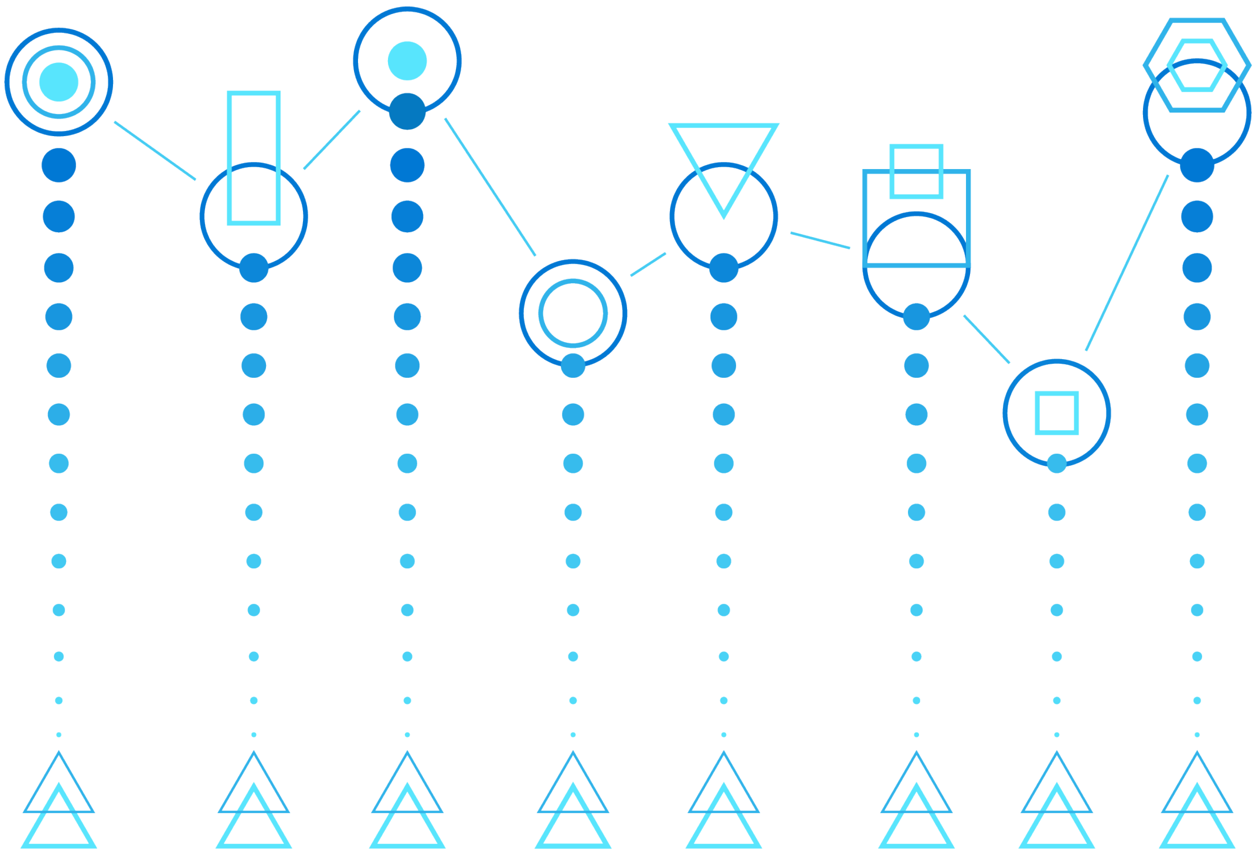
Microsoft

Azure Integration Services



© 2018 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

Some information relates to pre-released product which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

Contents

Integration today

No application is an island. To get the most from the software you build or buy, you need to connect it to other software. This means that effective application integration is essential for just about every organization.

Sometimes, all you need to do is connect one application directly to another. More often, though, application integration means connecting multiple independent systems, often in complex ways. This is why organizations commonly rely on specialized integration platforms that provide the services needed to do this. Like so much else today, those platforms have moved from on-premises datacenters into the public cloud. Rather than use traditional integration technologies such as BizTalk Server, more and more organizations are using *integration Platform as a Service (iPaaS)* solutions, i.e., cloud-based integration platforms

To meet this need, Microsoft provides *Azure Integration Services*. This iPaaS solution is a set of cloud services for mission critical enterprise integration. To achieve this goal, these services provide the four core technologies required for cloud-based integration. Figure 1 illustrates those four.

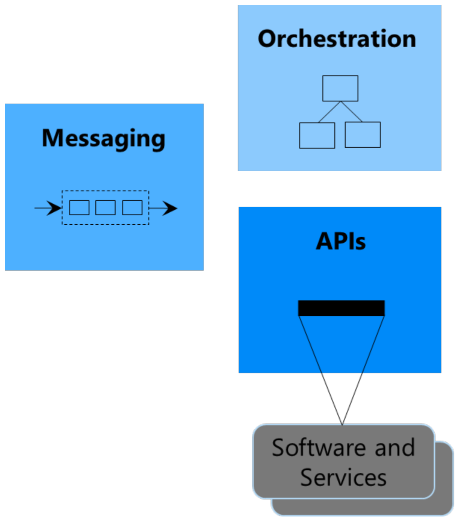


Figure 1: Modern cloud-based integration relies on four primary services.

The four core cloud services required for integration today are:

- A way to publish and manage application programming interfaces (APIs). This *API service* makes software services accessible to other software, whether those services run in the cloud or on-premises.
- A straightforward way to create and run integration logic. For example, you might need to implement a business process that relies on several different applications, all accessed via APIs. To create this kind of workflow, an iPaaS provides *orchestration*, typically with a graphical tool for defining the workflow's logic.

- Some way for applications and integration technologies to communicate in a loosely coupled way via *messaging*. This service provides queues that hold messages until they can be picked up by the receiver. This lets applications and integration software communicate asynchronously, even across diverse technology platforms, something that's often required in integration scenarios.

- A technology that supports communication via *events*. Rather than polling a queue in a messaging service, for example, it's sometimes easier and more efficient to learn about changes by receiving an event.

These cloud services, sometimes combined with other cloud technologies, can be used to integrate both cloud and on-premises applications. As described next, Azure Integration Services provides all four.

Azure Integration Services: The big picture

The four components of Azure Integration Services today are:

- *API Management*, which provides an API service.
- *Logic Apps*, supporting orchestration of business processes, workflows and more.
- *Service Bus*, providing reliable enterprise messaging.
- *Event Grid*, which allows raising and delivering events.

Figure 2 shows these four.

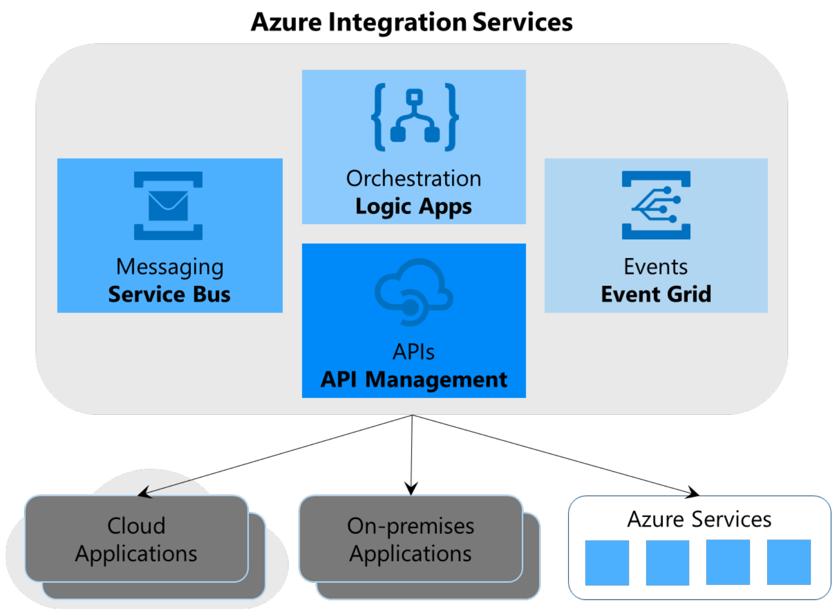


Figure 2: Azure Integration Services lets you connect cloud and on-premises applications through a unified set of cloud services.

It's important to realize that while these four services work together to provide an iPaaS, they're sold individually. You're free to use only the services you need. In fact, a significant difference between traditional on-premises integration software, such as BizTalk Server, and Azure Integration Services is that you don't need to purchase every service to get any of them. In the cloud, you can choose exactly which services you need, then pay only for those.

How would you use Azure Integration Services? Here are some of the most common scenarios:

- Connecting applications inside your organization. The applications might run in your own on-premises datacenter, in the cloud, or a mix of both. This kind of enterprise application integration (EAI) has been important for decades; it's now being adapted for a hybrid world.
- Connecting applications in your organization with those of a business partner. Commonly known as business-to-business (B2B) integration, this often relies on standard formats such as Electronic Data Interchange (EDI). As described later, Azure Integration Services supports these widely used standards.
- Connecting applications inside your organization to Software as a Service (SaaS) applications. As the business applications you buy continue to shift to SaaS, application integration requires connecting with these cloud services. For example, a business process in a hybrid environment might need to access both your on-premises SAP system and a SaaS CRM solution such as Salesforce.
- Integrating applications with Internet of Things (IoT) devices. The ever-increasing popularity of IoT raises a variety of new integration challenges. A cloud-based solution—an iPaaS—is uniquely suited to address these, as it can be accessed by devices running anywhere.

The Value of Azure

The four components of Azure Integration Services address the core requirements of application integration. Yet real scenarios often require more. Maybe your integration application needs a place to store unstructured data, for example, or a way to include custom code that does specialized data transformations. Because Azure Integration Services is part of the larger Azure cloud platform, these things are readily available. You might store unstructured data in Azure Data Lake Store, for instance, or write custom code using Azure Functions, a serverless compute technology.

Don't think of Azure Integration Services as an isolated, standalone iPaaS—it's not. Instead, it's part of the larger Azure offering, giving you immediate, integrated access to any other cloud services you need.

Azure Integration Services: Technologies

Azure Integration Services is a unified set of technologies. Still, to understand this set, you need to know the basics of each of its four components. What follows walks through each one, describing what it provides and why you'd want to use it.

API Management

Whether they run on-premises or in the cloud, modern applications typically expose at least some of their functionality through APIs. This lets integration logic access an application's functionality through its APIs. For example, a workflow orchestration might implement a complete business process by invoking different APIs in different applications, each of which carries out some part of that process.

Yet making APIs available for other software to call is harder than you might think. In fact, just exposing an application's API directly on your network isn't sufficient for typical integration scenarios. To do this effectively, you need to have solid answers to questions such as these:

- How do you limit the number of calls your application can receive from clients? Most applications that expose APIs can't handle an unlimited number of requests, so you need a way to control this.
- How do you control how calls to your application APIs are authenticated? Security is always important, especially for applications accessible via the public Internet.
- How do you make sure your APIs are fast enough? You might want to cache responses to frequent requests, for example, to speed up responses.
- How do you monitor and analyze how your APIs are being used? Patterns in API usage might indicate a trend that significantly impacts your business.
- How do you make it easy for developers to use your APIs? You need a way to provide them with documentation, code samples, and more.

Azure API Management addresses all these concerns. Figure 3 summarizes the role this cloud service plays.

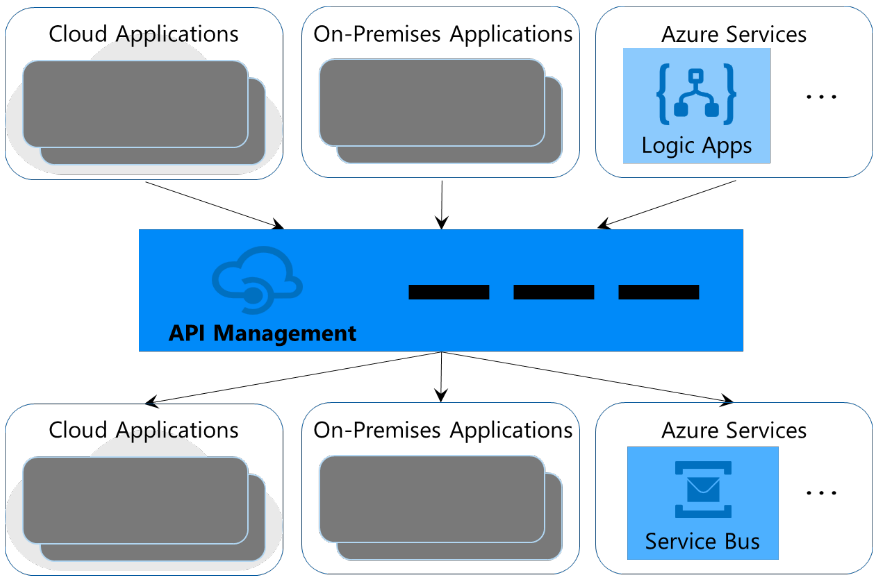


Figure 3: API Management exposes APIs from backend services to diverse clients.

API Management can expose REST and SOAP APIs from all kinds of backend software, including applications running in the cloud and applications running on premises. The technology stack used to build that software—.NET, Java, or anything else—is irrelevant; API Management works with anything. Even other Azure services can be exposed through API Management, letting you put a managed API on top of Service Bus, Logic Apps, and other services if you choose.

To do this, API Management creates facades—proxies—of the real APIs in backend applications. Clients, including Logic Apps, cloud and on-premises applications, and others, can call these facades. All that's required is that the client be able to create and receive HTTP requests. Each call is eventually passed on to the underlying application, but as described below, API Management can provide several useful services before this happens.

The fundamentals of what API Management does aren't hard to understand. But this simple picture raises some obvious questions. How are the APIs of backend services made available through this API service? And how do developers learn what they need to know to call these APIs? Figure 4 illustrates the answers to both questions.

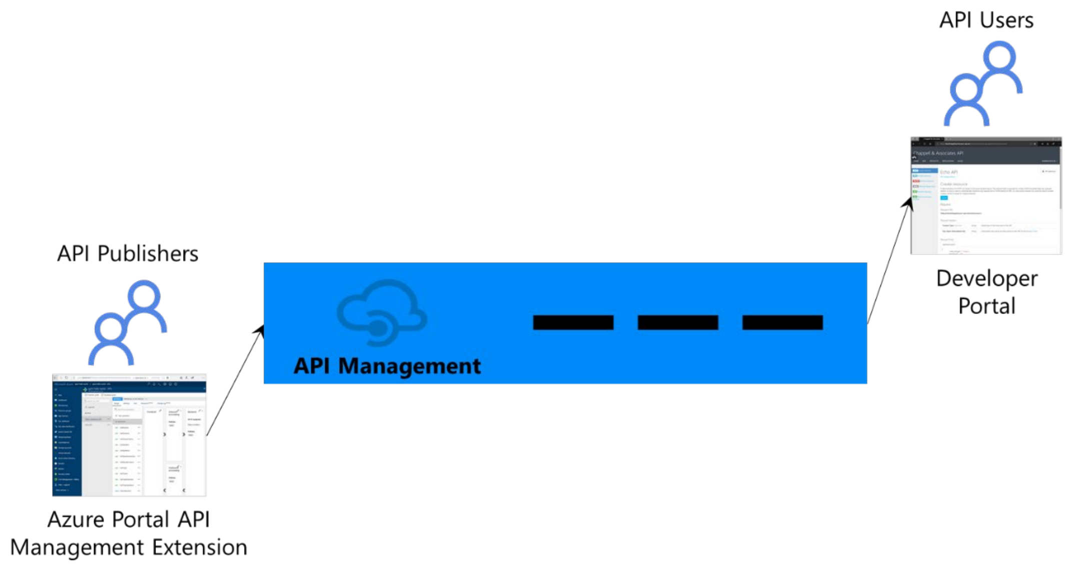


Figure 4: API publishers use the Azure Portal API Management extension to make their APIs available, while API users rely on the Developer Portal to learn how to access an API.

To make a backend API available to clients, the API's owner can use the API Management extension in the Azure Portal. This extension lets you specify the necessary details of the API, including the operations it contains and the parameters for each one. API Management then creates a façade for the API. When a call is made on this façade, API Management routes it to the associated backend application. Each call's result is returned the same way. (The service also provides an API that lets these things be done programmatically.)

But the API Management extension does more than just publish APIs. It provides analytics, for instance, letting you track the usage and the health of your published APIs. It also lets you set policies that control various aspects of how each API functions. There are dozens of policies; here are a few examples of what they allow:

- Restricting how many calls can come from a single source in a defined period.
- Specifying how a caller should authenticate itself.
- Blocking calls from specific IP addresses.
- Turning data caching on and off.
- Converting requests from SOAP to REST.
- Converting data from XML to JSON and vice-versa.

API Management also provides a Developer Portal, as Figure 4 shows. People who want to use an API, typically developers, can use this portal to learn how to do this and to request access to that API. Along with documentation describing the details—the API's operations and parameters—this portal also provides sample code for calling the API from C#, Java, JavaScript, and other languages.

In the modern world, pretty much everything is—or probably should be—exposed as an API. Doing this improves business agility and unlocks business value. Because it solves the problems that arise in exposing APIs, API Management gives you an effective way to achieve this goal.

Logic Apps

Integrating applications commonly requires implementing all or part of a business process. Think, for instance, about a process that accesses a cloud application such as Salesforce CRM, updates on-premises data stored in SQL Server and Oracle databases, and invokes operations in an on-premises application. Doing this means building custom logic that carries out the steps in this process.

But how should this logic be created? One option is to build it in the traditional way using C#, JavaScript, Java, or some other programming language. Yet a process that relies on several different applications might take a while to complete, and it might need to pause between steps. For scenarios like this, creating the logic using a workflow technology is often a better approach.

This is exactly what Logic Apps allows. Each logic app is a workflow that implements some process. This might be a system-to-system process, such as connecting two or more applications. Alternatively, it might be a user-to-system process, one that connects people with software and potentially has long delays. Logic Apps is designed to support either of these scenarios.

To implement a process, a logic app can access all kinds of other applications, including cloud applications, on-premises applications, and Azure Services. Figure 5 shows how this looks.

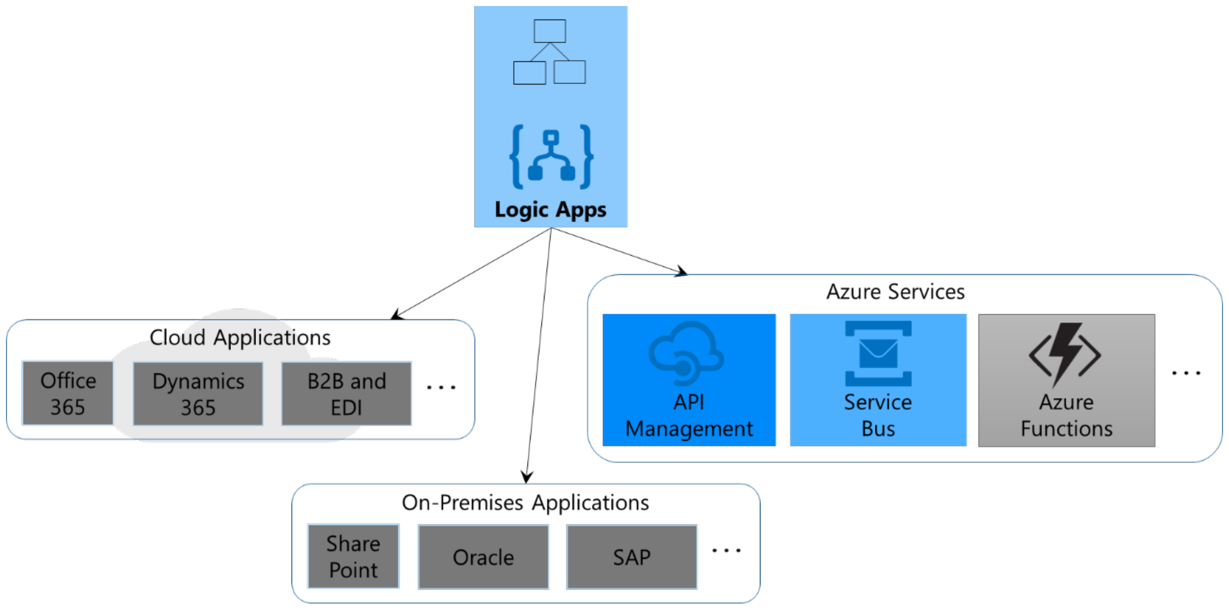


Figure 5: Logic Apps can access many kinds of software running in many different places.

A logic app consists of a series of actions, each of which is a logical step in the process this workflow implements. These actions can express conditionals (if statements), loops, and more. Critically, a logic app can also use actions to call external software and services.

One way to do this is by invoking APIs exposed through API Management. (In fact, a logic app itself can be exposed as an API in this fashion.) But to make it easier to work with common applications, data sources, and other services, Logic Apps provides more than 200 connectors, each providing a straightforward way to interact with a specific external service. There are connectors for interacting with Office 365, for example, along with Dynamics 365, Salesforce CRM, and many other cloud applications. There are also connectors for on-premises technologies such as SharePoint, SQL Server, Oracle, and SAP, along with links to standard protocols such as SMTP and FTP. Other connectors allow access to common integration functions such as data conversions, XSLT transformations, and EDI. Whatever their function, developers can use these connectors to create logic apps for even complex integrations quite quickly. And Microsoft also provides connectors to cutting edge technologies, including Azure Machine Learning and Cognitive Services, allowing logic apps to take advantage of modern approaches to building intelligent integrations, such as providing insights into the data being transmitted.

But how should logic apps be created? Under the covers, each one is expressed in JavaScript Object Notation (JSON). This helps logic apps fit well into existing DevOps environments—each one is just a text file—and it's even possible to create a new workflow directly in JSON. There's no need to do this, however. In fact, creating a logic app typically requires writing no code at all. Instead, developers use the Logic Apps Designer, a visual tool, to create new processes. Figure 6 shows an example of this tool's graphical interface.

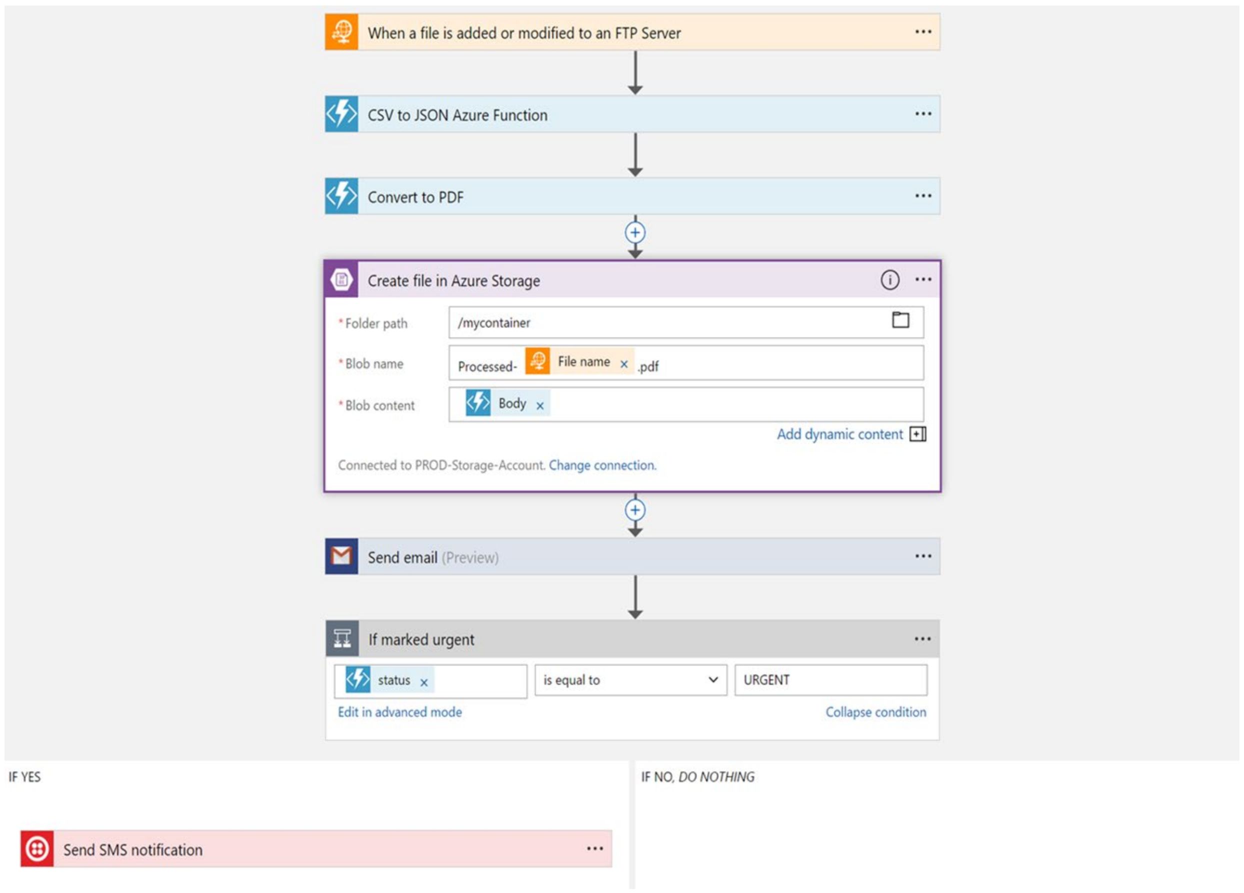


Figure 6: The Logic App Designer lets developers create logic apps without writing code.

As this example shows, the Logic App Designer lets its users create workflows by arranging actions on a surface. Like all logic apps, this one starts with a trigger, which in this case indicates that the workflow should run whenever a file is added to or changed on an FTP server. It then invokes two Azure functions, serverless chunks of code that transform the data in this file: first to JSON, then to PDF. In this example, the logic app's creator is in the process of defining the next action, which creates a file in Azure Blob Storage. After this, the workflow sends an email, then executes a conditional: If the newly added file is urgent, the logic app also sends a text.

This simple example doesn't access any cloud or on-premises applications—it mostly uses Azure services. Still, it illustrates the basics of how a developer creates a logic app. The key point is that writing code isn't required, letting developers implement business processes quickly and easily.

Logic Apps are at the heart of Azure's iPaaS offering, letting you easily connect the software and services you need to access. And because it allows you to integrate disparate services running both in the cloud and on-premises, Logic Apps provides an effective workflow solution for the world today.

Serverless integration

The rise of serverless computing is among today's most important innovations. Serverless technologies, such as Azure Functions, let developers focus entirely on writing code. All the computing infrastructure they rely on—virtual machines (VMs), support for scalability, and more—is managed for them. Because of this, creating applications gets faster and easier. And running those applications often gets cheaper, because you're charged only for the compute resources your code actually uses.

Logic Apps is also a serverless technology. While every logic app ultimately runs in some VM, all of this is hidden from developers. They don't need to think about infrastructure or scalability—it's all handled for them. This frees them to focus solely on creating integration logic. Logic Apps also provides serverless-style pricing, minimizing the cost of running integration logic.

Serverless technology first appeared in what's known as *Application Platform as a Service (aPaaS)* technologies. With Logic Apps, Azure Integration Services brings this innovation to iPaaS as well.

Service Bus

The essence of application integration is software talking to other software. But how should this communication happen? Sometimes, a direct call via API Management is perfect. In other cases, though, this synchronous style of communication won't work. What if both applications aren't available at the same time, for instance? For situations like this, an asynchronous approach is required.

This kind of communication is exactly what Service Bus provides. Because it lets applications exchange messages through queues, Service Bus allows non-blocking interactions between different chunks of software. Figure 7 illustrates this idea.

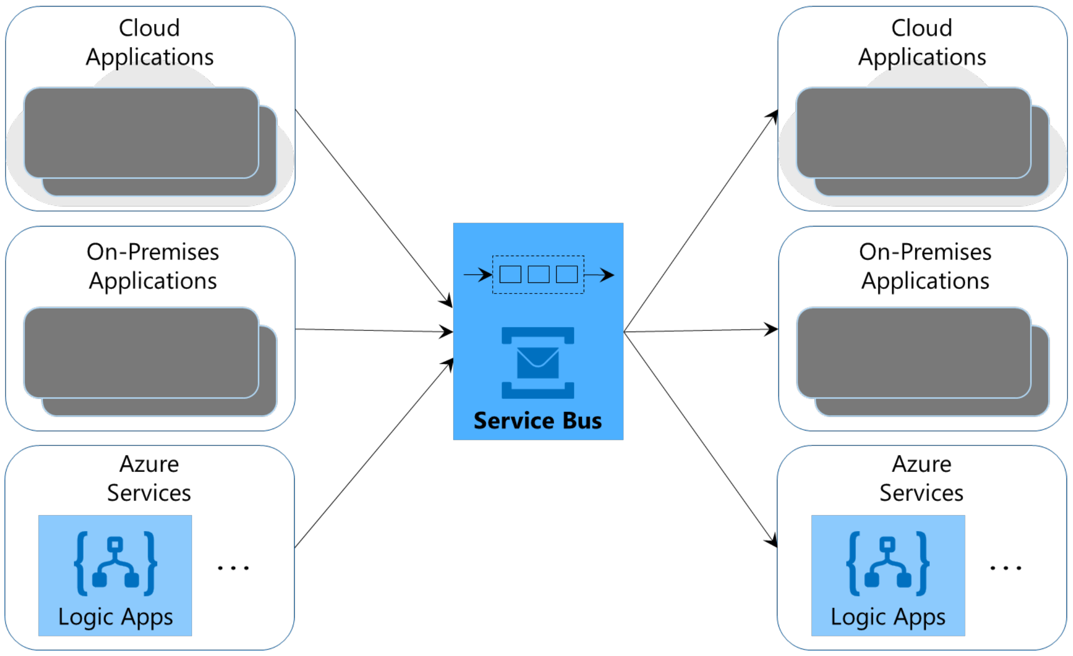


Figure 7: Service Bus provides asynchronous communication between all kinds of software.

As the figure suggests, Service Bus provides enterprise messaging among many kinds of software, including cloud applications, on-premises applications, and Azure services. Doing this well requires providing a diverse set of features—it's harder than you might think. Accordingly, Service Bus provides all the following:

- Queue semantics, including message persistence and strict first-in, first-out ordering of messages. The service also detects and deletes duplicate messages.
- Atomic transactions, letting a queue read or write be part of a larger operation that succeeds or fails as a single unit.
- Poison message handling, so a message that causes problems won't put a receiver into an ongoing loop.
- High availability, including geo-replication, along with built-in disaster recovery.

In short, Service Bus provides all the features required for enterprise messaging.

This Azure service also provides more, however. For example, Service Bus lets a user create one or more topics, then send messages to this topic. Receivers can subscribe to specific topics, then get only the messages sent to that topic. Figure 8 shows how this looks.

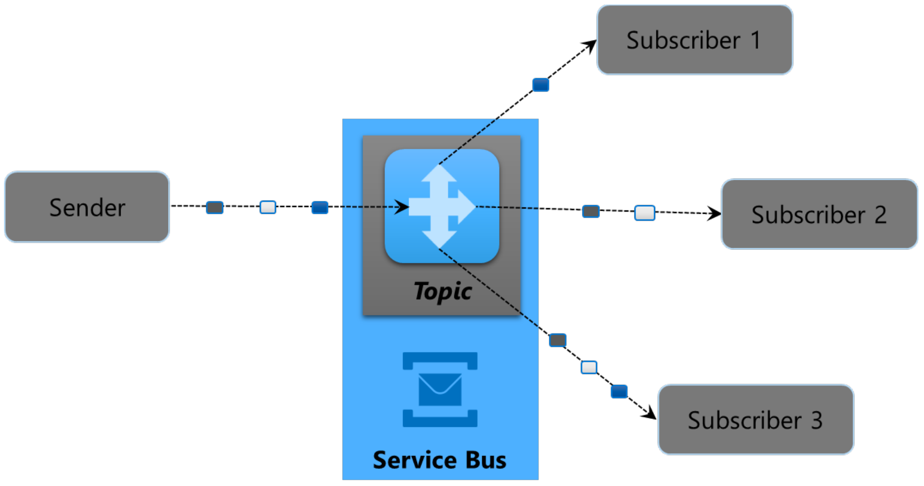


Figure 8: Service Bus lets subscribers selectively receive messages sent to a topic.

As the figure shows, receivers aren't required to get every message sent to a particular topic. Instead, each receiver can define filters that determine exactly which message subset they receive. For example, a receiver that subscribed to a topic with messages carrying financial transactions might elect to receive only those messages with an amount of 10,000 Euros or more.

Asynchronous, message-based communication is an essential aspect of enterprise integration. Service Bus provides this and more as a cloud service on Azure. It's a fundamental component of Azure Integration Services.

Event Grid

There are lots of integration scenarios in which communication through messages rather than API calls is the best approach. But requiring receiving software to periodically check whether a new message has arrived—commonly known as polling—can be wasteful. Why not let a receiver be notified via an event instead?

This is exactly what Event Grid allows. Rather than requiring a receiver to poll for new messages, the receiver instead registers an event handler for the event source it's interested in. Event Grid then invokes that event handler when the specified event occurs. Figure 9 illustrates this idea.

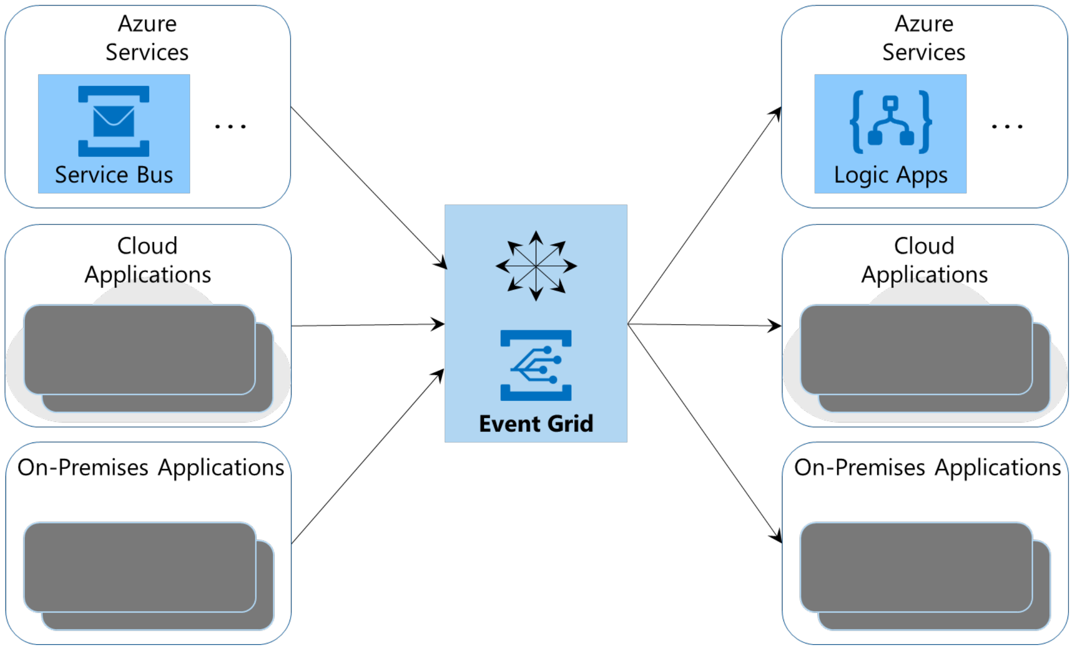


Figure 9: Event Grid invokes receivers when a particular event has occurred.

As the figure suggests, many Azure services can generate events. For example, the arrival of a new Service Bus message might cause Event Grid to send a message that starts a Logic App running, or the creation of a new blob in Azure Blob Storage might cause a custom cloud application to begin processing the contents of that blob. Using an event-driven approach can simplify application development, and it can also save money, since the receiver needn't waste cycles polling for new messages.

To receive an event from an Azure service, such as Blob Storage, the receiver subscribes to a standard topic provided for that service. But while subscribing to events from Azure services is probably today's most common use of Event Grid, it's not the only option. Cloud and on-premises applications can also create custom topics, letting Azure services and other software receive custom events by subscribing to these topics.

All of this raises an obvious question: Isn't Event Grid awfully similar to topics in Service Bus? Why does Azure Integration Services include both? The primary reason is that Service Bus is an enterprise messaging system, with all of the reliability and features that implies, while Event Grid provides only a simple and fast way to send events. Some of the differences that flow from these divergent goals are these:

- Service Bus topics work with messages, not events. They let receivers decide when to read a message, and they require the receiver to actively poll for new messages. With Event Grid, events are raised by Azure Blob Storage or something else, then delivered to subscribers. No polling is required to receive these lightweight events.

- Event Grid is significantly more scalable than Service Bus, supporting up to 10,000,000 events per second in a single Azure region. To achieve this, Event Grid might deliver events out of order, while Service Bus guarantees in-order message delivery.

- Service Bus is fast, but Event Grid provides near real-time performance, with 99% of events delivered in less than a second.

Asynchronous communication is a necessary part of an iPaaS. By providing both Event Grid and Service Bus, Azure Integration Services addresses whatever communication needs your scenario might require.

Data integration: The role of Azure Data Factory

While Azure Integration Services certainly can work with data, its focus is on application integration. What if you need to do data integration? You might like to implement a traditional extract/transform/load (ETL) process in the cloud with Azure SQL Data Warehouse, for instance, or load and process unstructured big data with Azure Data Lake Storage.

Azure Integration Services isn't necessarily the best choice for situations like these. Instead, Microsoft provides Azure Data Factory, a cloud service for data integration. This service addresses both the scenarios just described, along with many more.

Different kinds of integration require different integration technologies. This is why Microsoft offers both Azure Data Factory and Azure Integration Services.

Combining the services: A scenario

Every component of Azure Integration Services has value on its own. Yet the real value of these technologies becomes clear when all four are used together. Figure 10 shows an example of how you might do this.

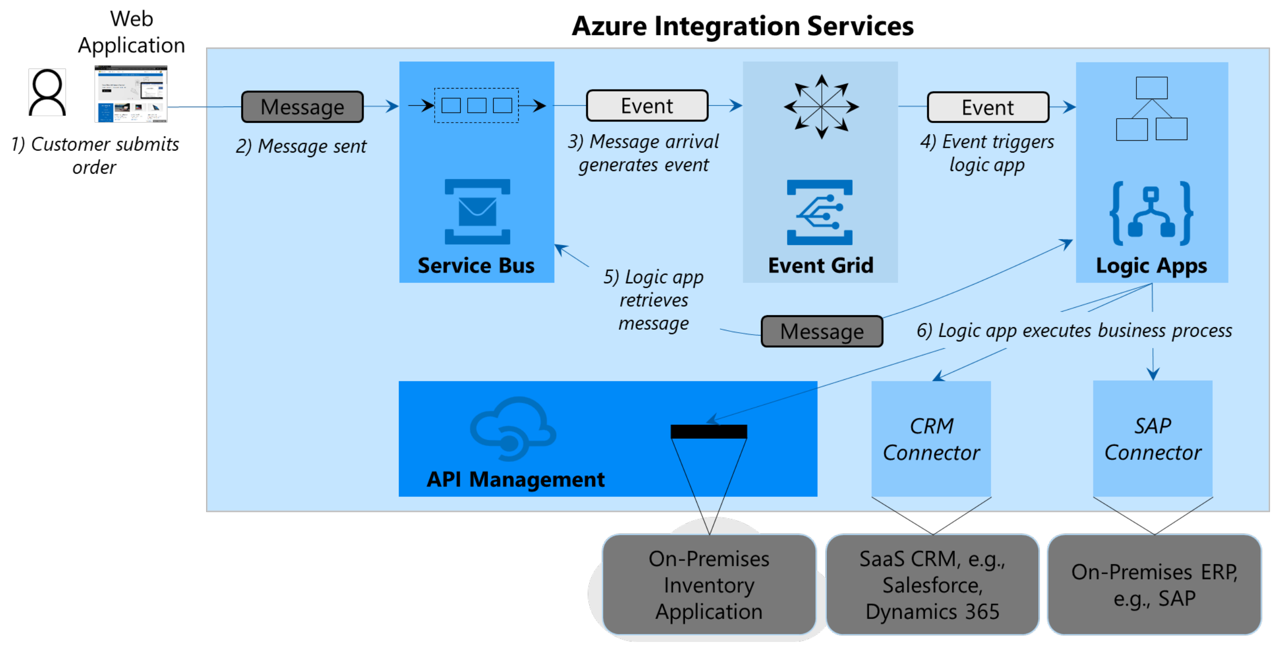

Web Application
Azure Integration Services
1) Customer submits order
Message
2) Message sent
Service Bus
Event
3) Message arrival generates event
Event Grid
Event
4) Event triggers logic app
Logic Apps
5) Logic app retrieves message
Message
6) Logic app executes business process
API Management
CRM Connector
SAP Connector
On-Premises Inventory Application
SaaS CRM, e.g., Salesforce, Dynamics 365
On-Premises ERP, e.g., SAP

Figure 10: The technologies in Azure Integration Services can be used together to provide a complete integration solution.

The scenario begins when a customer submits an order through a web application (step 1). This application sends a message to Service Bus describing the new order (step 2). This message might contain a JSON document identifying the customer, the item, and the quantity ordered. When this message arrives in a Service Bus queue, an event is generated for Event Grid (step 3). This event then triggers execution of a logic app (step 4), which immediately reads the message waiting in Service Bus (step 5). The logic app extracts the information in this message, then executes a business process by accessing the necessary applications (step 6).

In this example, that business process has three parts, each carried out by a different application. They are the following:

- The first requirement is to check whether the item and quantity ordered are available. To do this, the logic app accesses a custom on-premises inventory application. Because this application is accessible via API Management, doing this is straightforward.

- If the requested item and quantity are available, the process then ensures the customer is in good standing and retrieves the customer's mailing address. To do this, the logic app calls this organization's CRM system. Here, CRM is provided by a SaaS application, such as Salesforce or Dynamics 365, but this could also be an on-premises CRM system. To access this application, the logic app uses the appropriate connector for the CRM solution. (It's worth nothing that connectors rely on API Management under the covers, so that aspect of Azure Integration Services is still being used here.)

- If the first two conditions are fulfilled, the logic app places the order by accessing this organization’s ERP system, an on-premises SAP application. This interaction relies on the SAP Connector.

Although it’s not shown in Figure 10, the logic app could also do more. It might invoke an email system, for instance, once again through API Management, to send the customer mail acknowledging the order. API Management could also be placed between the web application and Service Bus in step 2, an option that would simplify the application’s use of this messaging service.

It’s important to understand that because Azure Integration Services runs on Azure, its components intrinsically share many things. They’re all accessed through the Azure Portal, for example, and all can be automated using PowerShell. They share a common deployment mechanism, along with monitoring and management services: Azure Monitor, Application Insights, and Log Analytics. These four components also use the same identity system and the same role-based security model, making security simpler to achieve. It’s even possible to create custom dashboards to monitor complete integration solutions as a unit. You could, for example, create a dashboard designed solely to keep you informed about the various components in the scenario shown in Figure 10.

The key point is that all four parts of Azure Integration Services can work together to provide a complete enterprise integration solution. And while these services run in the public cloud, the applications they knit together can run on-premises, on Azure, or somewhere else.

Conclusion

A modern enterprise integration solution—an iPaaS—provides a group of cloud technologies that work together. With Azure Integration Services, those technologies are:

- API Management for publishing and managing application APIs.
- Logic Apps for graphically creating workflows.
- Service Bus for enterprise messaging.
- Event Grid for event-based communication.

Each of these cloud services has value on its own. Together, they provide a full-featured iPaaS capable of connecting both cloud and on-premises software. Designed for mission-critical enterprise integration, Azure Integration Services is an iPaaS for the modern hybrid world.

For more information

Home page: Azure Integration Services

Video: Azure Essentials – Integrating your Apps with Azure

Webinar: Azure Integration Services

White paper: Driving Digital Transformation in Today's API Economy

Gartner April 2018 Magic Quadrant: Enterprise Integration Platform as a Service