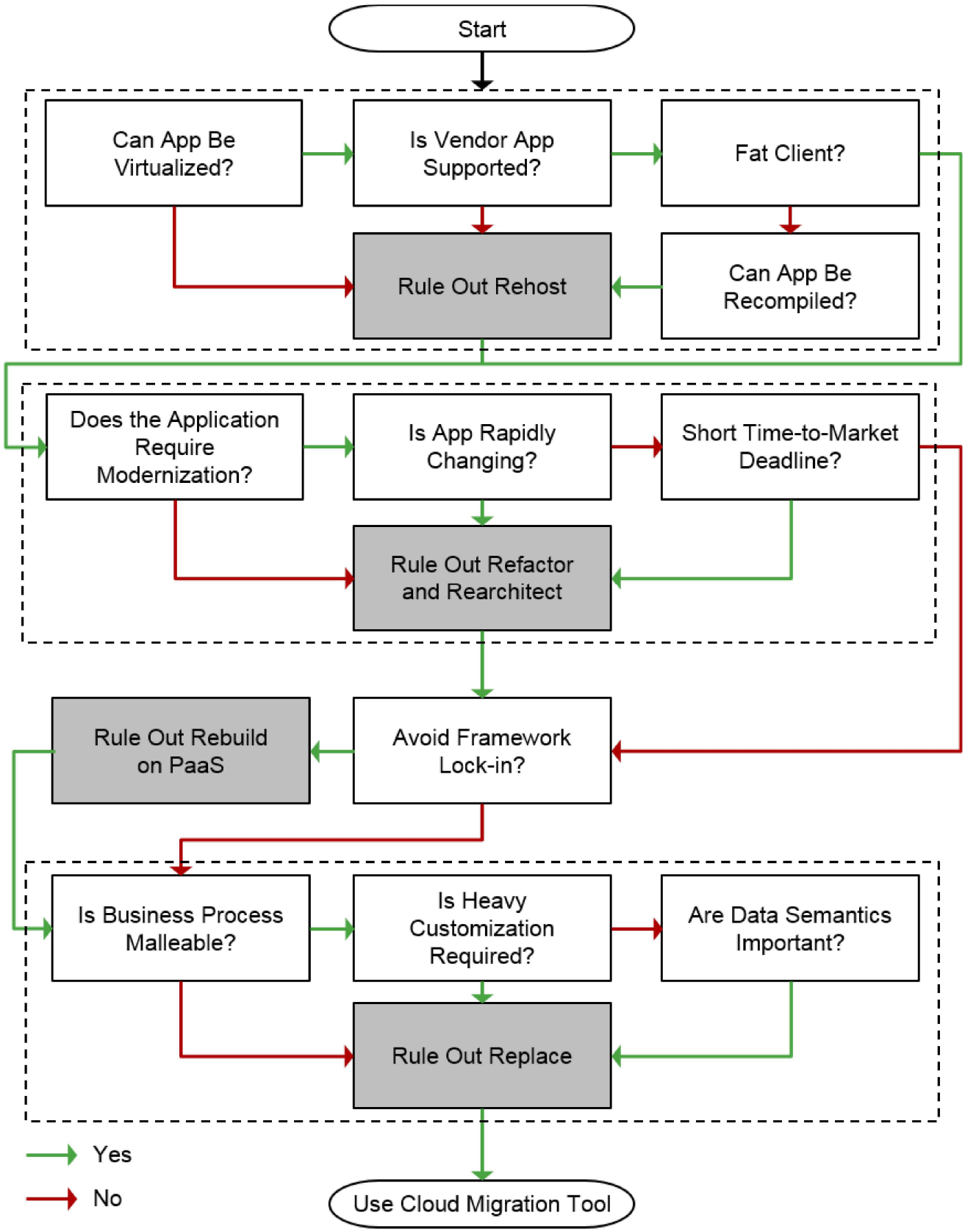
	Rehost	Refactor	Rearchitect	Rebuild	Replace
Source code	No change/ updated	Updated	Updated/new	New	N/A
Application configuration/ metadata	No change/ updated	Extended	Extended	New	N/A
Frameworks	No change	No change/new	No change/new	New	N/A
Runtime environment	No change	No change	No change/new	New	N/A
Application data	No change	No change/ transformed	No change/ transformed	Transformed	Transformed
Hosting hardware	New	New	New	New	New

Source: Gartner (March 2016)

Decision Tool

The decision tool, a spreadsheet included with this research, guides you to a recommendation for a cloud migration strategy for a given application. Different applications may require different strategies. The recommendation is driven by your understanding and selection of the migration goals, application characteristics, development and operations skills and other considerations discussed in the Requirements and Constraints section of this document. You will need to apply some of these considerations yourself to set the appropriate filters in the decision tool. Use Figure 1 and Table 2 to help you. Only attempt to use the decision tool after reading and thoroughly digesting this Decision Point.

Figure 1. Filtering the Options in the Decision Tool



Source: Gartner (March 2016)

Table 2. Explanation of Flowchart Labels

Flowchart Label	Full Question
Can App Be Virtualized?	Can the system administration team assemble a VM image with full application stack and configure hardware?
Is Vendor App Supported?	Can third-party software be run in a VM, and is it supported by the vendor?

Flowchart Label	Full Question
Fat Client?	Does the application have a "fat client" form factor?
Modernization Required?	Does the application require modernization?
Rule Out Rehost	The rehost alternative isn't suitable because of one or more criteria. Move on to other alternatives.
Can App Be Recompiled?	Can the development team recompile and repackage the application from source code?
Is App Rapidly Changing?	Are the demands for this application or its business requirements rapidly changing?
Short Time-to-Market Deadline?	Does the application have an extremely short time-to-market deadline?
Rule Out Refactor and Rearchitect	Both the refactor and rearchitect alternatives aren't suitable because of one or more criteria. Move on to other alternatives.
Avoid Framework Lock-In?	Is code or framework lock-in an unacceptable risk?
Rule Out Rebuild on PaaS	The rebuild alternative isn't suitable because of framework lock-in criteria. Move on to the final alternative.
Is Business Process Malleable?	Are business users prepared to modify their business processes?
Is Heavy Customization Required?	Does the application require a nontrivial level of process integration or customization?
Are Data Semantics Important?	Is preserving data semantics a priority?
Rule Out Replace	The replace alternative isn't suitable because of one or more criteria.
Use Cloud Migration Tool	Use the Cloud Migration Goal Assessment Tool.

Source: Gartner (March 2016)

Principles, Requirements and Constraints

When you choose a migration option, you operate under certain principles. These principles are unique to your organization. Make sure you fully understand where your organization stands because these principles directly impact the criteria you value as "most important" in the decision of where to migrate an application.

Principles

The following architectural principles impact the cloud migration execution decision:

Degree of autonomy: Because of pay-as-you-go (PAYG) and self-service characteristics, application migration to cloud providers can support a departmental autonomy strategy. This principle impacts the migration decision because some alternatives can be used more opportunistically, without central IT support.

Closed vs. open solutions: Organizations favoring solutions based on open standards have a reduced choice of cloud providers. Each alternative presents switching costs related to the service offered. Closed versus open means different things from virtualization, code, application runtime and data perspectives.

Single vendor vs. best of breed: The organization's sourcing principles may dictate the use of single vendors over best-of-breed vendors and influence the decision when multiple migration options meet an application's requirements. Incumbent platform vendors may not have offerings corresponding to all alternatives.

Platform domain specificity: Cloud platforms can be agnostic (or "generic") when they are used to build applications for any domain (for example, Microsoft Azure), or they may have a specific domain bias, or "flavor." Applications deployed on a domain-specific platform extend the domain-based features or templates and are often described as "adjacent applications." For example, Force.com applications can easily extend and leverage Salesforce CRM data and functionality. Organizations that have already made investments in a provider's services, such as Google Apps for Business or Salesforce CRM, may favor creating adjacent applications to extend their investments.

Lock-in, portability and interoperability: Portability would not be a consideration if an architect could migrate an application and its data to a cloud provider and leave it there for its lifetime. In reality, ignoring platform lock-in is risky, so architects must consider a multisourcing strategy for multiple facets of cloud portability. VM portability covers VM image format and management APIs for hardware IaaS. Code and framework portability determines how easy it is to take an application deployed on one PaaS offering and deploy it to an alternative platform (that is, another PaaS offering or an on-premises solution). Data portability measures whether an

organization can easily retrieve its data and move it to an alternative (cloud-based or in-house) application in a reasonable time, at a reasonable cost and with no loss in data semantics. For more information about designing for portability, see "A Guidance Framework for Designing Portable Cloud Applications." (https://www.gartner.com/document/code/270845?ref=grbody&refval=3266634&latest=true)

Requirements and Constraints

Your decision on how to move a given application to the cloud depends not only on the core principles upon which you manage the middleware and application portfolio in the organization, but also on the specific characteristics that apply to the individual application under your consideration. You should consider how your organization values the following migration goals and priorities, legacy application characteristics, modernization requirements, development and operations skills constraints, and cost considerations.

Migration Goals and Priorities

Your organization's objectives for cloud migration will direct the choice of an alternative. A number of often-conflicting factors influence those goals. Your cloud adoption and legacy modernization strategies will dictate relative priorities of the following goals:

Deliver rapidly to market: Organizations face deadlines related to infrastructure support end of life, regulatory compliance or seasonal business opportunities that require a rapid application migration. Teams will favor alternatives that allow migration with minimal architecture impact and without kicking off a development project.

For migration projects that include development of new functionality, reducing the volume of code needed to express a capability is another way to make development teams more productive. Cloud platforms vary in the degree they provide or support productivity features, such as customizable application templates and data models, metadata-driven engines and prebuilt components (for example, tools like data analytics, application performance monitoring and data stream processing, supplied either directly or via a community application store).

Deliver new application capabilities: Legacy modernization strategy may dictate that the application admits new functional requirements. More maintainable software drives agility so that systems can accommodate rapid, cost-effective changes to functional and nonfunctional system requirements (for example, capacity). More usable software reduces the cost and time to recoup implementation investment.

Another objective for revision work is optimizing the application architecture for cloud infrastructure. For example, if the application is not scalable on its current infrastructure, it won't scale any better in the cloud without significant revision.

Scale elastically: Organizations may require better (or more cost-effective) resources to meet variable demand, such as peak seasons or end of month. This capacity is either not available (due to capital constraints) or can only be made available in the traditional data center more slowly than customer demand requires.

Limit operating costs: Operating costs include head count (both development and operations) and infrastructure costs (including data center power and cooling). Instead of buying compute and storage upfront, cloud computing allows the organization to switch to a pay-as-you-go model, ensuring that you don't buy any capacity that you don't need. This consumption-based pricing also enables IT organizations to perform better operational budget forecasting.

Preserve capital: By shifting to an operating-expense-driven model of compute and storage consumption, the organization can preserve capital that it otherwise would have thrown into expensive data center investments (and associated depreciation). Improving your budgeting processes in this manner can enable the IT organization to better manage its priorities.

Wring out operational inefficiencies: A complex IT portfolio impedes an organization's ability to respond to new demands and business opportunities. Is the primary goal of cloud migration the ability to install, provision and secure new infrastructure quickly? Or does the enterprise desire to reduce or reassign development or operations head count assigned to this application? Note that moving applications to the cloud won't automatically fix bad processes that hamper operational efficiency.

Consolidate infrastructure or constrain expansion: For IT departments running out of power, cooling and physical space in their data centers, migrating or retiring applications frees up these precious resources for applications that must reside on-premises. This in turn enables the consolidation of remaining applications within the existing infrastructure.

Leverage existing investments: To maximize their ROI of funds already sunk into IT, organizations want to leverage existing investments in development and operations skills, experience, tooling, infrastructure and deployed applications. Portfolio management activities must evaluate the net value of these investments. For example, is the application's codebase valuable? Alternatively, is the organization willing to modify its business processes to use migration options that encapsulate prepackaged processes and capabilities?

Open multichannel access: Migrating an application to the cloud can enable wider, secure access to the application for all employees, external partners and customers, regardless of desired form factor or location. For example, better support for mobile users is a common feature of some innovative PaaS and SaaS solutions.

Compose and integrate solutions: Does the team desire to integrate the application more easily with other cloud, SaaS and Web applications?

Legacy Application Characteristics

You must catalog specific architecture characteristics of the application that constrain the possible choices. Aspects of architecture fit that influence cloud migration decisions include:

Application pattern: Architects should determine which of the following processing and interaction patterns characterize the application because cloud platforms often have a bias that makes it easier (or impossible) to host particular patterns:

- Web application (broken into the familiar three-tier or Model-View-Controller [MVC] pattern)
- Modern (sometimes called single-page) Web application
- Fat-client application
- Application provides Web services or APIs in addition to an interactive client
- System is a batch process or has regularly executed background processes

Business logic complexity: Codebases with complex or highly specialized business logic are more costly and risky to replicate and require extensive testing to ensure logical consistency.

Data parallelism: Data that exhibits high locality of reference, or that is already partitioned (for example, nonrelational, horizontally sharded key-value stores like MongoDB and Cassandra), will migrate more successfully onto horizontally scalable infrastructure.

Application licensing restrictions: Server virtualization software has matured to the point that, from a technology point of view, most applications can efficiently run inside virtual machines (VMs). But many application vendors have not matched that technical maturity with licensing and support policies offering virtualization compatibility. For the application under consideration, architects should determine whether the vendor supports third-party COTS components or dependent libraries that run on a VM.

Demand life span and volatility: An application may have an extremely limited life span, for example, if business demand is transient or the application satisfies one-off computing requirements (for example, the International Olympic Committee organized and streamed the 2014 Sochi Winter Olympic Games using Microsoft Azure [1] (#dv_1_2014_winter)). Variable and vacillating business demand can also influence whether the requirements warrant the attention of a development team, rather than prototyping several commercial software options or simply migrating the existing application as is while demand stabilizes.

Modernization Requirements

Decision makers should consider implementing the following modernization requirements when evaluating cloud migration options:

Scalability: Applications are required to cope with additional concurrent users and increased data volume or transaction frequency. Scalable applications satisfy these additional demands by incrementally adding hardware resources. An application that is already horizontally scalable is a better fit for migration onto infrastructure that offers to elastically scale resources in response to demand. Determining the degree to which the components are autonomous and stateless and finding the most critical processing bottlenecks are required prework for this decision. Do the architects need to change the application architecture to remove latency or parallelism bottlenecks? Scaling purely by throwing hardware at an inherently nonscalable application will become expensive in an elastic and PAYG environment.

Security: Enforcing consistent role-based access control and providing single sign-on for users require federation of identity provisioning and management systems between on-premises and cloud services. Web-based services increasingly require support for open authorization and distributed authentication frameworks such as OAuth (http://oauth.net/) and OpenID (http://openid.net/) (see "Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST" (https://www.gartner.com/document/code/277246?

ref=grbody&refval=3266634&latest=true)). Teams that want to federate more easily with other Web services and SaaS will favor providers that support interoperable identity management mechanisms. Additionally, business-critical applications require enforcement of policies related to auditing, data retention, data integrity, privacy and confidentiality.

Transactional integrity: Online transaction processing (OLTP) systems can have stringent requirements regarding distributed transactions and support for a two-phase commit (2PC) protocol when updating multiple data sources to ensure immediate consistency. Application semantics may also require that operations be completed in a strict order. Variable network latency, variable service provider availability and unpredictable service levels mean applications with 2PC requirements are not suitable models for cloud platforms. Architects should modify such applications to move transactional integrity semantics from protocol to application semantics. For more information on transactional integrity, see "Eventual Consistency and Its Implications: Can You Trust Your DBMS?" (https://www.gartner.com/document/code/276734?ref=grbody&refval=3266634&latest=true)

Integration requirements: How an individual application relates to others in the IT portfolio has significant impact on its migration prospects. Picking up and moving an application with multiple dependencies, point-to-point integration and complex interdependencies will be challenging, if not impossible. To leverage cloud computing beyond trivial use cases, organizations should plan to create an IT environment without boundaries. This requires an architecture spanning (or "cohabiting") internal, private and public cloud implementations. Modernization requirements may include implementing better interfaces before shifting the application to the cloud.

Migration options vary in the degree to which integration with on-premises applications is possible. For example, can the migrated application appear to be on a private subnet of the organization's network? Or, is the migrated application entirely cut off from on-premises applications except for the provider's proprietary import/export interfaces and limited APIs?

Preserving codebase value: Organizations with valuable codebases that provide competitive advantage will favor migration options that leverage this existing asset. This is of particular interest if your organization plans to capitalize on what's being called the algorithm economy. [2] (#dv_2_algorithm_economy)

Segmenting noncore (or context) capabilities: Developing new code or actively maintaining code that implements standardized or commodity processes provides little competitive advantage. Requirements for shared business services, such as HR, finance or horizontal applications (such as Web conferencing) are well-understood and riper for "buy" migration options than specialized competencies. Adopting some of the alternatives will depend on business users' willingness to modify business processes they regard as "specialized." Department-specific applications may also have limited scope, making them less risky to prototype on more innovative cloud platforms.

Development and Operations Skills Constraints

Externalizing an application by migrating it to a cloud platform implies handing over some operational control to the provider's staff. Examining skills profiles for the remaining development and operations teams will determine how much or little control is desirable to hand to the provider. Aspects to consider include:

Development team skills profile: Modernizing legacy software sometimes requires that a development team recompile or repackage the existing system reliably from the source code. Given layers of version control sediment, this is not always a straightforward task and may require specialist knowledge or technical folklore.

Taking advantage of scalable infrastructure requires that the development team has concurrent programming, workload decomposition and data partitioning experience. The development team also requires experience debugging distributed systems.

Resolving the tension between familiarity and innovation is a key to choosing between migration alternatives. Innovative teams may be amenable to learning new languages, or to relearning how to build and deploy applications using new frameworks and containers. In contrast, teams relying on the familiar may favor migration rather than optimization and move their traditional application platforms onto platforms as similar to the status quo as possible.

Operations team profile: The migration project should determine if the enterprise presently has skilled resources to efficiently install, manage and maintain the application. Another migration consideration is whether the system administration and database administrator (DBA) teams can competently create VM images and data packages that allow an application stack and its data to be migrated. Some cloud platforms offer very low-level services that are aimed at experienced system administrators.

New skills: Externalizing to a cloud environment requires IT solution teams to acquire new skills or upgrade rusty ones, such as vendor management, capacity planning, metrics and cost estimation (see "An Emerging IT Role: The Cloud Architect" (https://www.gartner.com/document/code/271822?ref=grbody&refval=3266634&latest=true)). Also falling into this category is DevOps, which focuses on improved collaboration and cooperation between the development and operations sides of the organization. DevOps requires a heavy investment in automation skills.

Migration Cost Considerations

Migrated applications have to meet adequate service levels. Breaking down SLAs into their constituent measurements and baselining them accurately are important premigration steps. Engineers tend to ignore the need to measure this data because budgeting for on-premises applications can capitalize resources to make the cost of those resources, such as CPU or storage costs that are already capitalized, disappear. With cloud deployments, all computing resources are measured, may be explicitly charged for, and cannot be hidden with accounting tricks. Before migrating an application to a cloud environment, perform iterative design optimization cycles to prove SLAs are achievable and to reduce cloud computing resource costs (see "How to Budget, Track and Reduce Public Cloud Spending" (https://www.gartner.com/document/code/272868?ref=grbody&refval=3266634&latest=true) and "Calculating and Comparing Data Center and Public Cloud IaaS Costs" (https://www.gartner.com/document/code/297328?ref=grbody&refval=3266634&latest=true)).

The following application interaction requirements will impact design (and execution cost) of a modernized or cloud-migrated application:

Low-level interaction patterns: For example, the level of "chattiness" of the application's distributed components impacts the need for API usage and data movement.

Data requirements: Architects must assess the volume of data the application generates and/or needs to store and the nature of the required storage. For example, the application may explicitly require file or block storage services and include specific performance requirements, retention policies or compliance restrictions.

User requirements: Quantitative analysis of the number of users (including concurrent users) accessing the application is important capacity- and migration-planning data.

Latency: Interactive applications are sensitive to network and processing latency for optimal usability. Latency due to distance between users and a remote service provider requires you to consider regional proximity. The goal should be to select a provider that can guarantee an acceptable response time.

Security: Architects must determine the sensitivity of a migrated application and whether it requires special handling of network, data or other types of security. Some migration options may make custom encryption impossible, while others allow for a wide variety of approaches (see "Implementing Effective IaaS Cloud Security in Amazon Web Services" (https://www.gartner.com/document/code/260748?ref=grbody&refval=3266634&latest=true)).

Data transfer: Architects must make assessments of the network bandwidth and speed required to support the ingress and egress data rates before migration. If the application requires a specific transport or network protocol for internal functions, or to provide access to clients, these form other migration design criteria.

Independent software vendor (ISV) licensing requirements: Cloud providers may use a Services Provider License Agreement (SPLA) from ISVs (for example, IBM or Microsoft). In an SPLA, the price of the software license is included in the PAYG compute instance price. However, the specific SPLA may not allow consumers to transfer internal software licenses to the cloud.

Alternatives

Service providers crowd the cloud computing market. Figure 2 illustrates the approximate place on the spectrum of options for a selection of well-known public cloud providers. Because comparing provider offerings is challenging, this Decision Point presents migration options as a choice between five alternative actions. This Decision Point leads readers to an appropriate migration strategy rather than toward a particular service provider. The alternative migration strategies considered in this Decision Point are differentiated in this section. No matter which alternative you choose, make sure you pair your choice with a clear strategy to shut off the old version of the application. Many cloud migration projects have been derailed by an inability to move the last few stragglers off of the legacy solution:

Rehost

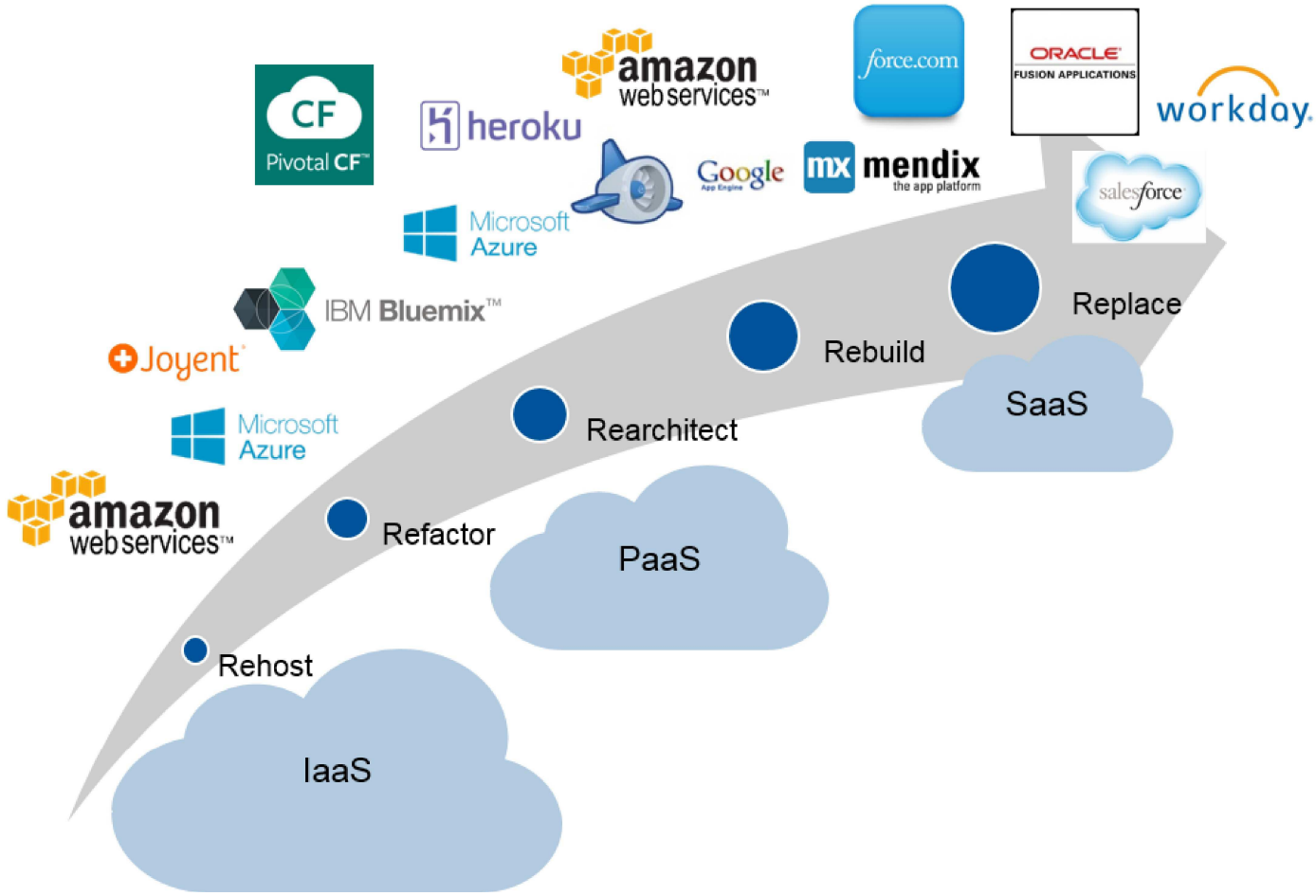
Refactor

Rearchitect

Rebuild

Replace

Figure 2. Cloud Migration Alternatives Mapped to the Three Cloud Tiers (IaaS, PaaS and SaaS)



Source: Gartner (March 2016)

The mapping between migration options and cloud tiers is not straightforward, and increasingly, the lines are blurred between IaaS and PaaS markets. This is reflected by the positioning of the providers and by the repetition of the largest ones that offer multiple tiers of service today. Providers like Microsoft that began with primarily a PaaS focus have dramatically expanded into IaaS (e.g., Compute and Azure Resource Manager) while the best-known IaaS provider, Amazon, continues to decorate its platform with a variety of PaaS capabilities (e.g., Simple Database Service and Lambda).

Rehost

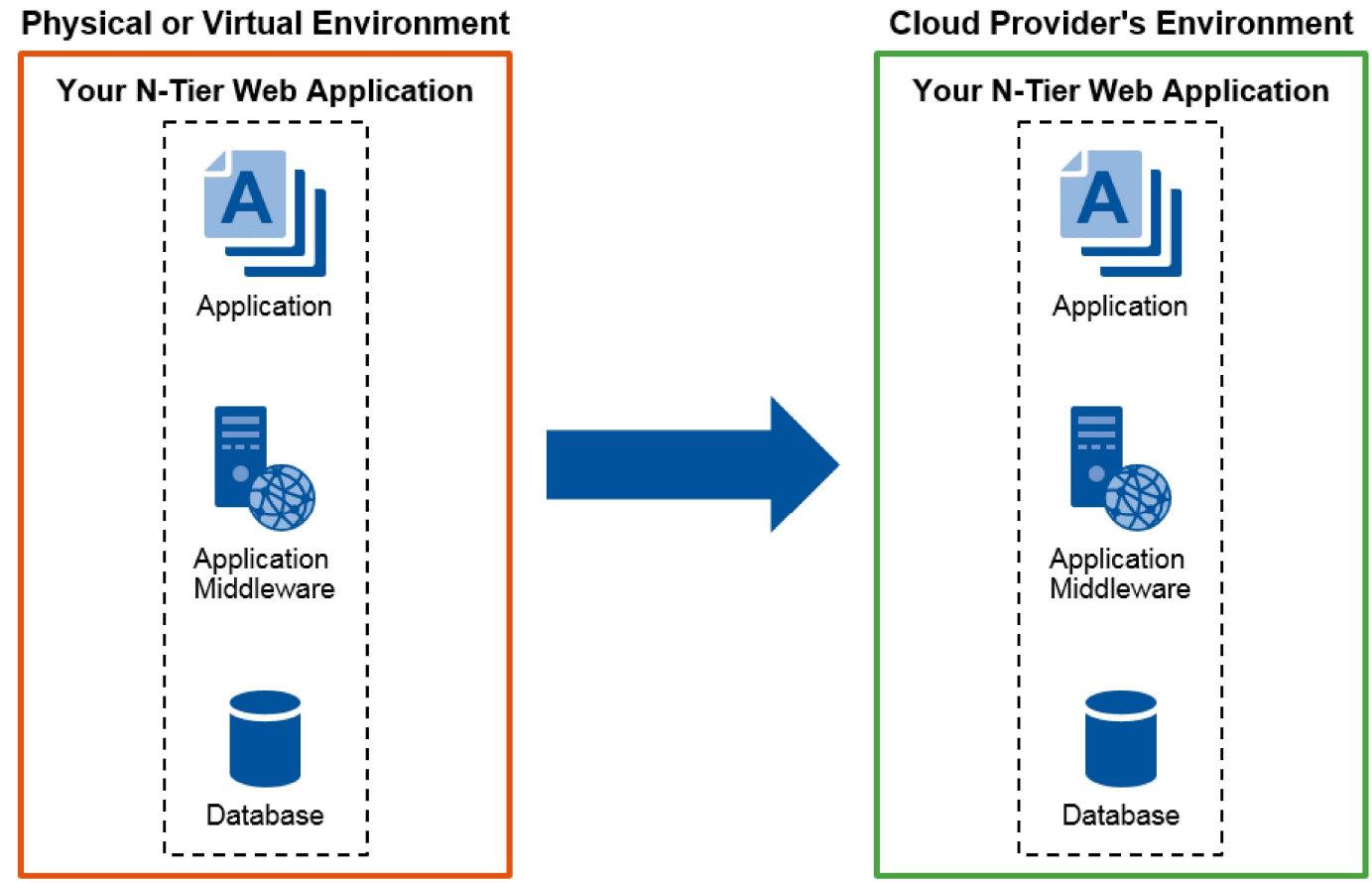
Table 3 concisely defines the rehost migration alternative, and Figure 3 depicts it graphically.

Table 3. Defining the Rehost Migration Alternative

Rehost Migration	
Definition	Redeploy the application to a different hardware environment Change the application's infrastructure configuration Similar to deploying a Java application server on a Linux x86 server instead of Solaris SPARC-based server
Cloud model tiers	IaaS
What services am I consuming?	VMs as a service/OSs as a service Block or volume storage as a service Templating service (optional)
Audience	Cloud architects, system administrators, operations staff and DevOps
Examples	Deploying an existing Java EE container with a Java EE application on EC2 Linux instances from Amazon Web Services (AWS), backed by Elastic Block Store (EBS) for persistent VM images, possibly declared using CloudFormation templates

Source: Gartner (March 2016)

Figure 3. Depiction of the Rehost Option



Source: Gartner (March 2016)

Rehost migration can work with systems where code modifications are impossible, but the application can be reconfigured to run on different hardware infrastructure. Assuming the application and its dependencies can run in a VM, this "lift and shift" strategy is possible not only for in-house-developed systems, but also for COTS and other code that cannot be rebuilt. Rehosting an application without making changes to its architecture can provide a cloud migration solution over an extremely short timeline. IaaS can provide elastic scalability if the application or data is already parallel or easily parallelizable. A stateless Web application that can be effectively load-balanced over multiple instances is a good fit. Retaining full control over the software development life cycle (SDLC) is also an advantage for teams with mature software development and operations processes.

The disadvantages of choosing the rehost option include "moving without improving" and the low-level do-it-yourself (DIY) nature of the service. The primary advantage of IaaS — that teams can migrate systems quickly without modifying their architecture — also becomes its primary disadvantage. If a team rehosts an application using IaaS without making changes to the application's architecture, the application will not benefit from the cloud characteristics of the infrastructure (for example, elastic scalability). Rehosting may afford efficiency and operations improvements in the same way as moving applications from existing servers into a new data center with server virtualization and consolidation. However, neither activity makes application architecture improvements. That said, where project goals align well (such as a mandate to quickly shut down a data center or to implement a simple disaster recovery strategy), rehosting to IaaS is an excellent option for fast time to market and a high level of organizational control.

The traditional unit of compute service offered by IaaS providers is a VM instance; today, that unit of compute is complemented with operating system (OS) container services (e.g., Amazon EC2 Container Service and Google Container Engine). Before the application's execution environment can be reproduced, the team needs to create a VM or container image configured with a complete stack with all its moving parts and dependencies (for example, application server). Providers manage image marketplaces that allow teams to pick the closest image "off the rack" or teams can build their own images. One show-stopping issue for IaaS migration is whether third-party applications or dependent libraries are licensed and supported for deployment in a VM. Early-adopter client experiences indicate that developers who deploy applications to the cloud, particularly using IaaS, find that they return to the bad old days when developers had to do their own system administration. Unless the migration strategy simultaneously brings along application development, data management, security and operations, developers can find themselves swimming in elastic Internet Protocol (IP) numbers and hacking Domain Name System (DNS) registries just to keep their applications running.

Each IaaS provider has different APIs for management and configuration, and several competing VM image and storage formats exist. Work on potential standards solutions such as Open Virtualization Format (OVF), vCloud and OpenStack is progressing. To date, the largest IaaS player, Amazon, has not chosen to participate, thus leaving its Amazon Machine Image (AMI) format and APIs, and its

CloudFormation environment declaration format, as de facto standards in the market and its ecosystem. The downside of this potential lock-in is that tools developed to automate rehosting an application on one provider will not work with another. Effort spent assembling VM images will be wasted if the team decides to switch providers. Lock-in is a concern in the cloud/no-cloud decision, but it is not a disadvantage to IaaS when compared with the other options. Any migration to the cloud involves lock-in concerns, but of the five alternatives, rehost has the least lock-in concern. "Evaluation Criteria for Cloud Infrastructure as a Service" (https://www.gartner.com/document/code/277367?ref=grbody&refval=3266634&latest=true) discusses the IaaS market in greater detail.

When to Rehost

The rehost alternative is appropriate when the primary goal is to leverage existing software investments. Rehosting on IaaS is most appropriate for the following scenarios:

- Compute-intensive applications that are built for parallelism but don't require high-performance interprocess communications (IPC) and have independent datasets, and applications for which load balancing already increases scalability and availability.
- Transient applications for which the hardware is not available or worth additional investment, and for organizations that lack capital to procure enterprise IT infrastructure (for example, startups or new lines of business).

Rehost options can be ruled out when licensing compatibility for third-party software cannot be extended to VMs or if the architecture doesn't fit (for example, fat-client architecture).

Refactor

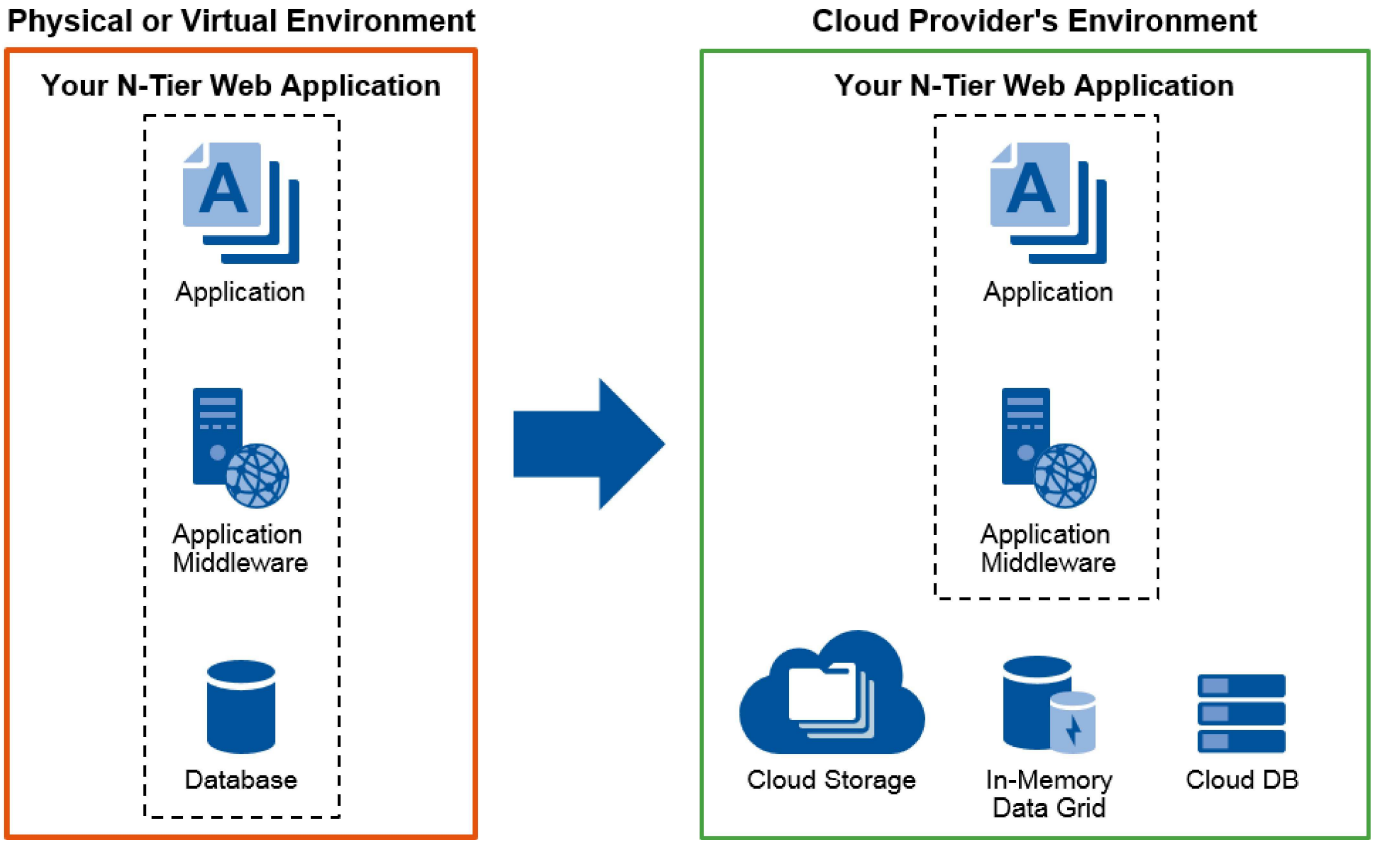
Table 4 concisely defines the refactor migration option, and Figure 4 depicts it graphically.

Table 4. Defining the Refactor Migration Alternative

Refactor Migration	
Definition	Running your applications (usually Web applications) on cloud provider's infrastructure Make application code or configuration changes to connect the application to new infrastructure services Similar to linking in a new database driver, identity management system or authentication module; also similar to moving a Java EE application from IBM WebSphere to Red Hat JBoss (same type of container, some different frameworks and configuration) Necessary changes vary from none to widespread code change to invoke new APIs Relink/recode is "backward-compatible" PaaS; existing programming models, languages and frameworks that can be used and extended
Cloud model tiers	IaaS or PaaS
What services am I consuming?	Provider-supplied cloud-based frameworks and management tools that allow developers to take advantage of cloud characteristics of provider's infrastructure
Audience	Cloud architects and developers
Examples	Deploying Ruby on Rails Web application to Heroku (including linking a monitoring service based on New Relic) or deploying an ASP.NET app to Microsoft Azure Web Apps within App Service Replacing a third-party load balancer with the cloud provider's load balancer (such as Amazon Elastic Load Balancing) For an application hosted on AWS EC2, replacing a cloud-hosted Oracle DBMS instance with AWS Relational Database Service (RDS) instead

Source: Gartner (March 2016)

Figure 4. Depiction of the Refactor Option



Source: Gartner (March 2016)

The primary advantage of the refactor strategy is blending familiarity with innovation. "Backward-compatible" PaaS means developers can reuse languages, frameworks and containers they have invested in, thus leveraging code the organization considers strategic. The team can also take advantage of providers' best practices for cloud architecture, which are baked into providers' framework abstractions. For example, some providers advocate a nonrelational model for big, distributed datasets. Each provider has its own framework abstraction for workload scaling (for example, Heroku has dynos, slugs and workers; Google App Engine for Java has automatic scaling using a Servlet abstraction; and Microsoft's Azure App Service platform uses instances).

The disadvantages of a refactor strategy include missing capabilities, transitive risk and framework lock-in. At this early stage in the PaaS market, some of the capabilities that developers depend on with existing platforms can be missing from PaaS offerings. Architects should verify that a PaaS supports the required frameworks and APIs. For example, many existing Java applications can run on Google App Engine without modification, but not all Java class libraries are supported. [3] (#dv_3_the_jre) Other PaaS offerings may not be so restrictive, such as those based on the open-source Cloud Foundry framework (e.g., IBM Bluemix or Pivotal Web Services), but applications may require refactoring in other architecture aspects to fit into their runtime environments.

Some PaaS providers depend on IaaS providers for infrastructure (for example, Heroku uses Amazon Web Services), which creates a transitive risk for the consumer. For the refactor option, consumers need to be concerned with code and framework portability. Some PaaS providers do not provide an on-premises product option in addition to hosting an application on their own infrastructure (for example, Force.com or Google App Engine), but others do (such as Red Hat OpenShift, Pivotal Cloud Foundry, or Microsoft Azure and Azure Stack). Code written for providers' specific frameworks (for example, Azure Service Bus routing or Google App Engine data store) will only work against the provider's APIs, although the code may be written in C# or Java. Lock-in is a bigger concern in refactoring than in rehosting because the development team is adapting the application to exploit cloud-enabling application infrastructure services (such as data stores). Using those features ties the application more tightly to a specific cloud provider.

When to Refactor

Refactoring a modified application onto a cloud provider's platform is optimal when leveraging an existing mature development skill set and codebase is paramount and code portability is a concern. Application fit is crucial: When the application fits a standard pattern (for example, n-tier Web application) or can be easily redesigned to work within the providers' restrictions, refactoring can be a productive and cost-effective execution environment.

Refactoring is not an appropriate option and can be ruled out when an application cannot easily be adapted to meet provider's restrictions or when skills or infrastructure to rebuild existing code are unavailable (for example, build scripts or dependencies cannot be located and fixed, or developers cannot be assigned the task).

Rearchitect

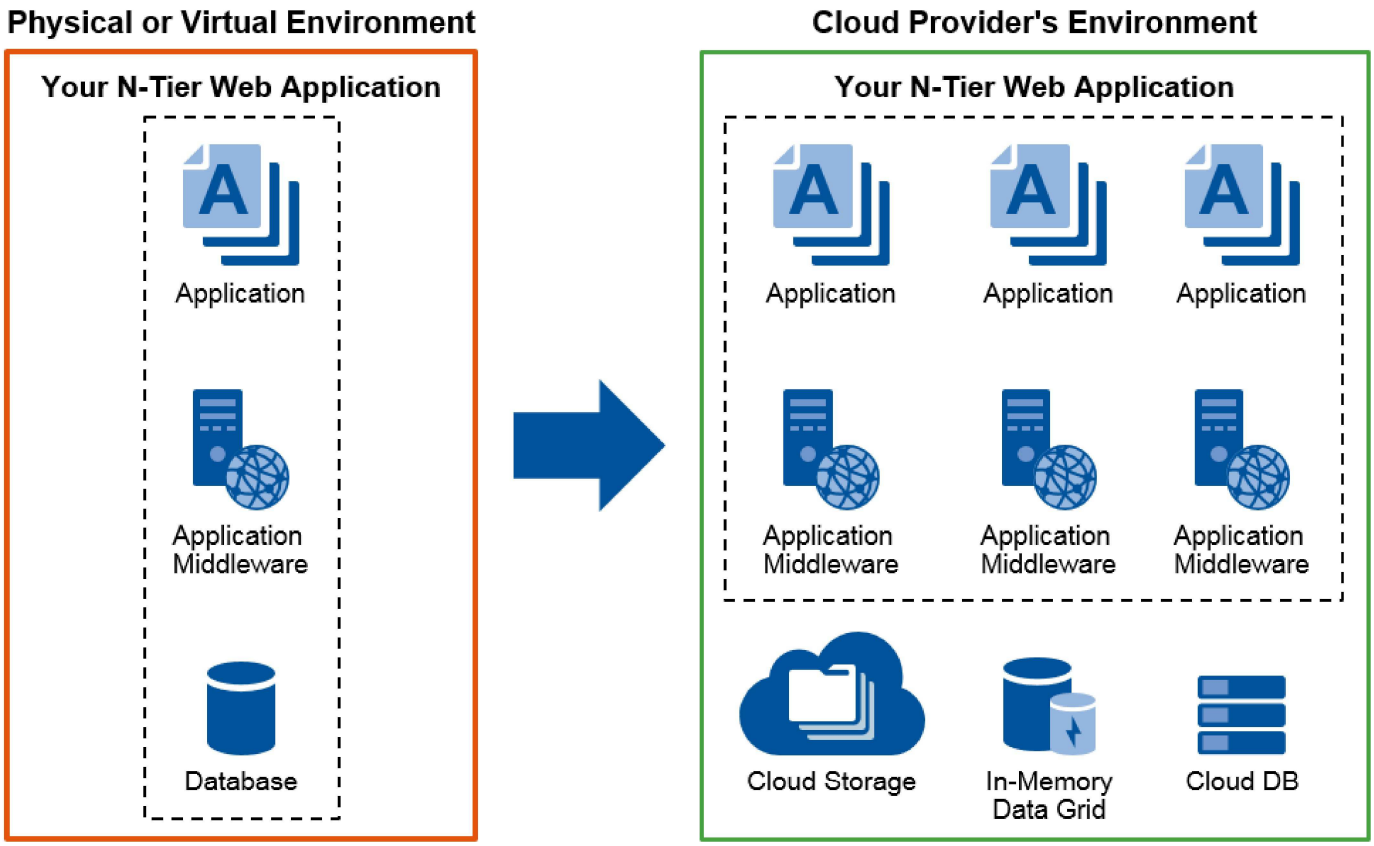
Table 5 concisely defines the rearchitect option, and Figure 5 depicts it graphically.

Table 5. Defining the Rearchitect Migration Alternative

Rearchitect Migration	
Definition	Modify or extend the existing codebase to support modernization requirements, then use rehost or refactor options to deploy to cloud Scale of changes encompasses major revisions to add new functionality or to rearchitect the application for the cloud
Cloud model tiers	IaaS or PaaS
What services am I consuming?	Either VMs as a service (IaaS) or frameworks and management tools (PaaS) that allow developers to take advantage of the cloud characteristics of a provider's infrastructure
Audience	Cloud architects and developers
Examples	Redesigning a monolithic Java application, decomposing the functions into smaller parallel chunks, and then deploying on Amazon EC2 Container Service

Source: Gartner (March 2016)

Figure 5. Depiction of the Rearchitect Option



Source: Gartner (March 2016)

Choosing the rearchitect alternative, IT organizations can take advantage of a valuable codebase by enhancing it to meet other cloud adoption or legacy modernization goals. This is the first step toward a cloud-native architecture for the migrated application. Enabling elastic scalability, dynamic reconfiguration and creative billing options most likely requires substantial rearchitecting. Compared with the other migration alternatives, rearchitect puts the organization in a position to optimize the application to leverage the cloud characteristics of providers' infrastructure.

The downside of rearchitecting an application is that kicking off a (possibly major) development project will require upfront expenses to engage a development team. Depending on the scale of the work, rearchitect is the option likely to take most time to deliver its capabilities. Of course, if the organization has already decided to undertake substantial rework of an existing application, you should be less concerned about this downside.

When to Rearchitect

Rearchitect is an appropriate option when the application needs a major revision to incorporate new capabilities or to work more effectively on a cloud platform.

Rearchitect should be ruled out if time to market is the overriding priority, project scope creep is an unacceptable risk, or the codebase is no longer valuable to the organization.

Rebuild

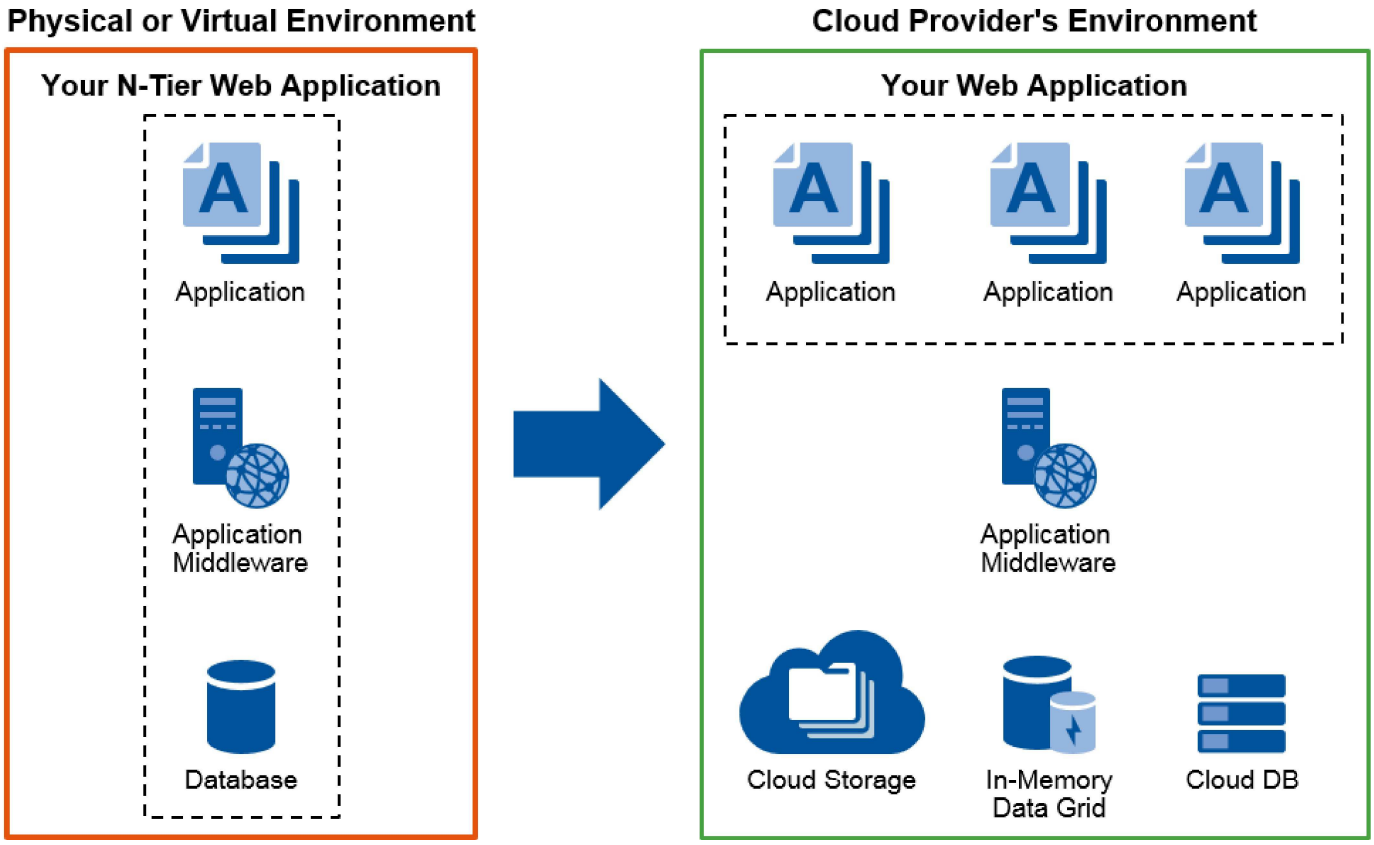
Table 6 concisely defines the rebuild migration option, and Figure 6 depicts it graphically.

Table 6. Defining the Rebuild Migration Alternative

Rebuild Migration	
Definition	Rebuild your solution on provider's application platform Discard code for an existing application Rebuild requires rearchitecting the application for a new container Tends to be "forward-compatible" or incompatible PaaS: Not all existing programming models, frameworks and languages will be retained Similar to moving from Java to .NET, but more likely to be programming for a container at a higher level of abstraction
Cloud model tiers	PaaS
What services am I consuming?	An externally managed application platform for building and running applications
Audience	Cloud architects, developers and "citizen developers"
Examples	Building a Force.com application for order management Modernizing a C and FORTRAN financial risk calculation application by redesigning it in C#, then using Windows Azure platform libraries and tools to deploy it to Microsoft's cloud Other providers include Mendix, OutSystems, Progress Rollbase and the forthcoming Azure PowerApps

Source: Gartner (March 2016)

Figure 6. Depiction of the Rebuild Option



Source: Gartner (March 2016)

Although rebuilding requires giving up the familiarity of existing code and frameworks, the advantage of rebuilding an application is access to innovative features in the provider's runtime environment. Developer productivity is improved with tools that allow application templates and data models to be customized, metadata-driven engines, and communities that supply prebuilt components. This option of metadata-driven (that is, "point and click" programming) approaches rather than code gives nonprofessional developers (that is, "citizen developers") the opportunity to develop and deploy simple applications (see "Extend IT's Reach With Citizen Developers" (https://www.gartner.com/document/code/292672?ref=grbody&refval=3266634&latest=true)). PaaS offerings allow applications, when they are written, to transparently and automatically scale to comply with the provider's framework and container model. Finally, cloud-native platforms that exploit multitenancy automatically upgrade every tenanted application, simultaneously freeing the consumer from patching, upgrading dependencies and migrating users between versions. For more information on PaaS platforms, see "Evaluation Criteria for Enterprise Application Platform as a Service." (https://www.gartner.com/document/code/277818?ref=grbody&refval=3266634&latest=true)

With innovative PaaS platforms, lock-in is the primary disadvantage. Abandoning familiar programming languages and frameworks means that second sourcing strategies to mitigate lock-in risk may not work. If the provider makes a pricing or technical change that the consumer cannot accept, breaches SLAs, or fails catastrophically (for example, Cloud Control [4] (#dv_4_cloudcontrol)), the consumer is forced to switch, potentially abandoning some or all of its application assets.

When to Rebuild

Rebuilding an application for a PaaS service is most appropriate when rapid prototyping is crucial or the scope of an application is limited (for example, limited by life span or limited to a specific audience). Another "sweet spot" for PaaS is extending previous investment in a cloud platform (for example, if the organization's customer data has already been migrated to Salesforce CRM or if the company extensively uses Google Apps as its productivity suite). This Decision Point presumes the rebuild option targets cloud-native PaaS environments that range from high-control (where standards-based programming languages and frameworks can still be used, with some limitations, like on Google App Engine) to high-productivity (which are totally proprietary, modern fourth-generation language [4GL] environments like Force.com).

Rebuilding an application for a proprietary PaaS should be ruled out when risk of code or framework lock-in is considered unacceptably high. If rapid time to market for the application is paramount, rebuilding complex functionality for a new application runtime is not a good choice. Even when you target a rebuild with standards-based languages, the degree of development required makes the project akin to starting over; you should expect to gain little, if any, reuse of existing code. In other words, if reuse is a high priority for your project, rebuilding may be a less than ideal alternative.

Replace

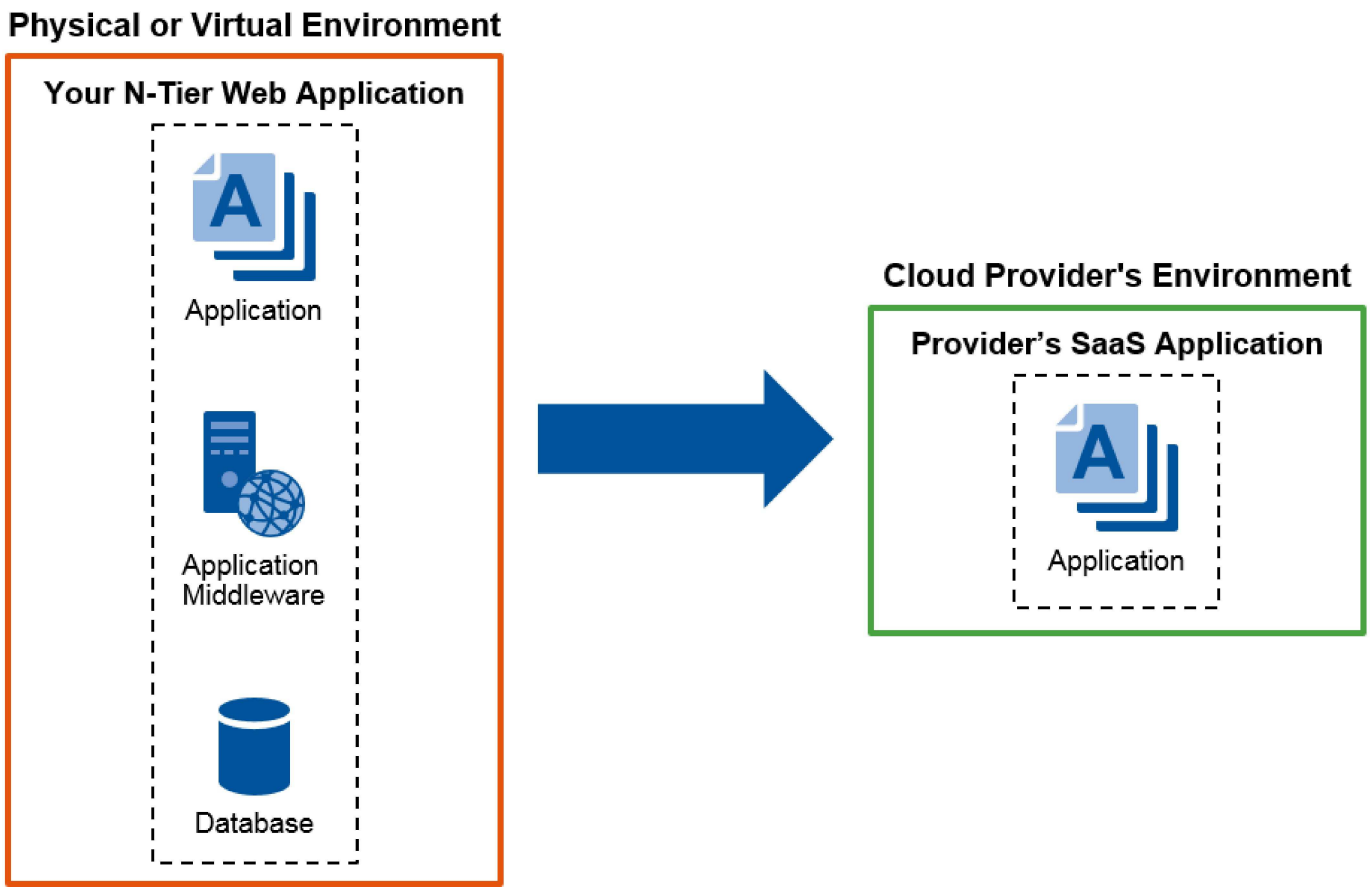
Table 7 concisely defines the replace migration option, and Figure 7 depicts it graphically.

Table 7. Defining the Replace Migration Alternative

Replace Migration	
Definition	Discard an existing application (or set of applications) and use commercial software delivered as a service (SaaS) to satisfy those business requirements Typically, existing data requires migration to the SaaS environment
Cloud model tiers	SaaS
What services am I consuming?	A complete, turnkey application solution (that is, the IT organization does not build the solution, but may configure and integrate it); users access SaaS via a user-centric interface, such as a Web browser or a mobile device; application data import/export is achieved with an API or configuration/admin console
Audience	Cloud architects and end users include business or IT; developers or DBAs perform initial configuration
Examples	Salesforce CRM and SugarCRM for customer records management Workday for HR processes Omniture for Web analytics, Live Meeting for Web conferencing Office 365 for productivity suite

Source: Gartner (March 2016)

Figure 7. Depiction of the Replace Option



Source: Gartner (March 2016)

Unlike custom-built software or single-tenant COTS, SaaS providers can monitor and leverage group behavior within a single tenant (one department or company) or across many tenants. Although the benefits for the provider are obvious, there are both advantages and disadvantages for the customer. Customers benefit from better-informed product decisions that are driven by substantial data (rather than the focus groups and lead customer strategies of old); on the flip side, customers give up any notion of privacy in how they use the

application, even if their data is encrypted and effectively invisible to the provider. The provider still knows what features you are using and when you are using them. Many of the typical advantages of buy-before-build sourcing apply to replacing applications with SaaS. Avoiding investment in mobilizing a development team is a clear advantage when requirements for a business function change quickly.

Data within SaaS applications has a tendency to become landlocked because it is difficult to interpret, access and use. The information model for the application is designed to be convenient to the vendor, and few SaaS applications include documentation on their logical data models. As a result, an additional set of data semantics must coalesce with existing meanings — of which there may be several — in an organization.

Furthermore, organizations may struggle to gain access to the data even if it they comprehend it. Will data and processes within SaaS be usable within an organization's processes, workflows and analytical environments without data transformations that hinder performance? Latency may force the use of data extracts, file transports and data imports as the only means for reasonable data access. For more information on validating the technical architecture of SaaS solutions, see "Evaluating Public Cloud SaaS Providers: Developing RFP Criteria." (https://www.gartner.com/document/code/273276?ref=grbody&refval=3266634&latest=true)

Unless a SaaS adoption plan includes a firm schedule for retiring a replaced application (or applications) and the governance oversight to see it through, new software (even delivered as a service) adds more complexity and redundancy to the application portfolio landscape.

Stretching SaaS beyond basic reconfiguration can be difficult. SaaS varies in the range of configuration and personalization possible for each tenant. A SaaS provider's ecosystem size and influence determine whether APIs and partner solutions can be used for heavier integration work. Unless the business processes replaced by the SaaS can be isolated, integration with existing systems and processes may be more difficult than with other migration options because APIs may not be available, well-built, well-documented or usable. SaaS adoption often requires a change to internal business processes. This is not a good option if the process is unique or provides a competitive advantage or if business users are not willing to adopt a different process.

When to Replace

Financial control scenarios that favor SaaS include urgent cash flow and lower operational expenses requirements. Upfront costs are likely to be lower with SaaS than alternatives, and providers often offer try-before-you-buy access, thereby providing application value more rapidly. SaaS also offers an opportunity to rapidly reassign or trim IT head count.

SaaS is also appropriate when demand for a commodity service can't be met in-house (for example, Web conferencing and Web analytics) or when the enterprise doesn't have the IT skills or resources to run complex software in-house.

Choosing SaaS is also a sensible strategy when business requirements change quickly because SaaS avoids the involvement of a development team. For example, when a company is figuring out enterprise social networking capabilities, prototyping different products can be valuable.

SaaS is best avoided when the organization is unsure about the level of integration and customization to the core feature set it will require. Enterprises sometimes overlook crucial configuration and ecosystem requirements, which limits the APIs and partner solutions used for integration work. SaaS can lose its long-term value if the organization has little or no flexibility over the data model, semantics, locations, sources or security, or when data switching costs are exorbitant. Some SaaS vendors will charge enterprises for extracting data from their services. SaaS is also ruled out if the enterprise is not willing or able to modify its business processes.

Future Developments

Risks of platform lock-in exist at every tier of the cloud. Organizations should monitor standardization efforts and vendor support to enable portability of the following aspects:

VM portability: As discussed in the Rehost section of this Decision Point, organizations such as Distributed Management Task Force (DMTF (http://dmtf.org/standards/vman)) are busily developing standards for VM image format and cloud management APIs submitted frequently by vendor coalitions. There are a variety of overlapping proposals to consider, including other open-cloud initiatives such as vCloud (http://communities.vmware.com/community/developer/forums/vcloudapi) and OpenStack (http://openstack.org/) , and standards efforts are fragmented. Without Amazon's participation, these efforts are valiant, but represent only a minority of the industry and may not have significant impact on enterprise decision making.

Container portability: The latest great hope in portability comes from a finer-grained form of virtualization based on operating-system containers. This movement, first popularized by Docker (see "Docker Democratizes Virtualization for DevOps-Minded Developers and Administrators" (https://www.gartner.com/document/code/271225?ref=grbody&refval=3266634&latest=true)), has achieved enormous traction over the last 18 months. It is now bolstered by the emergence of a standards body, the Open Container Initiative (https://www.opencontainers.org/) (OCI), which defines a standard for container life cycle, runtime and file system structure. The first production-ready implementation of the OCI standard, rkt (https://coreos.com/rkt/) 1.0, is available as of early 2016 from CoreOS.

Already supported in myriad PaaS frameworks (for example, Cloud Foundry and OpenShift) and container orchestration tools (such as Kubernetes and Apache Mesos), containers promise to be big players in the movement of workloads from private to public clouds and across multiple cloud providers.

Code portability: Deploying applications across multiple cloud environments will become a driver and a differentiator. Standards efforts presently focus on IaaS and data portability issues. In the absence of open standards, progress on code portability tends to be limited to coalitions of vendors and service providers, such as the relationships forming around Cloud Foundry and OpenShift. Portability is such a hot adoption issue for organizations that proprietary PaaS offerings with high lock-in risk will not survive. This will force some innovative pure-plays out of the market.

Data portability: The Open Data Protocol (OData (http://www.odata.org/)) pioneered by Microsoft offers a Web protocol for querying and updating data using HTTP, JavaScript Object Notation (JSON) and Atom Publishing Protocol (AtomPub).

Data access based on standards, interoperable or not, does not solve data portability issues because access does not concern itself with data semantics. For example, if an organization wants to move its application back on-premises or to another provider, the following questions are pertinent: Can it get all of its data out of the provider's data store in a reasonable time and at a reasonable cost? Will it still understand it? How many of its original relationships are preserved? Can its current provider supply useful metadata (for example, logical or physical data schema)? Now that the organization has migrated away from the provider, can it prove that its data has been deleted?

Gartner Recommended Reading

"A Guidance Framework for Designing Portable Cloud Applications" (https://www.gartner.com/document/code/270845?ref=ggrec&refval=3266634&latest=true)

"Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST" (https://www.gartner.com/document/code/277246?ref=ggrec&refval=3266634&latest=true)

"Eventual Consistency and Its Implications: Can You Trust Your DBMS?" (https://www.gartner.com/document/code/276734?ref=ggrec&refval=3266634&latest=true)

"An Emerging IT Role: The Cloud Architect" (https://www.gartner.com/document/code/271822?ref=ggrec&refval=3266634&latest=true)

"Extend IT's Reach With Citizen Developers" (https://www.gartner.com/document/code/292672?ref=ggrec&refval=3266634&latest=true)

"Evaluation Criteria for Enterprise Application Platform as a Service" (https://www.gartner.com/document/code/277818?ref=ggrec&refval=3266634&latest=true)

"Evaluating Public Cloud SaaS Providers: Developing RFP Criteria" (https://www.gartner.com/document/code/273276?ref=ggrec&refval=3266634&latest=true)

"Docker Democratizes Virtualization for DevOps-Minded Developers and Administrators" (https://www.gartner.com/document/code/271225?ref=ggrec&refval=3266634&latest=true)

Evidence

[1] "2014 Winter Olympics: Behind the Scenes With Microsoft Dynamics and Windows Azure" (http://blogs.microsoft.com/work/2014/03/06/2014-winter-olympics-behind-the-scenes-with-microsoft-dynamics-and-windows-azure/)

2 "Algorithm Economy" (http://www.gartner.com/technology/research/algorithm-economy/)

3 "The JRE Class White List" (https://cloud.google.com/appengine/docs/java/jrewhitelist)

4 "cloudControl" (https://twitter.com/cloudcontrolled/status/699530071481196544)