Comparing Integration Options for Cloud-Hosted, SaaS and On-Premises Applications

Published 21 February 2019 - ID G00379554 - 53 min read

By Analysts Kevin Matheny, Jeffery Skowron

Initiatives: Application Architecture and Platforms for Technical Professionals

As application portfolios move to the cloud, the complexity of application integration increases. Technical professionals integrating SaaS, cloud-hosted applications and on-premises applications must choose the right integration approach for each individual use case.

Overview

Key Findings

- Cloud-based applications are proliferating as organizations adopt SaaS and cloud-native development to enable digital business. At the same time, the number of options available for integrating these applications with each other and with on-premises systems is growing.
- Technical professionals face increasing pressure to deliver integrations quickly and securely, to provide visibility into the health and performance of those integrations, and to discover and manage dependencies across this radically decentralized landscape.
- Adaptability of integrations to cope with ongoing change is more important than reusability of existing solutions, despite organizational pressure to maximize the value of existing investments in an ESB or iPaaS.
- The ease of delivering point-to-point integrations via self-service platforms, APIs and SaaS provider tooling will lead to a brittle integration architecture due to the lack of consistency, central coordination and capabilities.

Recommendations

As a technical professional responsible for integrating on-premises, cloud-hosted and SaaS applications, you should:

- Avoid a "one-size-fits-all" approach. Instead, choose the right approach for each integration use case, seeking to maximize the benefit delivered by that integration in the context of your overall portfolio, reusing existing tools when appropriate.
- Use integration middleware such as an ESB or iPaaS to make it easier to monitor and manage your integrations when your integration landscape is complex.
- Track all integrations that involve either custom code or SaaS platform tooling. When you undertake such integrations in the interest of saving time, you're also taking on technical debt that will be exposed when one or more of the interfaces change.

Comparison

Cloud-based applications are proliferating as organizations adopt software as a service (SaaS) and migrate their custom-made and commercial off-the-shelf (COTS) applications to the cloud. Organizations have an increasing need for integrated business processes and real-time access to data and functionality across these radically decentralized systems.

Do not seek local optimizations for a single integration at the cost of suboptimizing your integration portfolio as a whole. But don't go too far in the other direction, emphasizing the needs of the portfolio to the point where individual integrations are suboptimized. To choose the right approach for a given integration for a given application in a given scenario, you must balance not only the considerations of that specific integration, but also the overall needs of your organization's application integration portfolio. You will face this decision each time one of the many applications in your organization needs to be integrated.

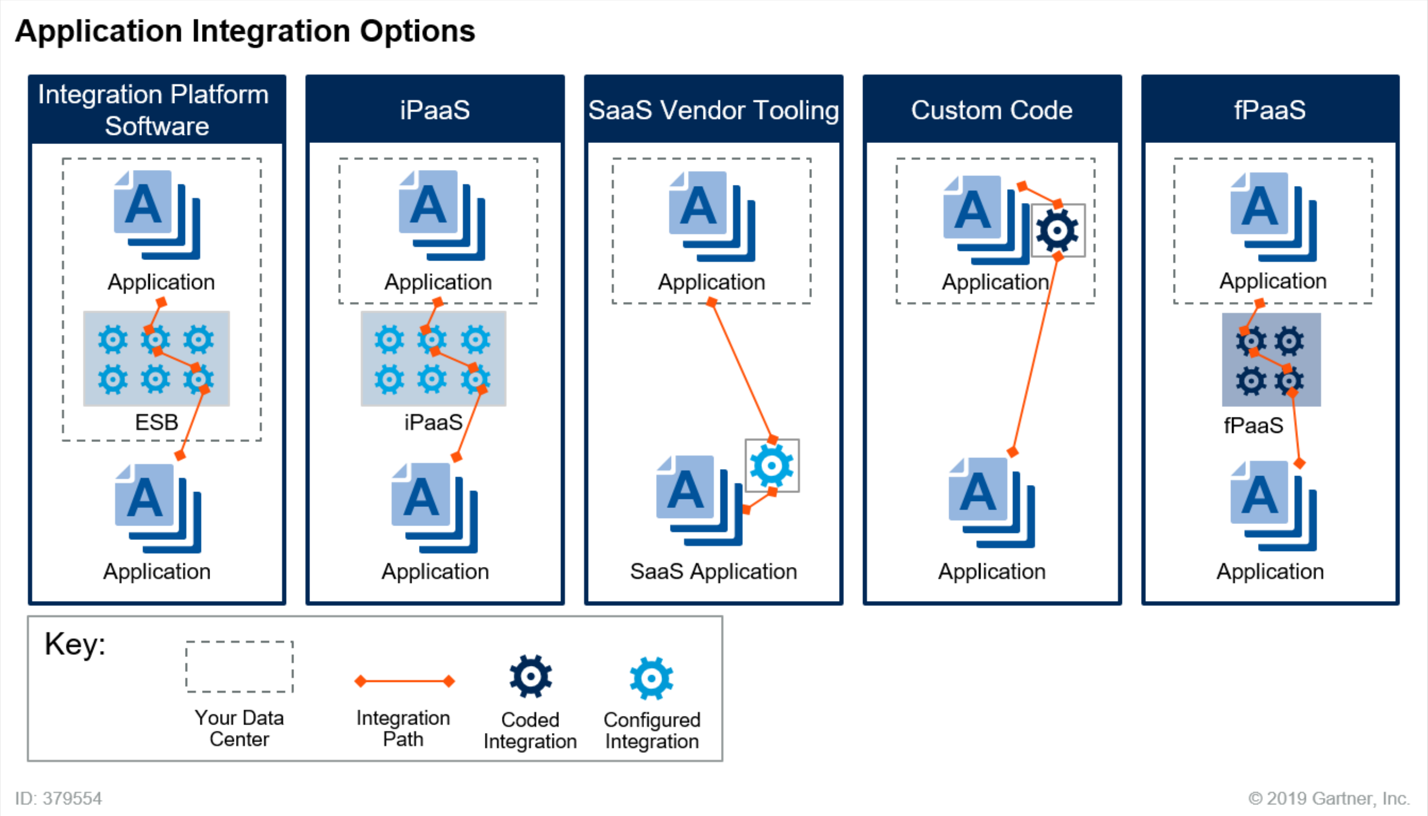
Solution architects responsible for application integration can use this research to answer the question:

> **How do I choose the right approach to create integrations between my cloud-hosted, SaaS and on-premises applications?**

Integration Connection Approaches

We have identified five common approaches to integrating cloud-hosted, SaaS and on-premises applications, shown in Figure 1. These approaches are drawn from our experience with client inquiries and represent the vast majority of integrations that we see in practice.

Figure 1. Five Options for Application Integration



Source: Gartner (February 2019)

Most of the organizations we've spoken with have multiple approaches to application integration within their portfolio. This is a reflection of the diversity of needs across the portfolio. It is unlikely that a single approach will be the best fit for the needs of all of your integrations. Additional information about each of these approaches is shown in Table 1.

Table 1: Five Approaches to Application Integration

Approach ↓	Definition ↓	Example Products ↓	Priorities ↓
Integration Platform Software	On-premises integration suites deployed as an enterprise service bus (ESB) or distributed integration platform (DIP)	MuleSoft Mule ESB, TIBCO Software BusinessWorks, Red Hat Fuse	Deep integration with on-premises applications, rich functionality

Approach ↓	Definition ↓	Example Products ↓	Priorities ↓
Integration Platform as a Service (iPaaS)	Cloud-hosted integration suites	Dell Boomi AtomSphere, SnapLogic Enterprise Integration Cloud	Cloud-to-cloud connectivity, user productivity, support for multiple styles of integration
SaaS Vendor Tooling	Integration tooling that is built into or comes with a SaaS application, or dedicated third-party tools available via a vendor application store	Salesforce Integration Cloud, Atlassian Marketplace	Low upfront cost, minimal effort to maintain
Custom Code	Custom-coded solutions specific to a single integration	N/A	Customization of the integration, avoiding the license and operating costs of integration platforms
Function PaaS (fPaaS)	Custom-coded integration sets built using fPaaS	AWS Lambda, Microsoft Azure Functions	Deep customization of integration pipelines

Source: Gartner (February 2019)

Integration Assessment Criteria

To guide you in evaluating the suitability of a given integration approach, Gartner has identified eight assessment criteria:

- **Ability to adapt —** How much of a solution can still be used if (or when) one or more of the applications being integrated need to be replaced, such as when you replace an on-premises application with a SaaS application?
- **Ability to extend —** How easy it is to make changes in the integration when required, such as adding fields to an integration between an ERP and an e-commerce application?
- **Ability to monitor —** How much visibility does this solution provide into integration uptime, performance and quality, such as integration with existing monitoring and alerting systems or built-in monitoring capabilities?

- **Ability to reuse —** How easy it is to reuse this solution for additional integrations, including reuse of artifacts such as business logic or data mapping? For example, can you reuse a product data integration between an ERP and an e-commerce system to share that data with search engines?

- **Security and data governance —** How much work is required to secure the solution, and how much control does it afford over sensitive or proprietary data? For example, does the integration functionality run on hardware you control?

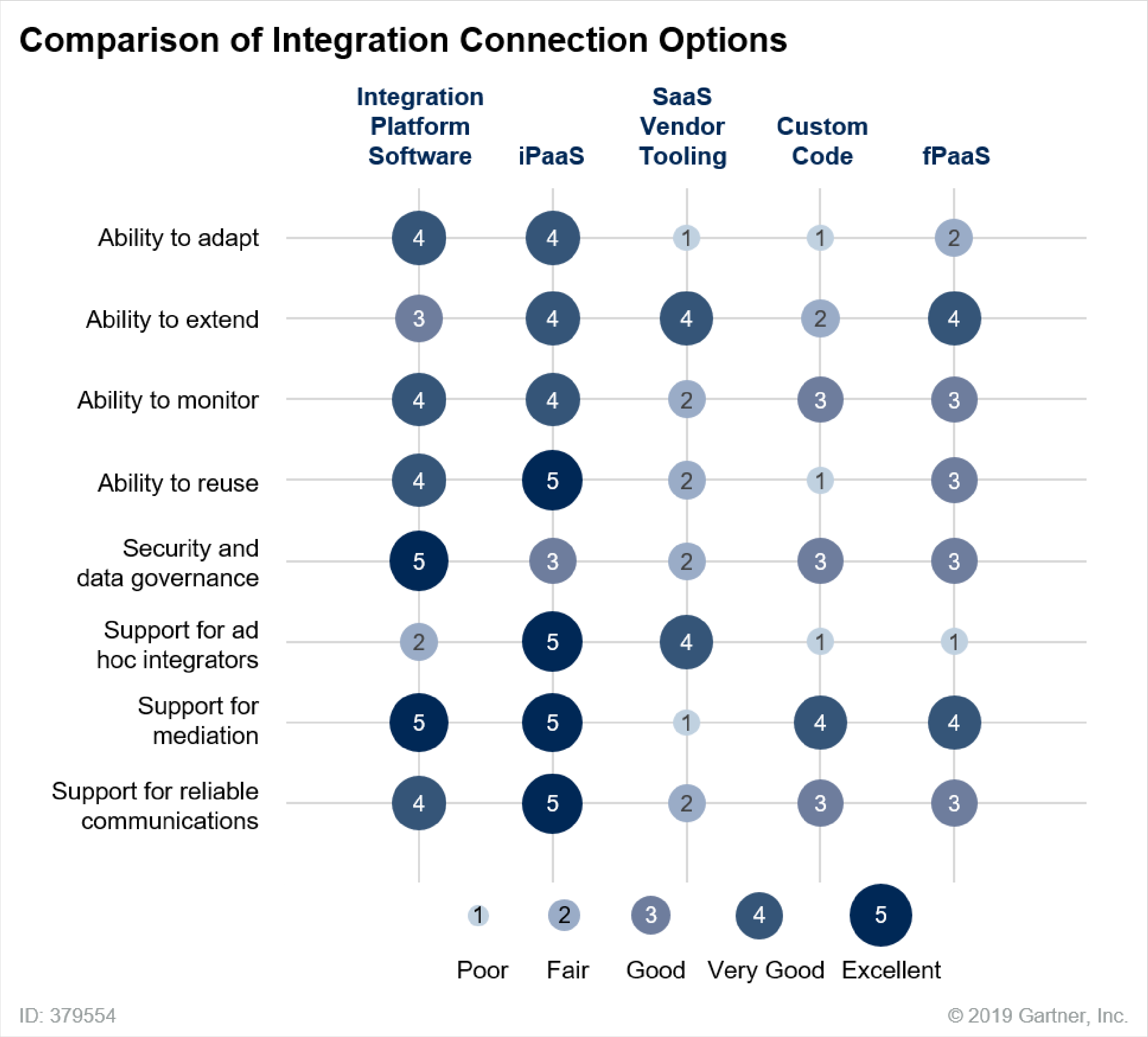
- **Support for ad hoc integrators —** How easy it is for staff members who are not integration specialists to implement or update integrations? For example, some developers may need to create integrations in order to deliver a project.

- **Support for mediation —** How well does the solution support connecting endpoints with disparate styles, such as connecting an API to a messaging system, or converting XML to JSON?

- **Support for reliable communications —** How well does the solution deal with intermittent or unreliable connections between endpoints, such as with equipment in the field or mobile devices?

Figure 2 summarizes Gartner's assessment of each integration connection type against the criteria. The Analysis section describes these connection types and criteria in more detail and provides an in-depth assessment of each connection type against the criteria.

Figure 2. Comparison of Integration Connection Options



Comparison of Integration Connection Options

	Integration Platform Software	iPaaS	SaaS Vendor Tooling	Custom Code	fPaaS
Ability to adapt	4	4	1	1	2
Ability to extend	3	4	4	2	4
Ability to monitor	4	4	2	3	3
Ability to reuse	4	5	2	1	3
Security and data governance	5	3	2	3	3
Support for ad hoc integrators	2	5	4	1	1
Support for mediation	5	5	1	4	4
Support for reliable communications	4	5	2	3	3

1	2	3	4	5
Poor	Fair	Good	Very Good	Excellent

Source: Gartner (February 2019)

Use Cases for Application Integration

Each organization and integration is unique. To determine the weighting to apply to each of the assessment criteria, architects must understand the logical flow of information and the underlying business purpose that drives the need for the integration. Gartner has identified five primary use cases, as shown in Figure 3:

- **One-Time Move** is a movement of data that happens only once for any given instance. For example, it may be used to set up a system or transfer locations. This usually involves significant amounts of data and may be coupled with an ongoing integration to keep systems in sync. Note that a "one time" move may be reused for a new instance, such as a retail store using a process to set up a local copy of item prices on an in-store server each time a new store opens.

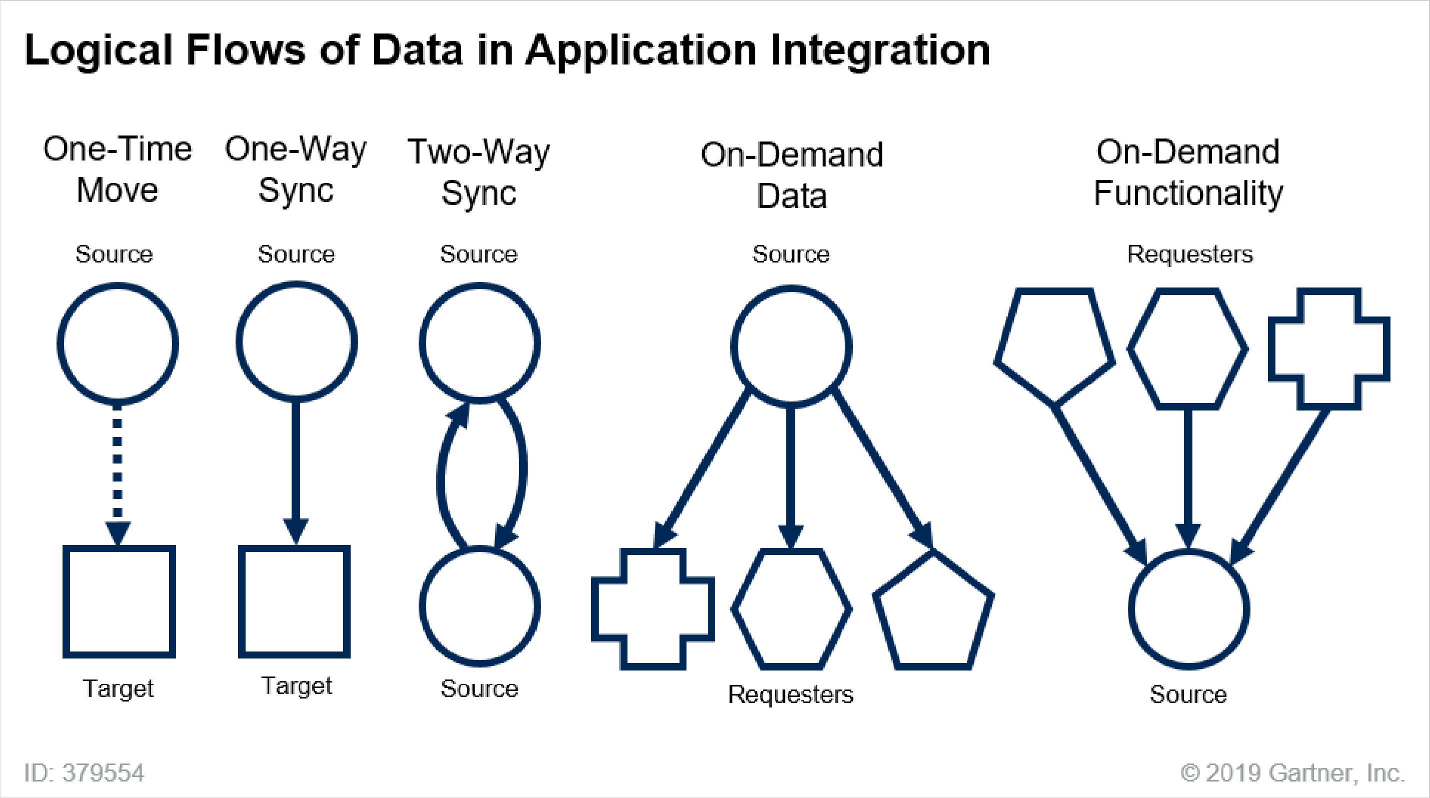
- **One-Way Synchronization** is one-way integration that is used to keep two systems in sync on a given data topic, such as prices for items in an online store. One-way sync may be scheduled, triggered or invoked on demand and can be implemented as either a push or a pull.

- **Two-Way Synchronization** is two-way integration that is used to keep two systems in sync on a given data topic. This is used when the data topic may be modified on either end of the integration, such as when both customers and staff members are updating open service tickets. Two-way sync may be scheduled, triggered or invoked on demand and can be implemented as either a push or a pull.

- **On-Demand Data** is used to make data, such as product inventory levels, available to calling systems as needed. This is invoked from the calling system.

- **On-Demand Functionality** is used to make functionality, such as customer account creation, available to calling systems as needed. This is invoked from the calling system.

Figure 3. Logical Flow Patterns for Application Integration



Source: Gartner (February 2019)

Additional information about the logical flow of data in each of these use cases is included in The Details section at the end of this assessment.

Analysis

Application integration is not a new problem, but the job of the integration architect has become significantly more complex as application portfolios expand to include cloud-hosted and SaaS applications. In addition, the increase in the number of ways that integrations can be implemented has made the challenge of finding the right approach harder, not easier.

The ecosystem of SaaS applications contains many solutions for creating direct integrations between applications, tempting integration architects to choose expediency over long-term viability. Technical professionals in organizations with investments in integration tooling find that they are under pressure to use this tooling in as many cases as possible to maximize the value of that investment. This is the sunk cost fallacy in action.

> **Do not standardize on a single integration approach or toolset in an attempt to maximize the value of existing investments; you will decrease the effectiveness of your integration portfolio as a whole.**

Instead of trying to limit complexity by standardizing on a single tool, focus on maximizing the value of each integration. This will mean that some integrations need tools you don't already have in order to be fully effective. The guidance in this assessment assists you in determining which technologies you should use in these cases. For additional information on choosing application integration platform technology, see "A Guidance Framework for Evaluating Application Integration Platforms." (https://www.gartner.com/document/code/370439?ref=authbody&refval=3902300)

Five broad considerations when you evaluate integration technologies:

- Making technology choices is about making trade-offs.
- You cannot buy a one-size-fits-all solution.
- Complexity has a cost.
- Point-to-point integrations are easy to build, but scale poorly.
- You can't add security in later.

The following subsections expand on each of these areas.

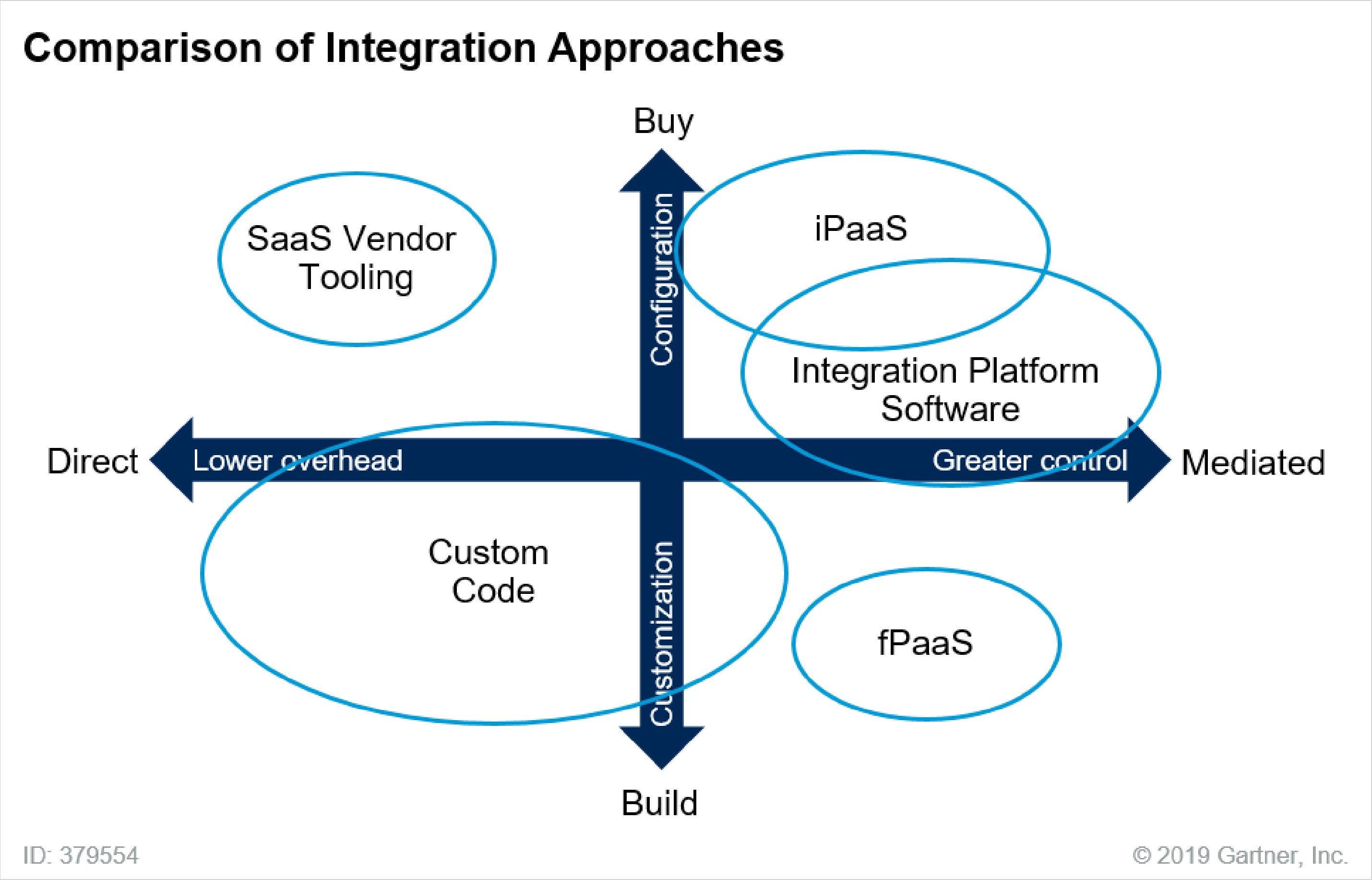
Choosing Application Integration Technology Means Making Trade-Offs

These options have different strengths and weaknesses. At a high level, they represent a trade-off on four axes:

- **Buy vs. build —** Is the integration technology developed in-house or sourced externally?
- **Direct integration vs. mediated integration —** Is the integration point-to-point, or is there some form of independent middleware?
- **Configuration vs. customization —** Can the integration be accomplished with “off-the-shelf” components configured for use? How unique or niche are the applications being integrated or the integration itself?
- **Lower overhead vs. greater control —** What is the business value of the integration? How much visibility into integration performance and reliability is required?

Figure 4 shows the relative positions of the integration options on these axes. Although SaaS vendor integration tooling may be available at no incremental cost from the SaaS vendor, Gartner classifies it as a “buy” option because the organization is sourcing a complete solution from an external party. For similar reasons, although custom-coded solutions may leverage open-source frameworks with minimal in-house development, Gartner considers this to be a “build” approach because the organization is taking responsibility for delivering, operating and maintaining integration functionality.

Figure 4. Comparing the Qualities of Five Integration Approaches



Source: Gartner (February 2019)

A Hybrid Integration Platform Is Not a Product

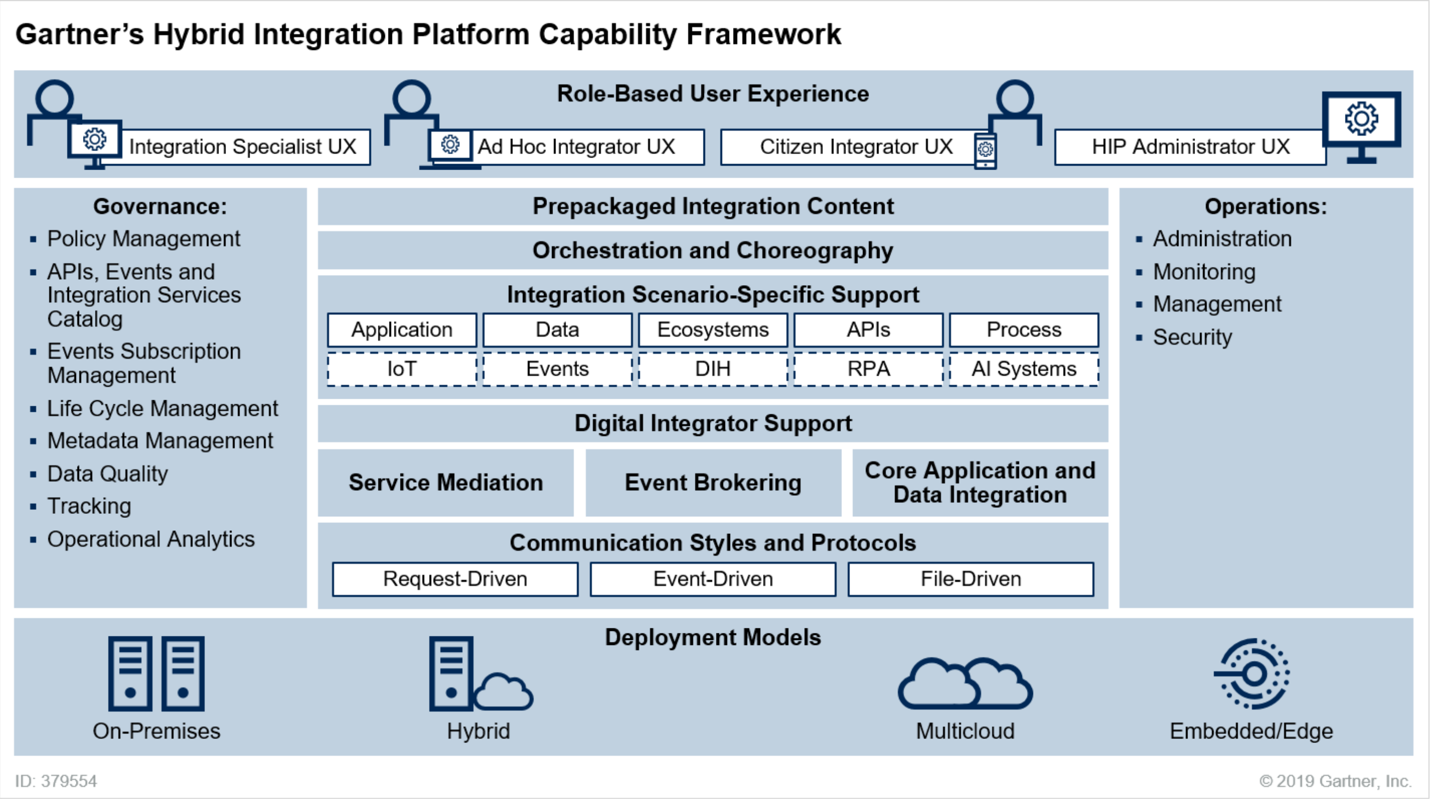
No single integration product or tool will work for all use cases. In "How to Implement a Hybrid Integration Platform to Tackle Pervasive Integration," (https://www.gartner.com/document/code/300867?ref=authbody&refval=3902300) Gartner presents the Hybrid Integration Platform (HIP) Capability Framework. A HIP is a way for organizations to create an integration platform aligned to business need instead of the traditional technology-focused approaches of the past. Your HIP will be composed of multiple technology capabilities, each selected because it provides your organization with the ability to meet a particular integration requirement.

As shown in Figure 5, a complete HIP must cover all of your organization's integration needs, including:

- Ad hoc or citizen integration personas
- Service- and message-based application integration
- Data-centric integrations
- Mobile application integration

- B2B integration
- Managed file transfers
- API publication and management
- Integration orchestration and composition

Figure 5. The Hybrid Integration Platform



Source: Gartner (February 2019)

You should think of your HIP as an integration toolbox containing integration tools that you can choose from to implement any given integration. Establishing the components that will populate your HIP toolbox is based on the golden rule that "one size does not fit all."

> **Resist the urge to single-source your integration tooling, and look with skepticism on the claims of integration vendors that their offerings will meet all your needs.**

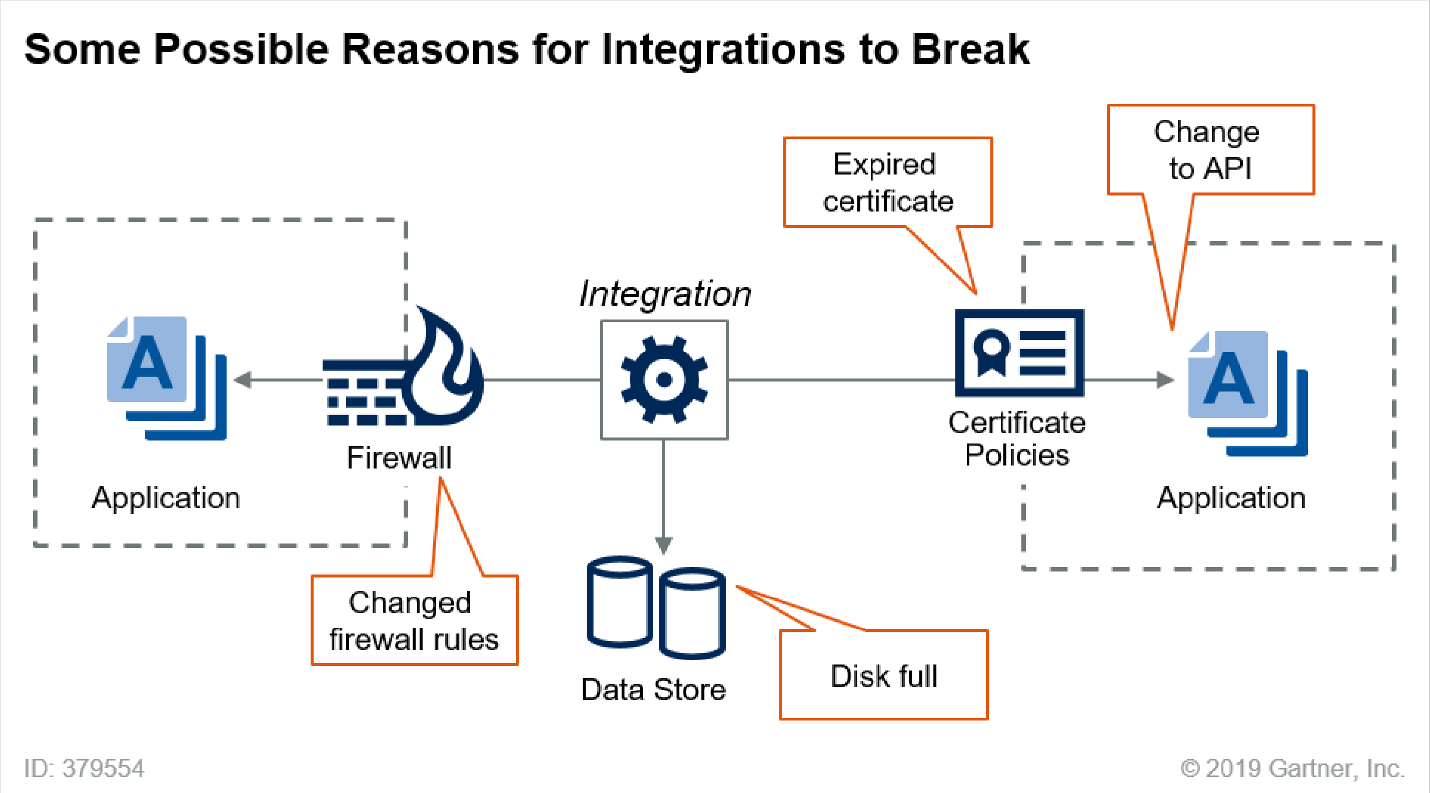
For additional guidance on choosing integration tools to create your HIP, see "Choosing Application Integration Technology." (https://www.gartner.com/document/code/314871?ref=authbody&refval=3902300)

The More Integrations You Create, the More You Must Monitor and Manage

Integrations are active connections between two or more applications. Any change to those applications can result in the need to change the integration, and changes to applications may come from any direction. SaaS vendors update functionality without warning, COTS vendors issue patches or hotfixes, and internal development teams release business-driven changes to systems.

As shown in Figure 6, integrations are also vulnerable to changes or problems with the infrastructure that connects applications. For example, an integration may stop functioning because it is unable to write log files to disk, or because an update to firewall rules prevents it from obtaining a connection to an application. If the integration itself is running as a stand-alone application or on an integration platform, the infrastructure of the integration itself may be a source of issues.

Figure 6. Potential Sources of Integration Issues



Source: Gartner (February 2019)

The greater the number of integrations in your portfolio, and the more different integration options you are using, the higher the likelihood that something will go wrong with one of your integrations. You can guard against this by:

- Designing integrations for observability and including integrations in your organization's monitoring and alerting processes
- Democratizing integration, placing responsibility for managing integrations on the product teams that need them rather than on a central integration team
- Designing integrations for security, creating layered approaches to authentication and APIs that expose discrete sets of data or functionality to aid in access control
- Using an iPaaS where appropriate and relying on the platform's provider to ensure that it is working, removing the integration platform itself from the set of things you need to monitor
- Leveraging common platforms and shared patterns for integrations (where doing so does not suboptimize the integrations) to reduce the variance between solutions
- Using standard patterns and code libraries, including open-source tools, to reduce the variance between integrations
- Documenting integration decisions using architecture decision records [1] to aid in issue diagnosis and troubleshooting

You must also determine the degree of functionality that you require from the middleware you use for each integration. Simple connectors that pass through requests transparently, or that do minimal transformations and mapping, are easier to maintain.

> **The key trade-off is between the degree of functionality provided by the integration and the effort required to maintain it.**

You should prefer solutions that are already part of your integration portfolio in order to avoid needlessly increasing the complexity of your integration portfolio. However, do not let this limit your options. The right solution for an organization is a mix of integration technologies that allows each integration to be implemented in an optimal fashion. You should have an exception process to evaluate when a new tool would be beneficial for a new use case.

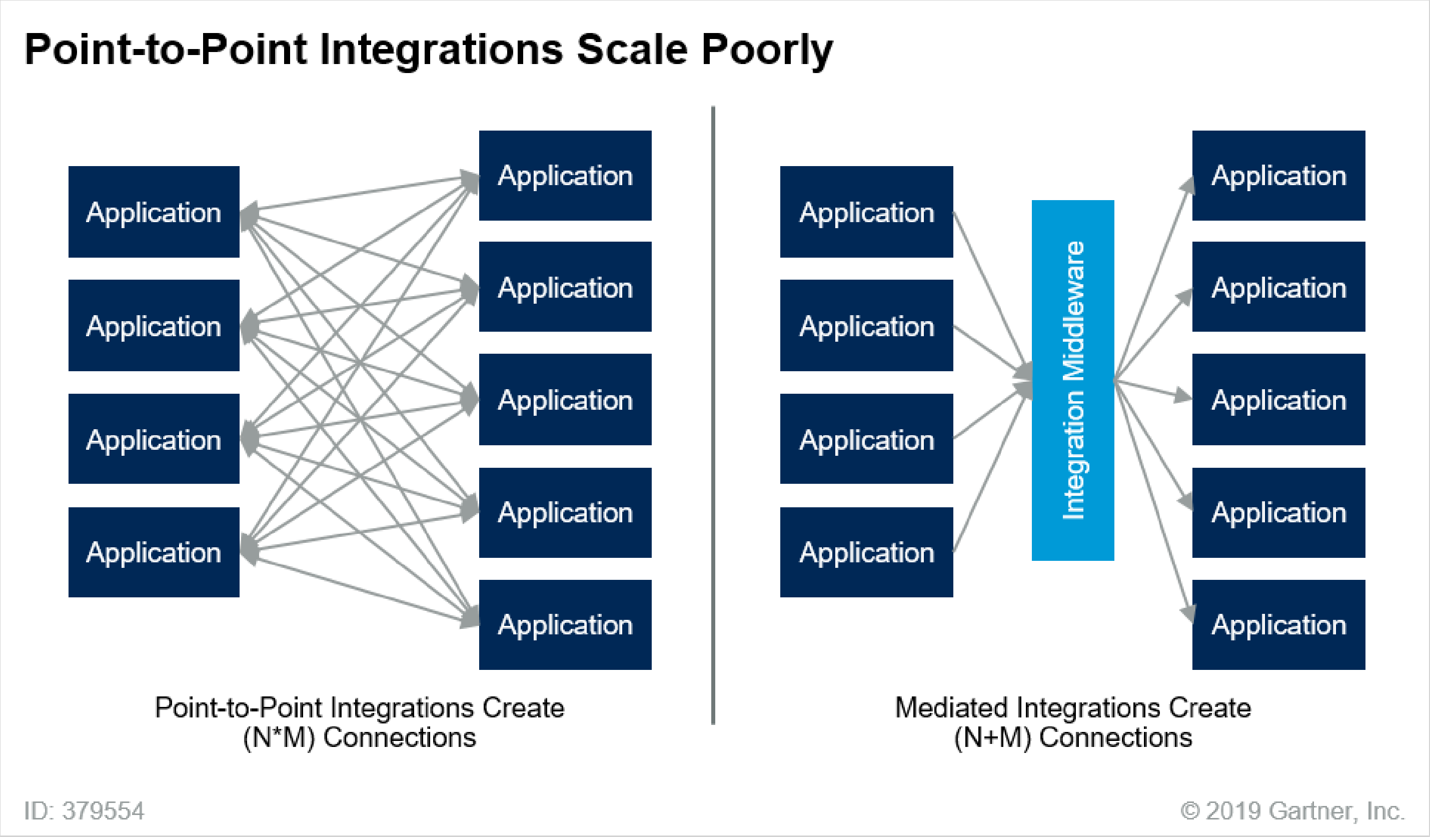
Use Point-to-Point Integrations With Caution

In "Where Does Point-to-Point Integration Belong in Your Integration Strategy?" (https://www.gartner.com/document/code/343143?ref=authbody&refval=3902300) Gartner defines point-to-point integration as a tightly coupled integration between two or more endpoints.

For example, an integration that links Salesforce and SAP S/4HANA with a single transform of data is a point-to-point integration. However, an integration between Salesforce and SAP ERP via an abstract reusable API or persisted data store is not point-to-point, because the integration process has been split into two. The mediation in the middle of the integration enables it to be reused.

Point-to-point integrations are the simplest form of integration, but that simplicity comes at a price. Because they are tightly coupled to the applications for which they are created, they cannot be reused, and each individual integration increases the complexity of your overall integration portfolio in a linear (or worse) fashion, as shown in Figure 7. In addition, point-to-point integrations can be challenging to monitor, and it is easy for these one-off solutions to become lost.

Figure 7. Scaling Problems With Point-to-Point Integrations



Source: Gartner (February 2019)

The key to successful use of point-to-point integrations is to limit their use to cases where:

- The number of uses for the specific data or functionality being integrated is small.
- The applications being connected are likely to remain stable.
- The team responsible for creating the integration is willing to commit to documenting, monitoring and updating the integration as needed.

When you choose to implement a point-to-point integration, you must be vigilant in monitoring and governing the integration. SaaS solutions are updated on a frequent basis by their providers, and interfaces or business processes that initially seemed stable will change. Be prepared to refactor your point-to-point integrations into a mediated architecture. Define triggers that will tell you when it is time to make such a change. Potential triggers for moving to a mediated solution include:

- The data model or interface of any of the integrated applications changes significantly, requiring rework of the integration.
- The data or functionality being exposed is needed by one or more additional consumers.
- The importance of the integration to the business increases, and with it, so does the need for monitoring of integration reliability and performance.

Make Security a First-Class Consideration

Connecting applications and synchronizing data between them has multiple security implications. At a minimum, the applications must authenticate to each other. However, authentication of a request is just the tip of the security iceberg. One of the unfortunate side effects of connected applications is that they are at risk of losing access granularity. Although any given application may have robust access control with role-based permissions at the object level, carrying that granularity across multiple applications is a challenge. It is simpler to allow one application to have complete access to all the data in another, without any attempt to pass the authentication granularity down the chain. However, this leaves the system open to greater exploit in case of a compromise of one of the upstream apps.

To combat this, avoid opening access to the full breadth of any particular application or integration endpoint. For example, do not provide the full set of an e-commerce system’s capabilities as a single API. Instead, allow access selectively, with granular APIs exposing specific functionality such as inventory look-up or shopping cart management, and use an API gateway as a policy enforcement point. This not only creates greater consistency and reduces the chance of errors in implementation of security, but also reduces the burden on development and integration teams.

Critical business data integration requires additional protections, such as encryption of data, but all integrations require a design that enforces proper authentication and authorization granularity. For your point-to-point integrations, follow the vendor-provided frameworks to enforce secure communication over TLS 1.2 or newer and also to implement the necessary authentication and authorization elements.

Any exposed URL or API endpoint is a potential attack vector. In addition to solving for access control, you must also account for denial-of-service attacks, exploits and abuse. For additional guidance related to security, identity and access management for cloud-hosted applications and APIs, see the following Gartner research:

- "Comparing Security Controls and Paradigms in AWS, Google Cloud Platform and Microsoft Azure" (https://www.gartner.com/document/code/343562?ref=authbody&refval=3902300)
- "Protecting Web Applications and APIs From Exploits and Abuse" (https://www.gartner.com/document/code/317335?ref=authbody&refval=3902300)
- "Selecting the Right API Gateway to Protect Your APIs and Microservices" (https://www.gartner.com/document/code/349440?ref=authbody&refval=3902300)
- "Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST" (https://www.gartner.com/document/code/277246?ref=authbody&refval=3902300)
- "Defining Cloud Web Application and API Protection Services" (https://www.gartner.com/document/code/337777?ref=authbody&refval=3902300)

Assessment Criteria

These criteria describe desirable characteristics of an integration solution in the context of an overall integration portfolio. Ratings for each of the approaches against these criteria, along with the rationale for those ratings, are in the Integration Approaches section of this assessment.

Note that these ratings are for the integration approaches in the abstract, and the specifics of your implementation of any given approach will have the final say in determining how well — or how poorly — that solution meets your needs.

Ability to Adapt

Integrations do not last forever. At some point, you will replace one or more of the sources or targets of any given integration. This may be due to a change in provider, such as a switch from one SaaS provider to another. Or, it may be a change in implementation, such as moving from an on-premises installation of an enterprise financial application to a cloud-hosted version. This criterion ranks the approaches in terms of the amount of work required to make changes in such circumstances:

- An approach is **Poor** if it is designed to work specifically with a given set of applications and if changing out any of those applications requires that the integration be rebuilt from scratch.
- An approach is **Excellent** if swapping out any of the applications involved in an integration is a simple, painless process.

If any of the applications to be integrated are likely to change, you should weight this criterion highly. Conversely, if the applications are likely to be stable in the long term, this criterion is less important.

Ability to Extend

Integrations are dynamic. Your needs may change, an application update may create the opportunity for improvement, or you may discover flaws in an integration. This criterion ranks the approaches in terms of the ease of making changes to existing integrations:

- An approach is **Poor** if changes to the integration require effort comparable to reimplementing from scratch, including the entire cycle of design, development, testing and deployment.
- An approach is **Excellent** if making changes is simple, straightforward and can be integrated with your existing continuous integration pipeline.

Ability to Monitor

Integrations have business value. When the integration is not working, or is degraded, that value is diminished or lost, and your organization could be losing vast amounts of money. This criterion ranks the approaches in terms of their ability to provide visibility into integration uptime, performance and quality. This may take the form of integrating with your existing monitoring suite, or it may rely on the native capability of the solution to provide dashboards, metrics, reporting and alerting:

- An approach is **Poor** if it neither offers native capabilities for monitoring and alerting nor exposes information about integration status and performance for use in your existing monitoring solutions.
- An approach is **Excellent** if it provides both built-in solutions for monitoring and alerting and the ability to expose information about integration reliability and performance to your existing monitoring and alerting systems.

The weighting of this criterion is linked both to the importance of the integration itself — mission-critical integrations deserve first-class monitoring — and to the existence of monitoring and alerting solutions in your organization. If you do not have robust monitoring capabilities for your application portfolio in general, you may not value such capabilities for integrations as highly.

Ability to Reuse

Integration is pervasive in digital business. As you rationalize and modernize your application portfolio, the need for application integrations will increase. The proliferation of SaaS solutions and the move of organizations to the cloud provide fuel for this fire. This criterion ranks the approaches in terms of their ability to support multiple integrations for a single source system as well as their ability to enable reuse of integration artifacts such as process orchestration logic or data mapping:

- An approach is **Poor** if it is custom-made or unique to the integration at hand and if artifacts are hard to discover and apply in other contexts.
- An approach is **Excellent** if integration assets can be reused for other integrations and if the integrations themselves can be exposed for reuse by other applications.

You should weight this criterion highly if you plan to leverage your integration approach for other integrations in the future, or if you want to use, orchestrate or choreograph these services as part of a modern service-oriented architecture. If the systems involved are unlikely to be involved in other integrations, or if you do not believe that the business logic or functionality of this integration will be useful in other contexts, you should weight this criterion lower.

You should weight this criterion highly if you have a high degree of uncertainty in your requirements or if the applications on either side of the integration have a history of frequent releases. In these cases, you will likely make frequent changes to your integrations.

Security and Data Governance

Integrations represent potential risk and exposure. Not only is any exposed integration endpoint a potential attack vector, but integrations may also involve sensitive or proprietary data — for example, a customer’s personally identifying information (PII), payment data or trade secrets. Care for your customers, your business and your reputation makes exercising control over your data and ensuring that your integrations are secure primary concerns. This criterion ranks the approaches in terms of the support they provide for implementing security and the degree of control they afford over the path that data takes:

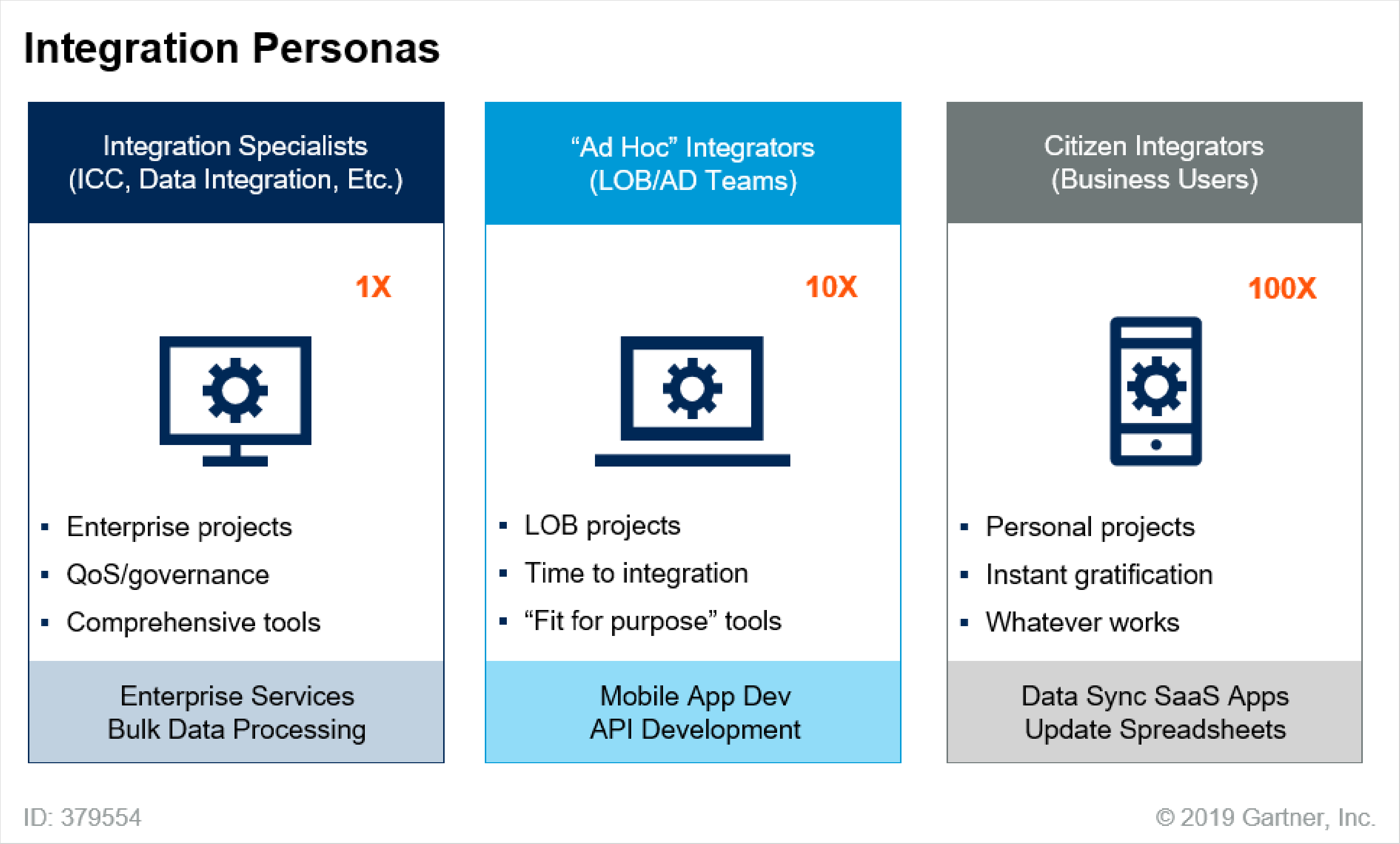
- An approach is **Poor** if it does not support encryption and secure connectivity, it does not offer granular authentication and authorization, and it runs on infrastructure you do not control.
- An approach is **Excellent** if it offers encryption both in transit and at rest, it supports granular authentication and authorization, integration workloads are under your control, and it provides guardrails for implementing security.

You should weight this criterion highly if the integration involves sensitive, business-critical or customer data or if the systems involved need a high level of security and access control.

Support for Ad Hoc Integrators

Integrations enable digital business. Placing all of the work of creating application integrations on a single team of integration specialists creates bottlenecks and risk. Gartner identifies three personas involved in integration: integration specialists, ad hoc integrators and citizen integrators, as shown in Figure 8. Specialist integrators are staff whose full-time job is creating and managing integrations. Ad hoc integrators are staff whose job requires them to create integrations, but who don't specialize in it. Often, these are developers or technical staff. In other cases, these people may be nontechnical team members who are the most knowledgeable about the data, functionality and business rules involved in integrating two applications. Enabling ad hoc integrators to implement and update integrations can reduce the load on overworked integration specialist teams. For additional information on the ad hoc integrator role, see "Expand Your Application Integration Capacity by Enabling Ad Hoc Integrators (https://www.gartner.com/document/code/355302?ref=authbody&refval=3902300) ."

Figure 8. Three Integration Personas



Source: Gartner (February 2019)

This criterion ranks the approaches in terms of their support for including staff other than integration specialists in the integration process:

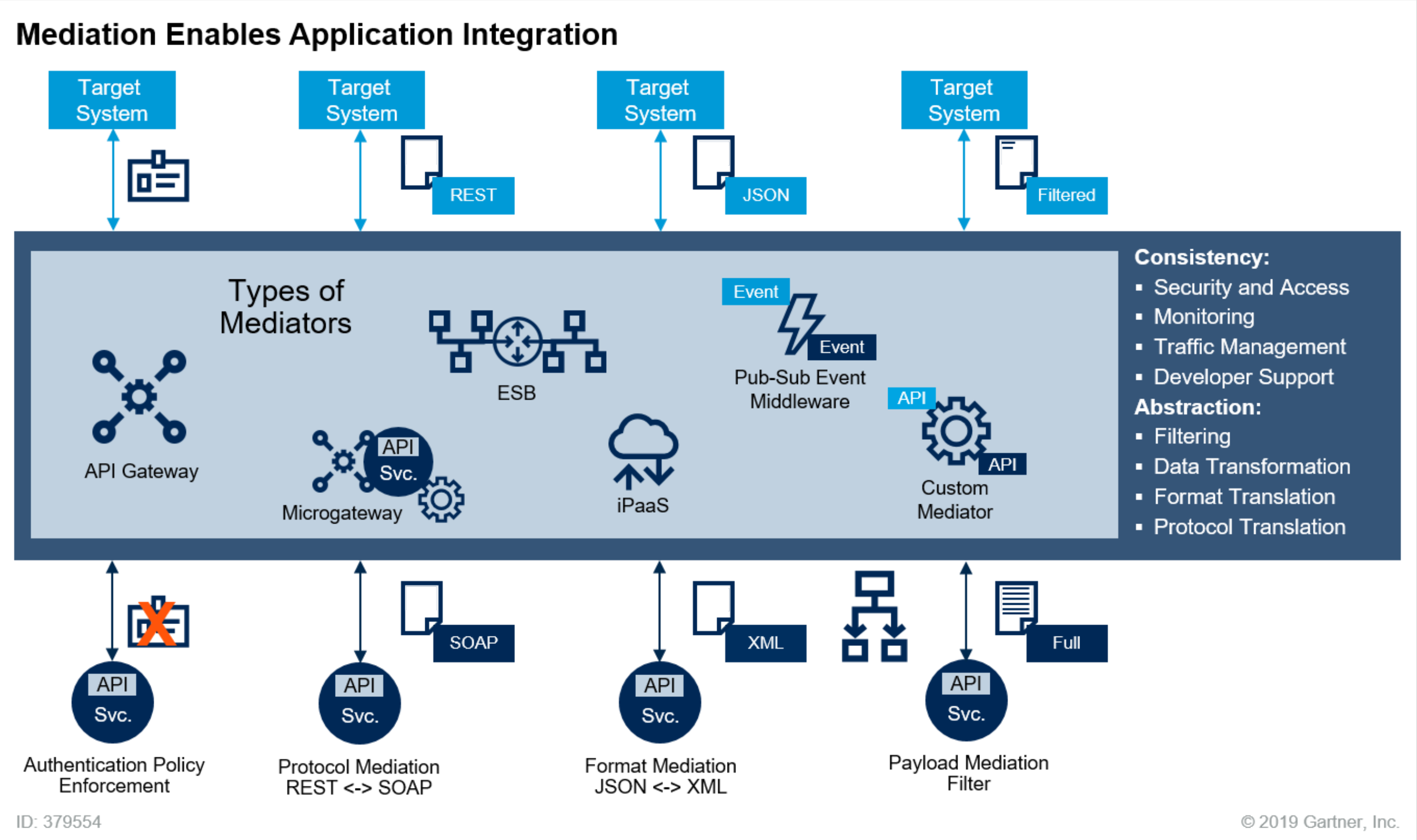
- An approach is **Poor** if it requires specialized technical skills to implement or update integrations.
- An approach is **Excellent** if it offers UI-based tooling for implementing and updating integrations in a drag-and-drop fashion.

You should weight this criterion highly if you have staff outside of your integration team who are ready and willing to engage in creating application integrations. If the integrations you need to create are highly complex or deeply technical in nature, you should weight this criterion lower.

Support for Mediation

Integrations need to span disparate endpoints. For example, you may need to connect a legacy system that exports data as comma-separated values (CSV) files with a SaaS application that offers only a REST API endpoint for data upload. As shown in Figure 9, mediation is supported by a variety of technologies.

Figure 9. Mediation Capabilities



Source: Gartner (February 2019)

This criterion ranks the approaches in terms of their ability to mediate between disparate styles and to provide transformations between protocols and formats:

- An approach is **Poor** if it forces you to conform to a limited set of predefined options.
- An approach is **Excellent** if it supports translating between a range of styles, protocols and formats, including the ability to extend its capabilities to support unique or niche applications.

You should weight this criterion highly if your planned integration includes applications with different integration styles, protocols or formats that are unique or niche. Conversely, if you will be connecting applications that use common protocols and formats, you should decrease the weighting of this criterion.

Support for Reliable Communications

Systems are not always available. Whether due to application failures or network issues, any of the applications that you are connecting may be unavailable.

This criterion ranks the approaches in terms of their ability to compensate for intermittent or unreliable connections by incorporating retry logic and persisting data:

- An approach is **Poor** if it offers no support for persistence or retries.
- An approach is **Excellent** if it offers robust support for persistence and retries and is independent from any of the applications being integrated.

You should weight this criterion highly if you know that you will have an unreliable connection, or if you need highly reliable communications between the parties.

Detailed Scores for Integration Approaches

Integration Platform Software

Integration platform software is a common technology choice for exposing application functionality and data for reuse. If your organization already has this technology in place as an ESB or DIP, it can play a key role in your integration strategy, not only exposing functionality, but also managing the integrations between applications in your portfolio. Integration platform software offers strong capabilities across the board, but requires specialized skills to use effectively, as seen in Table 2.

Table 2: Rating Details for Integration Platform Software

Criterion ↓	Rating ↓	Justification for Rating ↓
Ability to adapt	4	**Good —** ESBs and DIPs are optimized for integrating multiple different kinds of endpoints, so swapping out either end is relatively straightforward.
Ability to extend	3	**Fair —** ESBs and DIPs require expertise to implement integrations, but provide tooling to support testing and deployment of changes.
Ability to monitor	4	**Good —** ESBs and DIPs offer native solutions for monitoring the reliability and performance of integrations.
Ability to reuse	4	**Good —** Reuse of integration is a core competency of ESBs and DIPs. Not only can artifacts such as data maps be reused, but the integrations themselves can be exposed as services.

Criterion ↓	Rating ↓	Justification for Rating ↓
Security and data governance	5	**Excellent —** You retain control of data throughout its path because an ESB or DIP is located within your data center or on (possibly virtual) hardware you control. These solutions also provide “guardrails” in the form of existing controls and act as a policy enforcement point for implementing security.
Support for ad hoc integrators	2	**Limited —** ESBs and DIPs have a high learning curve, requiring specialized skills to use effectively.
Support for mediation	5	**Excellent —** Translating between integration styles, protocols and formats is a core capability of ESBs and DIPs.
Support for reliable communications	4	**Good —** Because an ESB or DIP is in between the connecting systems, it can persist and retry as needed, and it can monitor connectivity.
Scale: 1 = Poor; 2 = Limited; 3 = Fair; 4 = Good; 5 = Excellent		

Source: Gartner (February 2019)

This approach is the right choice for a given integration when most or all of the following are true:

- You have a large portfolio of integrations to manage.
- You need to transform or modify the data being passed between the applications to make the integration work.
- You need to reuse the data or functionality of the source system for other integrations.

This approach is a poor choice when either or all of the following are true:

- You want to enable ad hoc integrators to assist in implementing integrations.
- Your application portfolio is pivoting to the cloud.

iPaaS

Instead of operating an integration platform yourself, you can choose to use an iPaaS. In addition to cloud-native organizations such as Dell Boomi, Jitterbit and SnapLogic, long-term players in the ESB space — such as MuleSoft, Oracle and TIBCO Software — now offer iPaaS. An iPaaS offers strong capabilities across the board, but may pose concerns about security and data governance, as seen in Table 3.

Table 3: Rating Details for iPaaS

Criterion ↓	Rating ↓	Justification for Rating ↓
Ability to adapt	4	**Good —** iPaaS is strongly optimized for this, and offerings include a library of existing integrations that makes swapping out one system for another relatively painless.
Ability to extend	4	**Good —** Drag-and-drop approaches make changes easy, although support for continuous integration pipelines may be limited, requiring manual deployment of changes between environments.
Ability to monitor	4	**Good —** All iPaaS offerings include native capabilities for monitoring integration quality and performance.
Ability to reuse	5	**Excellent —** Integration assets such as data mappings and orchestration can be reused. Integrations can be exposed with a service façade or API, and can be managed and used as part of a service-oriented architecture approach.
Security and data governance	3	**Fair —** Although leading vendors provide the ability to deploy integration workloads to your own data center, giving you greater control, metadata about your integrations will still flow through the cloud. If you take advantage of data persistence or want to leverage retries, your data may be at rest on the iPaaS as well. These solutions generally provide good security, but sharply limit your ability to implement additional security if the built-in options do not meet your needs.
Support for ad hoc integrators	5	**Excellent —** iPaaS products excel here. All offer at least good support for this, with drag-and-drop UI approaches and tools to assist nontechnical users with creating integrations.
Support for mediation	5	**Excellent —** Data can be temporarily persisted in the platform, and conversion between styles, protocols and formats is a core capability of these platforms.
Support for reliable communications	5	**Excellent —** The platform can persist and retry, and because it is external to both ends of an integration, it is not affected by the loss of any given connection.

Scale: 1 = Poor; 2 = Limited; 3 = Fair; 4 = Good; 5 = Excellent

Source: Gartner (February 2019)

This approach is the right choice for a given integration when most or all of the following are true:

- You have a large portfolio of integrations to manage.
- You need to transform or modify the data being passed between the applications to make the integration work.
- You need to reuse the exposed data or functionality for other integrations.
- You want to enable ad hoc integrators to assist in implementing integrations.
- You want to outsource the operational aspects of your integration middleware.
- Your application portfolio is pivoting to the cloud.

This approach is a poor choice when some or all of the following are true:

- You have multiple custom-made or niche applications.
- Most or all of the applications you want to connect are in your data center.
- You want a high degree of control of the inner workings of your integrations.

SaaS Vendor Tooling

SaaS vendors such as Salesforce, Adobe's Marketo and Atlassian offer out-of-the-box integrations with specific partners. In addition, third parties offer extensions or plug-ins that enable easy integration between applications, such as the integrations available in the Atlassian Marketplace to link its Jira Software with requirement management systems. In all of these cases, the result is a direct integration between two applications. These integrations are easy for nontechnical users to implement and update, but come with a cost of flexibility and control, as seen in Table 4.

Table 4: Rating Details for SaaS Vendor Tooling

Criterion	Rating	Justification for Rating
Ability to adapt	1	**Poor —** These solutions are designed to work specifically with the applications being integrated. Changing out the SaaS application means starting over from scratch.
Ability to extend	4	**Good —** These solutions offer a configuration-based approach that makes updating the parameters of the integration between applications relatively painless.

Criterion ↓	Rating ↓	Justification for Rating ↓
Ability to monitor	2	**Limited —** Real-time access to data about integration reliability and performance is limited or unavailable. The more use you make of this style, the greater the number of unmonitored point-to-point integrations you will have, and the more fragile your systems will be.
Ability to reuse	2	**Limited —** Each integration must be configured independently. There is no opportunity to reuse artifacts such as data mapping across multiple integrations.
Security and data governance	2	**Limited —** Connectivity and the data path are controlled by the vendor tooling. Security implementation is at the mercy of the SaaS provider. Third-party plug-ins, in particular, represent potential risks.
Support for ad hoc integrators	4	**Good —** The configuration-based approach of vendor-supplied tooling offers an easy onramp for nontechnical staff members to implement integrations.
Support for mediation	1	**Poor —** These tools assume API-based connectivity with conforming data models. Support for custom-made or niche applications or new protocols is absent or limited.
Support for reliable communications	2	**Limited —** You are at the mercy of the tool vendor. Mitigating the shortcomings of the tool will require additional custom work.
Scale: 1 = Poor; 2 = Limited; 3 = Fair; 4 = Good; 5 = Excellent		

Source: Gartner (February 2019)

This approach is the right choice for a given integration when most or all of the following are true:

- The data or domain model of the systems being integrated is compatible out of the box, or the necessary transformations are supported by the provider's built-in capabilities.
- The integration is between systems that are stable and well-supported with low likelihood of change.
- The vendor's assurances about encryption and data security meet your requirements.

This approach is a poor choice when either or all of the following are true:

- One or more of the applications being integrated have other integrations already in place or needed.
- Either or both of the applications are expected to change frequently or be replaced in the near future.

Custom Code

Development teams always have the option to create a direct integration between systems. This may take the form of a stand-alone application, or it may be an extension of one of the applications being integrated. It may be written entirely from scratch or leverage open-source frameworks such as Apache Camel or Spring, but in all cases, this type of integration couples two or more systems directly to each other. The flexibility of custom code provides the ability to integrate between otherwise-incompatible applications, but these integrations are hard to reuse, as seen in Table 5.

Table 5: Rating Details for Custom Code

Criterion ↓	Rating ↓	Justification for Rating ↓
Ability to adapt	2	**Limited—** Solutions are coded to specific endpoints and must be updated if any of the applications involved are replaced. Consistent use of modern design and development practices can improve this score.
Ability to extend	2	**Limited—** Changes require development, testing and deployment of code. Good development practices, including test-first development and continuous integration, reduce the difficulty of making changes.
Ability to monitor	3	**Fair—** Because you control the implementation, you can choose to expose real-time data about integration quality and performance. However, development teams facing pressure to deliver on time may descope lower-priority features, such as monitoring, to meet project deadlines.
Ability to reuse	1	**Poor—** Each integration is coded separately. Reuse of integrations requires duplicating code. Discovery and reuse of artifacts is sharply limited.

Criterion	Rating	Justification for Rating
Security and data governance	3	**Fair—** Although you have direct control over the path taken by data and have the ability to encrypt it in transit and at rest as needed, you are at the mercy of your development team to implement secure coding practices. These solutions do not provide the “guardrails” that an integration platform does.
Support for ad hoc integrators	1	**Poor—** Creating integrations requires programming skills, which excludes businesspeople from the process. Doing integrations well is the domain of integration specialists, which will exclude a significant portion of your development teams from this work.
Support for mediation	4	**Good—** This approach offers the ability to mediate between otherwise-incompatible protocols or domains. The only limitations are the amount of work involved and the complexity of the resulting solution.
Support for reliable communications	3	**Fair —** Implementing retry logic and other reliable communications patterns is possible, at the cost of increased solution complexity.
Scale: 1 = Poor; 2 = Limited; 3 = Fair; 4 = Good; 5 = Excellent		

Source: Gartner (February 2019)

This approach is the right choice for a given integration when most or all of the following are true:

- You have the skill set in-house to design, develop, deploy, operate and maintain the integration.
- You are implementing a small number of integrations.
- You need tight control of the integration’s behavior.
- The integration is between applications that are stable and well-supported, with low likelihood of change.

This approach is a poor choice when either or both of the following are true:

- You need to reuse the data or functionality of the source system for other integrations.
- Either or both of the applications are expected to change frequently or be replaced in the near future.

fPaaS

An emerging pattern is the use of function platform as a service, such as AWS Lambda or Azure Functions, to deliver a customized set of integration capabilities. Like custom coding of individual integrations, this approach values the fit of a solution to the organization's needs highly, but is aimed at creating a suite of integrations. One example of this approach is the Telegraph Media Group's newsroom-trello-commons project. [2]

Note that implementing integrations in this way requires not only the skill set to do so in the near term, but an organizational willingness to support the platform in the long term. This means both a budgetary commitment to funding the staff, systems and processes for the platform and a commitment to treating the platform as a product rather than a delivery project. The ratings for this approach are seen in Table 6.

Table 6: Rating Details for fPaaS

Criterion ↓	Rating ↓	Justification for Rating ↓
Ability to adapt	2	**Limited —** These systems are designed to integrate specific applications, but their API-centric integration models grant a degree of portability.
Ability to extend	4	**Good —** Built with DevOps practices and continuous delivery, these systems optimize for the ability to deliver changes quickly.
Ability to monitor	3	**Fair —** These systems leverage monitoring capabilities of the underlying cloud platform or integrate with existing organizational capabilities for monitoring and alerting, rather than offering native monitoring.
Ability to reuse	3	**Fair —** Adding new integrations within the scope of the platform is easy, but extending the platform to support other types of integration is challenging.
Security and data governance	3	**Fair —** You have the ability to encrypt data in transit and at rest as needed. You are at the mercy of your development team to implement secure coding practices. These solutions do not provide the "guardrails" that an ESB or iPaaS does, but the platform approach provides a consistent policy enforcement point.

Criterion ↓	Rating ↓	Justification for Rating ↓
Support for ad hoc integrators	1	**Poor —** Implementations that Gartner has seen to date have not provided UI elements to enable nontechnical users to create integrations. Although it is possible to implement such capabilities, it represents a significant addition to the scope.
Support for mediation	4	**Good —** Mediation between styles, protocols or formats is one of the primary reasons to build such a platform.
Support for reliable communications	3	**Fair —** Capabilities for persistence and retry logic need to be built as part of the system. The independence of the fPaaS from the systems being integrated is an advantage.
Scale: 1 = Poor; 2 = Limited; 3 = Fair; 4 = Good; 5 = Excellent		

Source: Gartner (February 2019)

This approach is the right choice for a given integration when most or all of the following are true:

- You have a deep understanding of your integration needs.
- No solution exists that meets your needs well.
- You have the skill set in-house to design and develop using a function-oriented approach.
- You have leadership support for building and operating a platform.

This approach is a poor choice when either or both of the following are true:

- Your organization thinks in terms of projects rather than products.
- You do not have a robust implementation of agile methods, DevOps and continuous integration.

Guidance

Choosing an integration approach is a hard problem. There is no “silver bullet” that will solve all of your integration complexities, especially because application portfolios have become radically decentralized, including a combination of SaaS, cloud-hosted and on-premises applications. Use these principles to guide your efforts.

Supply Your Organization With Multiple Integration Tools

No single tool, platform or approach provides the right answer for every integration implementation. Even if your ESB has extraction, transformation and loading (ETL) capability, it will not match the performance and versatility of a dedicated ETL tool for tasks such as encoding or data validation. Even if your iPaaS has API management capabilities, it will not match the feature set of a dedicated API management tool for capabilities such as monetization or traffic management. This does not mean that you must have a specialized tool for every integration, but you must be prepared for those cases where your existing tools reach their limits. For example, if a new integration requires sophisticated data mapping that exceeds the capabilities of your existing ESB or iPaaS, you should evaluate both ETL tools and custom solutions to determine the best approach.

Your goal is to create a “hybrid integration platform” to provide your organization with a well-defined set of integration capabilities that enables each integration to be implemented effectively. To make this happen, you should:

- **Inventory your existing integrations.** Identify your organization’s key applications and work with operations teams and business owners to identify current integrations. For each integration, capture metadata, such as the applications it connects and any integration tools used, as well as operational documentation and contract terms.
- **Determine the tools and expertise you already have.** Working from your integrations inventory, identify the unique set of tools in use. Work with the owners of each tool to identify and document its capabilities, the skills required to make use of it, and whether those skill sets are available in-house or through partners.
- **Document your existing solution patterns to enable reuse.** Use the logical integration flows and integration approaches in this research as a basis, but look for specific patterns that may be unique to your organization. Capture the patterns using both diagrams and descriptions, as well as code samples and specific information about architectural decisions made during implementation.
- **Provide easy access to information on integration patterns and capabilities.** Using a wiki, Microsoft SharePoint site or other means of information sharing, create a single repository of information about existing integrations and the tools and patterns in use. Ensure that this repository is easily accessible throughout your organization, and inform architecture, development and operations teams of the existence of this repository while encouraging them to make use of it and contribute to it.
- **Keep your repository up-to-date.** The information in your integration repository is useful for troubleshooting and operations, as well as for the development of new integrations. Ensure that it remains useful by repeating this process on a regular basis — at least quarterly.

For additional guidance on choosing integration tools, see "Enabling Agile Integration With a Distributed Integration Platform" (https://www.gartner.com/document/code/368277?ref=authbody&refval=3902300) and "Choosing Application Integration Technology." (https://www.gartner.com/document/code/314871?ref=authbody&refval=3902300)

Make Sure Existing Integration Tools Meet Your Needs Before Using Them

You will face pressure from your organization to maximize the value of existing tools by using them in as many cases as possible. Although you should consider existing tools as an option for new integrations, you should carefully evaluate whether these tools are the right choice on their own merits. Rather than trying to maximize the value of the integration platform, seek instead to maximize the value of your investment in each integration.

Your goal is to ensure that you are not suboptimizing any given integration for the sake of others, nor are you suboptimizing multiple integrations for the sake of a single one. To do this:

- Document the strengths and weaknesses of your existing integration tooling, keeping track of the cases where you have decided for or against using a given approach and why. This may take the form of a lightweight architecture decision record. [3] If so, consider using a storage option that allows hyperlinking, such as a wiki, to allow for easier cross-referencing of these decisions.

- Evaluate multiple options for each integration. Each time you consider a new integration, use this research as a tool to review each of the options in your portfolio. Even a cursory review of available options is valuable as a reminder of possibilities. Pay particular attention to new capabilities that have been added to the portfolio — you will know the least about those options.

- Challenge "that's how we've always done it" responses to integration design. Each integration is unique in some way — it if were not, it would not be a separate integration — and therefore, each should be evaluated on its own. Use your knowledge of existing integrations as a guide, not a constraint.

Design Security Into Your Integrations

Integration endpoints represent potential attack vectors for data breaches or application security and performance problems. You should evaluate your integrations into critical business applications with the same level of scrutiny and rigor that you exercise over the applications themselves. To do this:

- Implement granular security to lock down both authorization and access to the lowest level of functionality or data possible for an integration flow.

- Ensure that sensitive data is encrypted in transit and at rest, and use Transport Layer Security (TLS) 1.2 or newer to safeguard connectivity between endpoints.

- Deploy and use runtime protection technologies such as web application firewalls (WAFs), cloud application security brokers (CASBs) and API gateways to reduce the attack surface of your integrations and to inspect traffic for threats.

Capture Integration Artifacts to Aid Discovery, Troubleshooting, Updates and Reuse

As your integration portfolio becomes increasingly diverse, the difficulty of discovering which integrations you have, troubleshooting them when problems arise, updating them as needed and enabling reuse increases dramatically. To aid in these efforts:

- Create a repository where integration artifacts — such as design documents, data maps and business logic — can be stored.
- Store existing documentation and records in this repository for future use.
- Encourage integration teams to document and share information.
- Use architecture decision records to capture high-level decisions that can escape standard design artifacts.
- Prefer searchable data formats such as a wiki to enable ease of use.

The Details

Logical Application Integration Flow Use Cases

Integrations exist to link applications to one another, enabling the movement of data or the invocation of functionality. Understanding the interaction pattern of the participating applications provides a foundation for determining the optimal integration approach. Gartner has identified five use cases that represent the vast majority of application integrations discussed in our inquiries. Each time you are implementing an integration, determine which of these use cases applies, and use that to help weight your assessment criteria.

Each use case is represented as a logical data flow diagram. These diagrams do not include the nature or location of integration middleware because any of the use cases can be implemented using any of the integration approaches. The key to these diagrams is understanding the direction of information flow and the responsibility for making decisions about the interaction.

One-Time Move of Application Data

This use case applies when you need to copy or move a discrete set of information from one application to another. The applications involved or the integration itself must provide the necessary intelligence to transform or interpret the data in some fashion, such as by applying business logic to source data or by creating multiple entities in the target system. This is an application integration, not a data integration. If your needs can be met by a simple ETL process or data transfer between instances, you should choose that solution instead of creating an application integration.

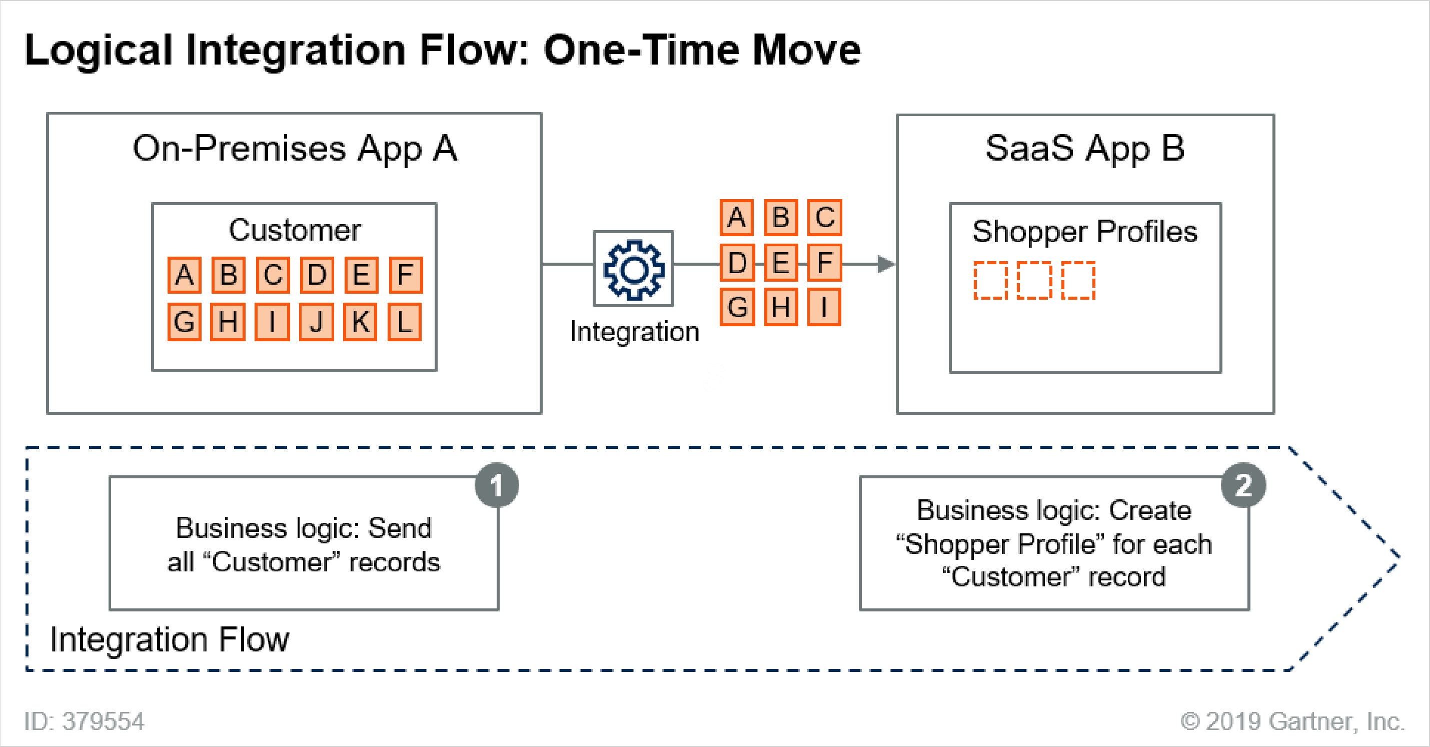
Although these integrations usually involve the movement of large amounts of data, regardless of size, the dataset to be moved must be bounded. If the data to be moved is open-ended, this will be a one-way sync, not a one-time move. In cases where the participating systems will coexist after the one-time move, a separate integration to provide ongoing sync or data access will also be required.

Examples include:

- Transitioning a given responsibility from one application to another, such as replacing one bug tracking system with another while retaining history and work in progress
- Moving from an on-premises installation of an application to a cloud-hosted instance or SaaS version
- Initial setup of a new instance of an existing application, such as a local copy of item pricing for a retail store location
- Initial population of a local data store for an application that will be fed by a subsequent one-way sync integration

The flow of information in this use case is shown in Figure 10. A source application, On-Premises App A, identifies which datasets to send to a target system, SaaS App B, which must process the data and apply business logic to create new entities.

Figure 10. One-Time Move of Information Between Applications


Logical Integration Flow: One-Time Move
On-Premises App A
Customer
A B C D E F
G H I J K L
Integration
A B C
D E F
G H I
SaaS App B
Shopper Profiles
1
Business logic: Send all "Customer" records
2
Business logic: Create "Shopper Profile" for each "Customer" record
Integration Flow
ID: 379554
© 2019 Gartner, Inc.

Source: Gartner (February 2019)

Although this pattern is often implemented as a batch, the specifics of implementation vary. Very large datasets may take multiple days to process and may require multiple cycles. In some cases, this "one time" process may be repeated many times as new copies of the dataset are needed.

One-Way Synchronization of Application Data

This use case applies when you need to keep a set of information consistent between two or more applications, where one application is the system of record and is making changes to that information on an ongoing basis. The source system is responsible for making information available to the target systems as the source becomes aware of new information. This ensures that the target systems do not have a runtime dependency on the source system at the cost of creating duplicates of the data in each target system.

This can be implemented in a variety of ways. The source system may expose an API, with updates, that is called by the targets, or the source may call APIs on the target systems to "push" data to them. The source may place messages on a queue or topic, or may drop files to a file system, with the targets retrieving them asynchronously. The key is that the source system is making the decisions about which data to send. (Contrast this with the on-demand data use case, as described in the On-Demand Data section, where target systems are responsible for identifying the dataset they need.)

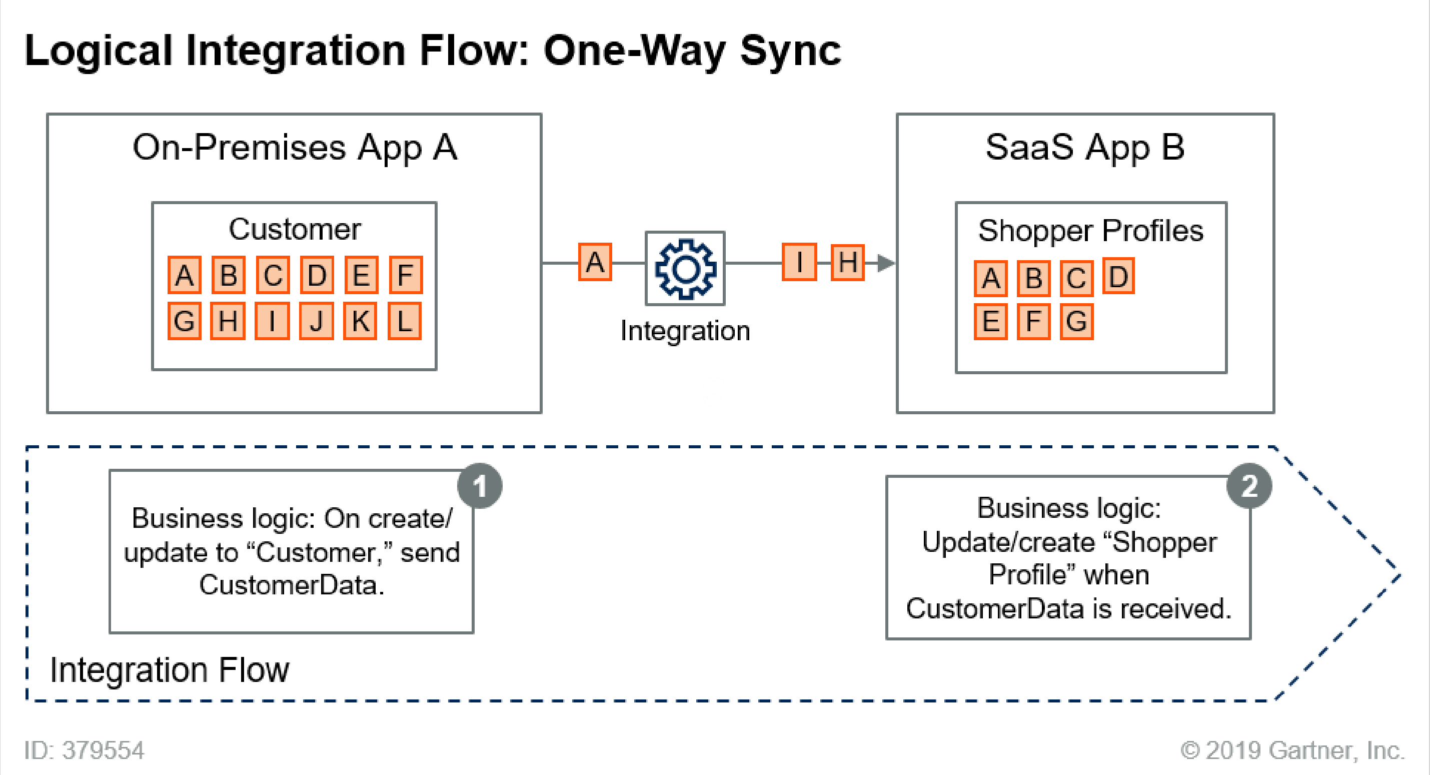
Examples of one-way sync include:

- Adding new items set up in an ERP system to the product catalog featured in an e-commerce application

- Updating item inventory as new products are received in the warehouse
- Streaming temperature data from sensors located in industrial machinery to an Internet of Things (IoT) platform in the cloud

The flow of information in this use case is shown in Figure 11. A source application, On-Premises App A, identifies new or changed information and makes it available to a target, SaaS App B, which must receive the information and store it or act on it as needed.

Figure 11. One-Way Sync of Applications


Logical Integration Flow: One-Way Sync
On-Premises App A
Customer
A B C D E F
G H I J K L
A
Integration
I H
SaaS App B
Shopper Profiles
A B C D
E F G
1
Business logic: On create/ update to "Customer," send CustomerData.
2
Business logic: Update/create "Shopper Profile" when CustomerData is received.
Integration Flow
ID: 379554
© 2019 Gartner, Inc.

Source: Gartner (February 2019)

Note that, although this is a "push" of information from a logical perspective, the implementation may not reflect this. For example, SaaS App B may be polling an API provided by On-Premises App A for changes, such as the getUpdated() and getDeleted() endpoints offered by the Salesforce SOAP API. [4]

Two-Way Synchronization

This use case applies when you have two or more systems that share a set of information and both systems are making ongoing changes to that data. Each system acts as source and target, sending and receiving information. Note that this is not the same as the Shared Data pattern documented in "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" [5] because the applications are not sharing a database. Instead, each system holds its own representation of the data objects, and nonshared information that is linked to the shared items will differ between the applications.

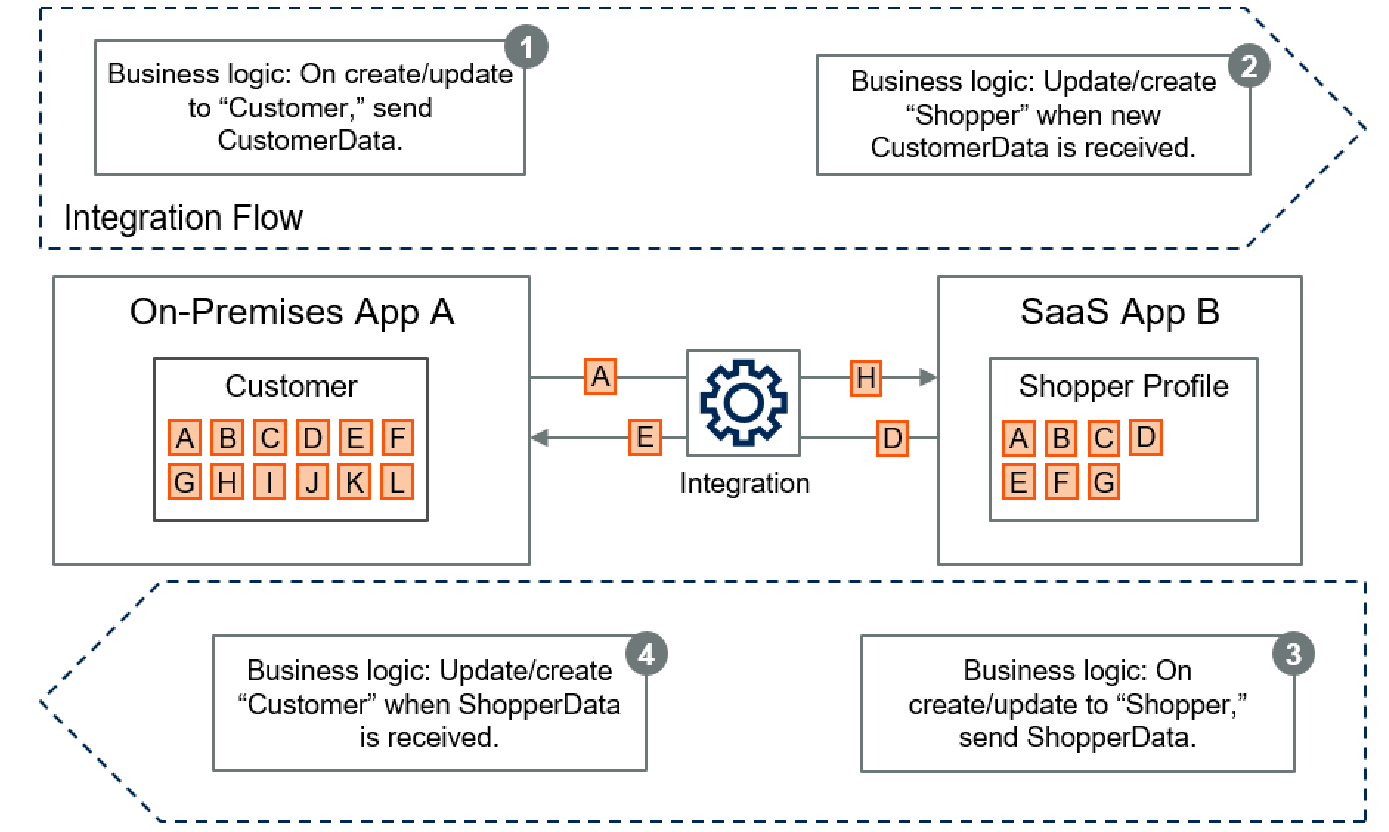
Examples of two-way sync include:

- A customer-facing application for creating trouble tickets that is paired with an associate-facing system for resolving them: Updates from associates must be visible to customers, and updates from customers must be visible to associates.
- An e-commerce system that allows customers to modify or cancel orders and that is paired with an order management system that controls order fulfillment: Canceled orders must not be shipped, and shipped orders must be prevented from being canceled.

The flow of information in this use case is shown in Figure 12. On-Premises App A shares a subset of its “Customer” data with SaaS App B’s “Shopper Profile” data. As each of the applications creates new entities or makes updates to the existing ones, it must send data to its partner system. In turn, each of the systems is also receiving data and creating or updating entities.

Figure 12. Two-Way Sync

Logical Integration Flow: Two-Way Sync


1
Business logic: On create/update to “Customer,” send CustomerData.
2
Business logic: Update/create “Shopper” when new CustomerData is received.
Integration Flow
On-Premises App A
Customer
A B C D E F
G H I J K L
A
E
Integration
H
D
SaaS App B
Shopper Profile
A B C D
E F G
4
Business logic: Update/create “Customer” when ShopperData is received.
3
Business logic: On create/update to “Shopper,” send ShopperData.

ID: 379554 © 2019 Gartner, Inc.

Source: Gartner (February 2019)

Two-way synchronization is challenging to get right. To be successful, you must have a means to identify new or changed information in each application, and all applications must be able to receive and process information in a timely manner to stay in sync. Data inconsistencies are inevitable in such a system. To deal with this, you must define and enact a consistency policy, such as:

- First update wins: The first update is accepted, and subsequent updates are ignored.
- Last update wins: As updates arrive, they overwrite previous updates to a given item.
- Trust hierarchy: Updates overwrite based on an explicit hierarchy.
- Merge changes: Updates are merged together to create a composite state using business logic.
- Escalation: Give the problem to a human to deal with.

The right policy (or policies) will vary depending on the needs of the specific integration. Policies that rely on ordering (first update or last update) or source (trust hierarchy) are easy to implement but have a high risk of the systems drifting out of sync. Merge strategies are complex because they require analyzing and executing business logic on the payloads. Relying on human intervention (escalation) is the most accurate, but it has the highest ongoing cost. If you do employ escalation, regularly analyze the inconsistencies and their resolutions to detect patterns and then implement these patterns as rules to reduce the workload on human operators.

On-Demand Data

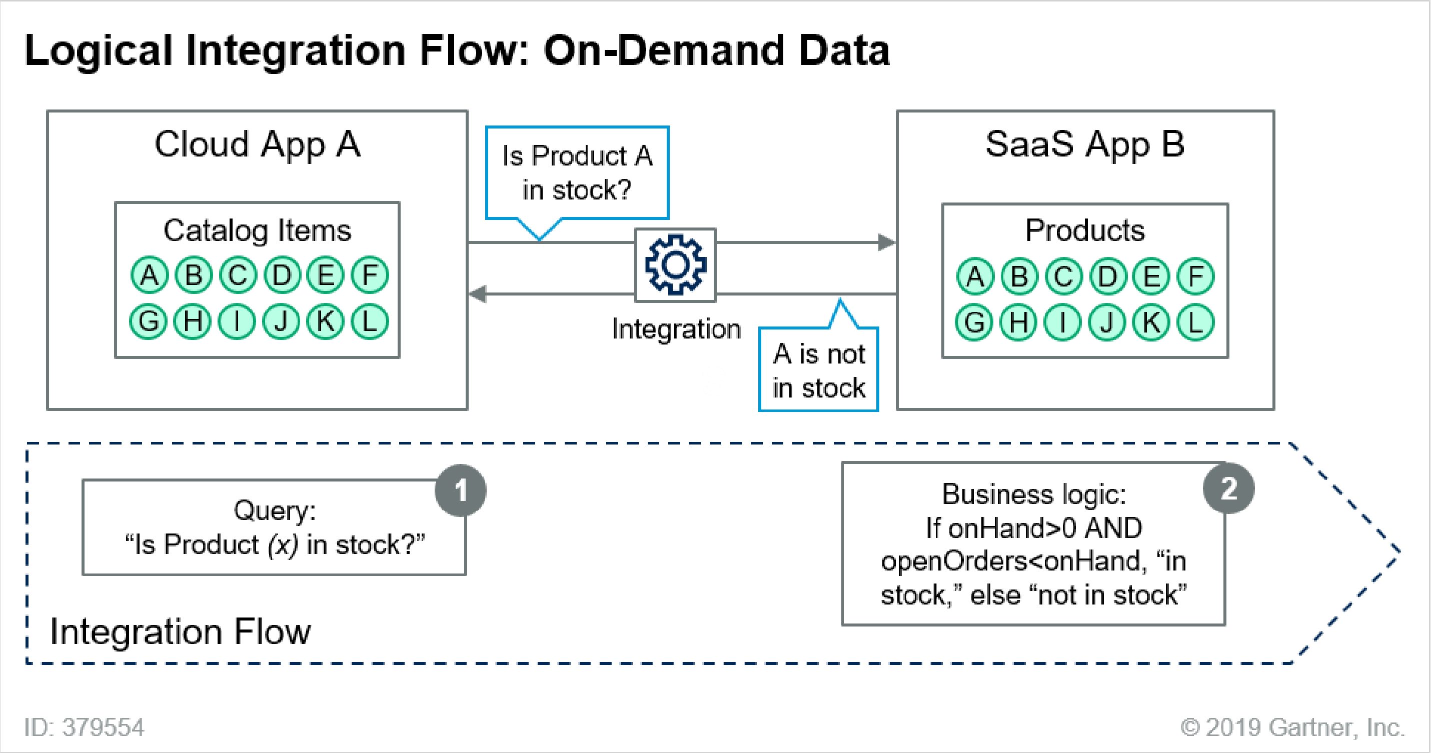
This use case applies when a target system needs to get specific information from a source system of record. The source makes information available via a well-defined interface, and the target requests that information as needed. This avoids duplication of data and reduces the risk of incorrect data being used to make decisions at the cost of creating a runtime dependency on the source system.

Examples of on-demand data include:

- An e-commerce system that must check to ensure inventory is available for a given product before taking an order
- A mobile application that requests the current balance of a user’s checking and savings accounts when displaying account information

The flow of information in this case is shown in Figure 13. A source system, SaaS App B, implements business logic for determining availability of products and provides interfaces for retrieving information. Cloud App A uses that interface to determine if products are in stock.

Figure 13. On-Demand Data



Source: Gartner (February 2019)

In contrast with one-way synchronization, where the source system is responsible for identifying the relevant data, the target system must have a means of controlling the dataset it will receive. Also, note that this is a read-only interaction — the calling systems are not making changes to the source.

On-Demand Functionality

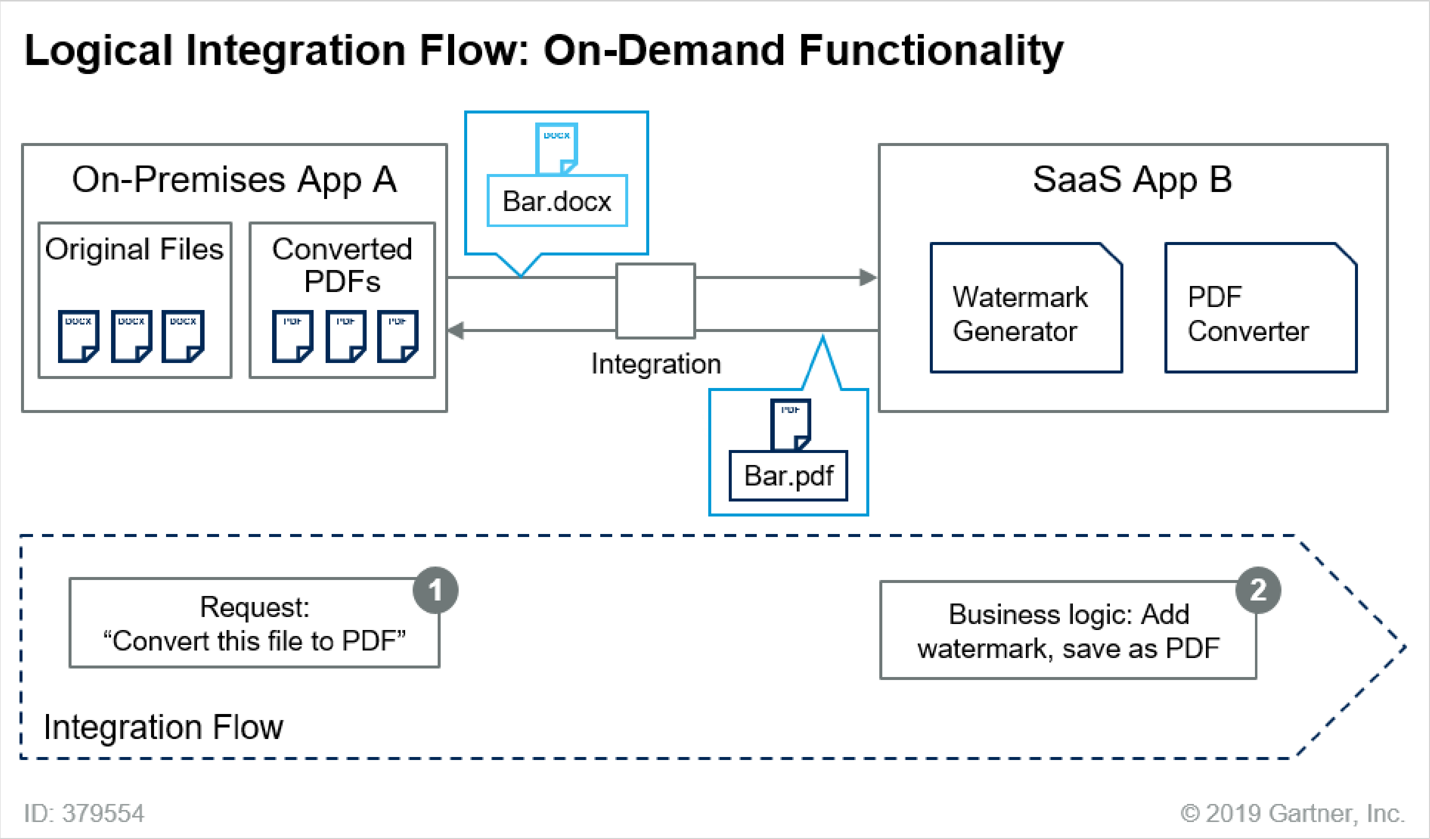
This use case applies when a target system needs a source system to perform an action. The source provides an interface for invoking this functionality, and the target calls it as needed, providing the data necessary to perform the requested action.

Examples of on-demand functionality include:

- An e-commerce front end calling the order management system to create an order for a customer
- A content management system sending a document to a conversion system to create a watermarked copy for customer download

The flow of information in this case is shown in Figure 14. A source system, SaaS App B, offers an interface for converting Microsoft Word files to a watermarked PDF. On-Premises App A invokes this functionality, sending in files and receiving PDFs in return.

Figure 14. On-Demand Functionality



Source: Gartner (February 2019)

The on-demand functionality approach can be implemented as a request/response API with the calling application waiting for the response, but this creates a runtime dependency and impairs overall system performance. To guard against this, you can use asynchronous techniques, such as messaging, or implement the API to provide an immediate response containing the URI of the result of processing in the Location header. [6]

Evidence

[1] "Lightweight Architecture Decision Records," (https://www.thoughtworks.com/radar/techniques/lightweight-architecture-decision-records) ThoughtWorks

[2] "Telegraph/Newsroom-Trello-Commons," (https://github.com/telegraph/newsroom-trello-commons) GitHub

[3] "Documenting Architecture Decisions," (http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions) Relevance

[4] "API Calls for Data Replication," (https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/calls_for_data_replication.htm) Salesforce

[5] G. Hohpe and B. Woolf. "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions." Addison-Wesley Professional. 10 October 2003

[6] "How Can I Let Users Create Resources That Might Take a Considerable Amount of Time. I Cannot Have My Users Wait on the API to Finish," (http://restcookbook.com/Resources/asynchroneous-operations/) REST CookBook

Document Revision History

How to Integrate Cloud-Hosted, SaaS and On-Premises Applications - 31 July 2018 (https://www.gartner.com/document/code/353947?ref=dochist)

Recommended by the Authors

Choosing Application Integration Technology (https://www.gartner.com/document/3496117?ref=authbottomrec&refval=3902300)

How to Assess Your Application to Adopt Cloud-Native Architecture (https://www.gartner.com/document/3813563?ref=authbottomrec&refval=3902300)

Assessing Event-Driven Middleware Technology for Modern Application Architecture (https://www.gartner.com/document/3655417?ref=authbottomrec&refval=3902300)

A Guidance Framework for Architecting Highly Available Cloud-Native Applications (https://www.gartner.com/document/3396654?ref=authbottomrec&refval=3902300)

Getting Started With Salesforce Sales Cloud Integration (https://www.gartner.com/document/3814772?ref=authbottomrec&refval=3902300)

Comparing Security Controls and Paradigms in AWS, Google Cloud Platform and Microsoft Azure (https://www.gartner.com/document/3877942?ref=authbottomrec&refval=3902300)

Protecting Web Applications and APIs From Exploits and Abuse (https://www.gartner.com/document/3688822?ref=authbottomrec&refval=3902300)

Selecting the Right API Gateway to Protect Your APIs and Microservices (https://www.gartner.com/document/3880018?ref=authbottomrec&refval=3902300)

Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST (https://www.gartner.com/document/3103918?ref=authbottomrec&refval=3902300)

Enabling Streaming Architectures for Continuous Data and Events With Kafka (https://www.gartner.com/document/3876001?ref=authbottomrec&refval=3902300)

Enabling Agile Integration With a Distributed Integration Platform (https://www.gartner.com/document/3890090?ref=authbottomrec&refval=3902300)

Recommended For You

Solution Path for a SaaS Adoption Framework (https://www.gartner.com/document/3969653?ref=algobottomrec&refval=3902300)

Evaluation Criteria for Public Cloud Application Platform as a Service (https://www.gartner.com/document/3894973?ref=algobottomrec&refval=3902300)

Use Monitoring for SaaS Despite Its Limitations (https://www.gartner.com/document/3957027?ref=algobottomrec&refval=3902300)

How to Evaluate SaaS Providers and Solutions by Developing RFP Criteria (https://www.gartner.com/document/3909079?ref=algobottomrec&refval=3902300)

Architecting Low-Code Cloud Applications With High-Productivity aPaaS (https://www.gartner.com/document/3883868?ref=algobottomrec&refval=3902300)