David Chappell

Data in a PaaS World

A Guide for New Applications

Sponsored by Microsoft Corporation

Contents

The Rise of PaaS Data Services 3

The Value of PaaS for Applications and Data 3

How PaaS Data Services Enable Polyglot Persistence 4

Azure PaaS Data Services: An Overview 5

Operational Data Services 6

SQL Database 6

DocumentDB 7

HDInsight HBase 7

Tables 8

Blobs 9

Other Data Services 10

Azure Search 10

Azure Redis Cache 11

Scenarios 13

Creating an Employee-Facing Application 13

Creating a Customer-Facing Application 14

Creating a Cloud Backend for a Mobile Application 15

Creating a Cloud Application with a Microservices Architecture 15

Creating a Multi-Tenant SaaS Application 17

Creating an Integration Application 17

Creating an Analytics Application 18

Conclusion 19

About the Author 20

The Rise of PaaS Data Services

Cloud computing has changed how we build applications. One of the most important aspects of that change is the advent of Platform as a Service (PaaS). Unlike Infrastructure as a Service (IaaS), where you create and manage your own virtual machines, PaaS provides a higher-level foundation that hides irrelevant details. Building on PaaS lets development teams focus on what they really care about—their application—rather than on managing infrastructure.

The idea of PaaS first became popular for compute. On Azure, for example, Cloud Services provided a PaaS foundation for application software right from the beginning. Today, other PaaS compute technologies are also available on Azure, including App Service and Service Fabric.

But the idea of PaaS isn't restricted solely to compute. Many other aspects of application development can also benefit from an approach that hides the underlying infrastructure. For example, rather than use a database management system (DBMS) running in an IaaS VM, an application can instead rely on PaaS data services provided by a cloud platform. Even though this kind of managed data service has been available for several years, the term "PaaS" has been applied largely to compute services. It's time to change this. PaaS data services are just as important.

The Value of PaaS for Applications and Data

To make the idea of PaaS compute and PaaS data clear, it's useful to contrast the IaaS and PaaS approaches to building applications. Figure 1 illustrates each one.

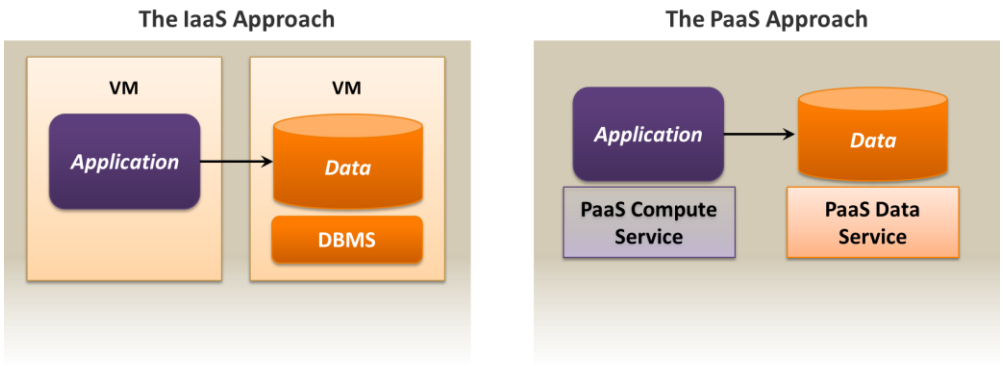

Figure 1: Cloud applications can use an IaaS approach or a PaaS approach for compute and data.

With IaaS, applications run in VMs that are commonly created and managed by the development team. Other VMs run a DBMS such as SQL Server, MySQL, or MongoDB. In essence, the IaaS approach takes the way we've built applications and worked with data in our own datacenters and moves it unchanged into the cloud. And if what you're trying to do is move existing on-premises applications to Azure, this can be the best approach.

But why do this for new applications? The cloud is a new world, so there's no need to mindlessly replicate the environment it replaces. By providing a managed environment, Azure and other cloud platforms can make life significantly better for people who create and run applications. As the PaaS approach in Figure 1 shows, developers can now just create application code on top of a PaaS compute service, then use a PaaS data service for the

application's data. Rather than copying the on-premises world, the PaaS approach rethinks how we build applications and work with data for the cloud.

PaaS data services (which are also sometimes referred to as *Database as a Service*) have a number of advantages over the IaaS approach of running a DBMS in an IaaS VM. Among the most important are the following:

- *A PaaS data service is fully managed.* The people who use the service don't need to install, patch, or manage a DBMS, which saves time and money.
- *A PaaS data service can provide built-in scalability.* Rather than require you to add database servers manually as your application's load increases, a PaaS data service can do this on demand with no downtime.
- *A PaaS data service can provide built-in reliability and fault tolerance*. Rather than requiring specialized skills and plenty of time to set up a clustered DBMS, a PaaS data service can do this for you.
- *A PaaS data service offers service level agreements (SLAs)*. Rather than worrying about availability yourself, you can rely on the cloud provider to meet these or face financial penalties.
- *A PaaS data service can cost you less.* Unlike the IaaS approach, where pricing is typically based on the number of VMs you run, PaaS data services are typically priced in terms of the things you actually care about, such as database performance and the amount of data stored. This lets you control your usage—and the amount you spend—in a more fine-grained way.

Nothing is free, however; adopting PaaS data services does limit your control over the underlying database software. Still, in practice, applications don't often need fine-grained control over the DBMS they're using. And using PaaS data services also brings another big benefit: It makes polyglot persistence much easier to do.

How PaaS Data Services Enable Polyglot Persistence

How many different data stores should an application use? Ideally, the answer should depend on the data the application is working with. An application that uses only relational tables can get by with just a relational store. One that uses multiple kinds of data, however, such as relational tables, JavaScript Object Notation (JSON) documents, and large binary data such as videos, could probably benefit from using different data stores for different data. Rather than cramming a multi-gigabyte video into a relational table, for example, why not keep it in a store designed for binary data? This approach is often referred to as *polyglot persistence*, and Figure 2 shows an example.

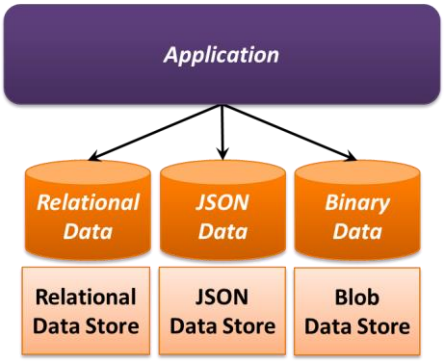

Figure 2: With polyglot persistence, an application uses different kinds of data stores for different types of data.

While polyglot persistence makes intuitive sense, it's not especially common today for on-premises applications. The biggest barrier is that buying, installing, and managing different data stores for different kinds of data can be difficult and expensive to do in your own datacenter. Because of this, on-premises applications (as well as cloud applications that use the IaaS approach) seldom do polyglot persistence. Instead, developers often use a single store for all of their data, regardless of what that data looks like.

But with PaaS data services, these problems fall away. Because the data services are managed for you, you no longer need to worry about installing or running separate data stores. You do still need to understand how to use these different options, so the value of using diverse data stores must outweigh this added complexity. Still, the extra overhead required to do polyglot persistence shrinks significantly with PaaS data services. This useful idea becomes much more practical.

The cloud also brings another reason to favor the polyglot approach: cost. As mentioned earlier, cloud platforms charge based on usage. A PaaS data service might levy a monthly fee per gigabyte of stored data, for instance, or perhaps charge based on how frequently data is accessed. However it's done, different data services can have quite different pricing models, and so taking a polyglot approach to these services can be an effective way to manage costs. Why pay for full relational semantics just to store fifty gigabytes of video?

Whether you're looking for polyglot persistence or just want a faster, easier, and cheaper way to build your application, PaaS data services make sense. In fact, an important aspect of choosing a cloud platform is making sure that it has the PaaS data services your application needs. To address these requirements, Azure provides a range of these services, as described next.

Azure PaaS Data Services: An Overview

While Azure today supports both IaaS and PaaS approaches to working with data, this cloud platform included a PaaS data service from the beginning. Today, that initial service has been joined by several more. What follows takes a high-level look at these services. The focus is on PaaS data services used for operational data, that is, read/write data an application uses in the course of its normal operation.

Operational Data Services

To make the best choice for your application and data, you need to have a big-picture sense of Azure's PaaS options. Figure 3 summarizes the PaaS compute and the PaaS data services that Azure provides.

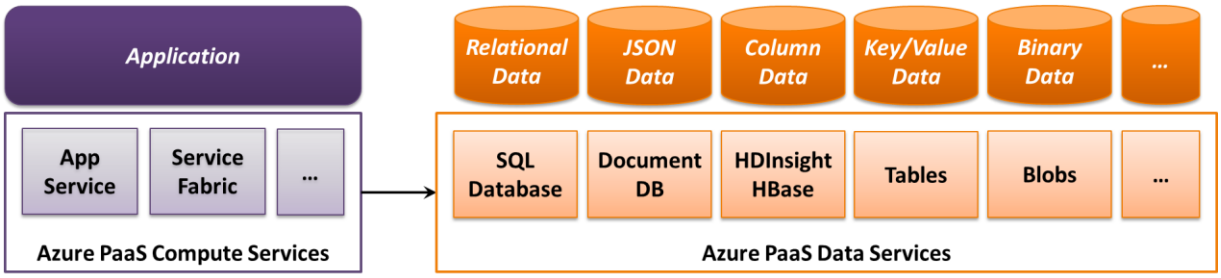

Figure 3: Azure provides a variety of PaaS compute services and PaaS data services.

For PaaS compute, Azure provides App Service and Service Fabric. It also supports Cloud Services, the platform's original PaaS compute technology. For PaaS data, Azure provides a variety of services for working with operational data. The rest of this section provides a brief introduction to each one.

SQL Database

Relational databases have been popular for decades, and they'll probably be popular for decades to come. To support a PaaS approach to working with relational data, Azure provides SQL Database. Figure 4 illustrates the basics of this technology.

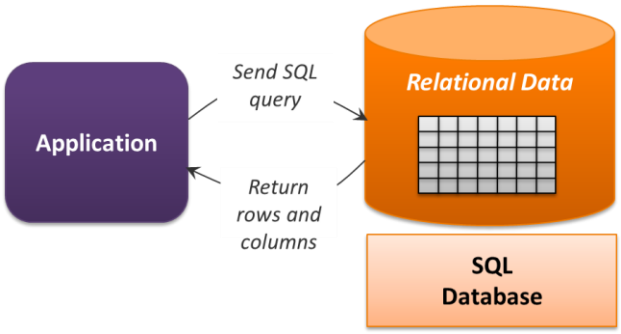

Figure 4: SQL Database works with traditional relational data.

Based on SQL Server, this service offers a familiar relational store, including support for SQL queries, transactions across an entire database, stored procedures, and more. It also provides built-in fault tolerance and scalability. Given the range of useful services that SQL Database provides, it's typically the most expensive of Azure's PaaS data services (although there are also low-cost options for smaller applications).

SQL Database is a good choice when an application needs the full power of a relational system. It's also a good choice when the development team is already familiar with SQL and relational technologies. Since this PaaS data service is based on SQL Server, learning it commonly isn't difficult.

Like any relational DBMS, however, SQL Database can be challenging to use with data whose structure changes frequently. Relational systems rely on schemas, and while SQL Database certainly does allow these schemas to change, applications built to expect a fixed structure for data commonly need to be updated for each schema change. This can sometimes be impractical. Also, other Azure PaaS data services can be significantly less expensive for storing some kinds of data, such as large binary files.

DocumentDB

In part due to the increasing popularity of JavaScript, JSON has become a de facto serialization format for many kinds of data. Given this, why not create a PaaS data service designed for storing and working with JSON at scale? This is exactly what DocumentDB is, as Figure 5 illustrates.

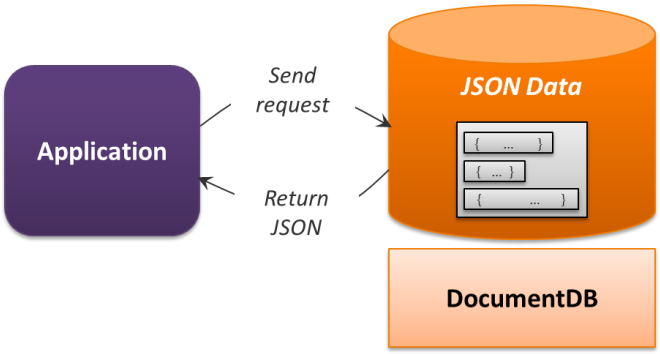

Figure 5: DocumentDB works with JSON data stored in documents.

As its name suggests, DocumentDB stores *documents*, each containing JSON data. This PaaS data service allows RESTful access to the documents it contains, and it also lets applications issue queries using an extended subset of SQL. And like SQL Database, DocumentDB provides transactions, built-in scalability, and built-in high availability.

DocumentDB is an attractive choice for developers working in JavaScript and other modern languages, all of which provide built-in JSON serializers. It's also good for situations where the data's structure changes frequently, because unlike a relational database, DocumentDB doesn't define a fixed schema. Different documents are free to contain whatever JSON the application requires. Also, this PaaS data service automatically indexes all of the data it stores, making it faster for applications to find what they need.

Yet because DocumentDB isn't a relational database, it can be less familiar to many developers—it takes time to learn (although it does provide a SQL-based query language). This PaaS data service also has some limitations, such as the inability to do joins across different collections of documents. And while it's commonly less expensive than SQL Database, working with large amounts of data in DocumentDB is still likely to cost more than with some PaaS services, such as Tables and Blobs.

HDInsight HBase

HBase is part of the Hadoop technology family, and so it's designed for processing big data. The approach it takes is vaguely reminiscent of a relational system, with data stored in tables. Don't be confused, however; those tables aren't relational. Figure 6 illustrates the idea.

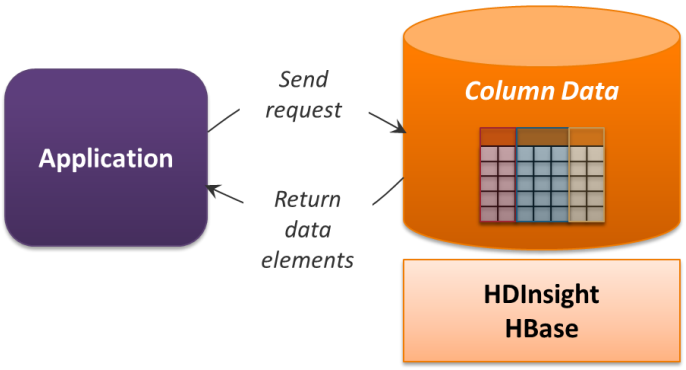

Figure 6: HDInsight works with data stored in column families.

HBase can be viewed as a *column-family store*. As Figure 6 suggests, the columns in each table are grouped into families, and requests for data can specify which column family to look in. Unlike relational tables, however, HBase allows adding a new column to a column family at runtime—the schema isn't fixed. It's also designed to be very scalable, letting applications create tables with millions of columns and billions of rows. And while HBase itself provides only simple query capabilities, HDInsight HBase also supports the open source Apache Phoenix technology for issuing SQL-like queries against an HBase table.

HDInsight HBase is an excellent choice for applications that need to create big but sparse tables. It's also a good option when the data it stores will be processed with Hive or another member of the Hadoop family, since all of these rely on the same underlying HDInsight clustering technology. In addition, HBase can be less expensive than the full relational functionality provided by SQL Database.

Like DocumentDB, however, HBase is a NoSQL technology, and so it's unfamiliar to many developers—it has a learning curve. It also has limited support for transactions, forcing developers to solve some kind of problems in new ways. And because HBase is part of HDInsight, using it requires setting up (and paying for) a cluster. Doing this takes more work than just calling the APIs provided by the other Azure PaaS data services described here.

Tables

Some situations need the full power of a relational database, including SQL queries against relational tables. Others, however, can get by with a much simpler approach to storing and accessing data. For situations like these, Azure provides Tables, illustrated in Figure 7.

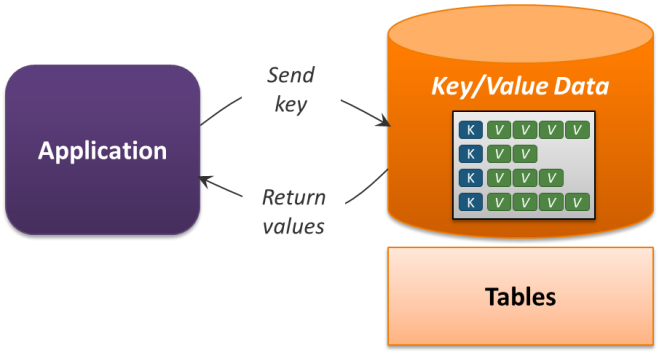

Figure 7: Tables works with data structured into groups of values, each with a unique key.

Despite the name of this PaaS data service, it doesn't really store data in tables. Instead, an application accesses data by providing a unique key. This service then returns whatever set of values are associated with that key.

This simple approach works well in a surprising number of situations. Think about storing user profile data, for example. Each user can have a unique key, with each key providing access to whatever profile data is stored for that user. Different users can have different data—there's no fixed schema—and so Tables provides a quite flexible approach. It's also very inexpensive, occupying the bottom rung on Azure's pricing ladder.

Tables provides only very simple queries, however; applications most often need to know what key to provide to access a desired set of values. Transaction support is also quite limited, another big differentiator from SQL Database and DocumentDB. And data access times can vary considerably from one request to another, a factor that can be important for some applications. Still, Tables can be the right choice, especially for applications that work with large amounts of simply structured data.

Blobs

All of the PaaS data services described so far have provided some kind of structure for data: relational tables, JSON documents, or something else. Sometimes, though, applications need to store just unstructured binary data. Common examples of this include videos, images, and PDF documents. For situations like this, the best choice is often Blobs. Azure Blobs offer what's sometimes called *object storage*, and Figure 8 shows the basics of this PaaS data service.

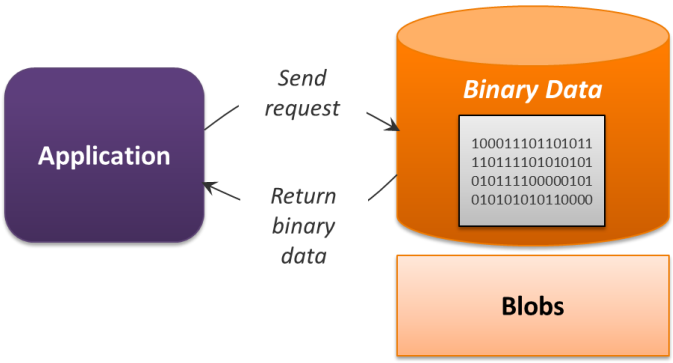

Figure 8: Blobs works with simple binary data.

Blobs are very inexpensive. Along with Tables, this service sits on the bottom rung of Azure's data services pricing ladder. The service is also quite scalable, capable of storing large chunks of data. Yet the service Blobs provides is simple, with no queries and no support for transactions. Still, Azure applications that take a polyglot approach to persistence frequently use Blobs, because the need to store large chunks of unstructured data cheaply is so common.

As is probably clear by now, each of Azure's PaaS data services provides a different set of characteristics. To help you make sense of this diversity, Figure 9 summarizes the fundamentals of each one.

	SQL Database	Document DB	HDInsight HBase	Tables	Blobs
Kind of Data	Relational tables	JSON documents	Column-family tables	Keys and values	Binary
Queries	SQL	SQL-like	SQL-like	Simple	None
Transactions	Yes	Yes	Limited	Limited	No
Typical Relative Cost	Highest	Medium	Medium	Lowest	Lowest

Figure 9: Each of the Azure PaaS data services has unique characteristics.

Other Data Services

Along with the technologies described so far, Azure also provides two other PaaS data services for working with operational data. Neither one addresses quite the same type of problem as the ones already mentioned, but they're both important to understand. Those services are Azure Search and Azure Redis Cache.

Azure Search

For many people, search has become the most attractive way to interact with applications. Rather than choose items from menus, why not let an application's users search for what they're interested in, much as they would on the Internet? Allowing this would make many applications significantly easier to use.

Yet making every development team create its own search engine is asking too much. What's needed is a PaaS service that these teams can use to more easily add search capabilities to their application. The goal of Azure Search is to provide this service.

To help developers give users the experience they expect, Azure Search provides things such as automatic bolding of search terms in results and a way to control the order in which these results are returned. It also supports the ability to provide suggestions, offering possible search phrases based on a user's initial entry. The goal is to make it significantly simpler for development teams to build a powerful search capability into their applications.

Azure Search can also play an important role in polyglot persistence. In the traditional world, using a single PaaS data service for everything often allows creating an index that lets you easily search your data. For example, SQL Database allows creating indexes on specific columns in a table, while DocumentDB automatically indexes everything you put into it. A query can use this index to rapidly find data.

With polyglot persistence, however, this kind of query-level index isn't possible. Given the diversity of Azure's PaaS data services, no single approach would work at this level. What is possible, however, is an application-level search across all of these services and more. This is exactly what Azure Search provides, as Figure 10 shows.

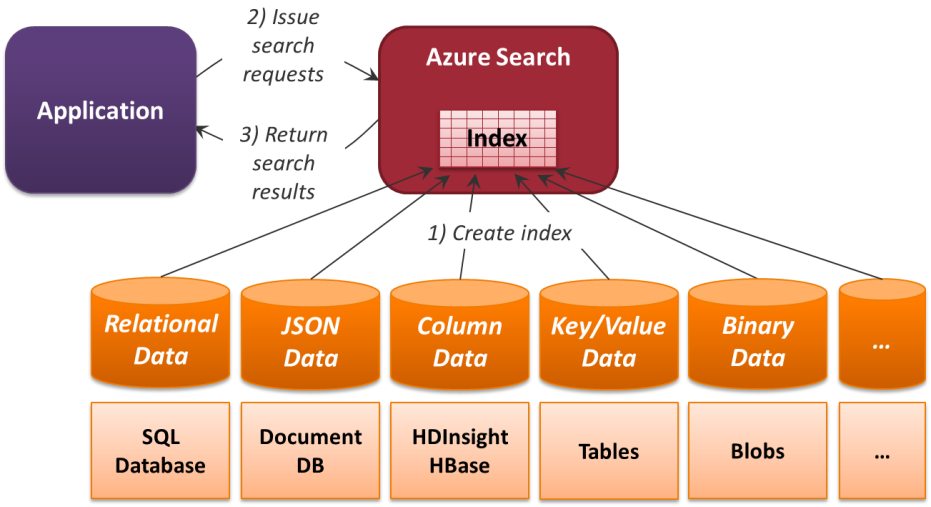

Figure 10: Azure Search can act as an application-level index across diverse data sources.

To use Azure Search requires first creating an index (step 1). The data in this index can come from one or more of Azure's PaaS data services (or from somewhere else—you're not limited to just these services). It's important to realize that an index isn't a primary data store; it just contains easily searchable copies of data stored somewhere else. But once an index exists, an application can issue search requests that use it (step 2).

For example, the application might wish to access all information about a particular customer. By searching for this customer's unique identifier, the application might be able to retrieve that information, regardless of where it's stored (step 3). Using this approach, Azure Search can provide an application-level index across all of Azure's PaaS data services. And because Azure Search can see inside common formats, including Office files and PDFs, it can provide a straightforward way to make even blob data searchable.

Azure Redis Cache

Providing good performance was an important goal for the people who created Azure's PaaS data services. But all of them store data on disk—they're persistence services, after all—which has unavoidable implications for performance. Isn't there a faster way, especially for situations where data is mostly read?

There is: applications can use Azure Redis Cache. Figure 11 illustrates the idea.

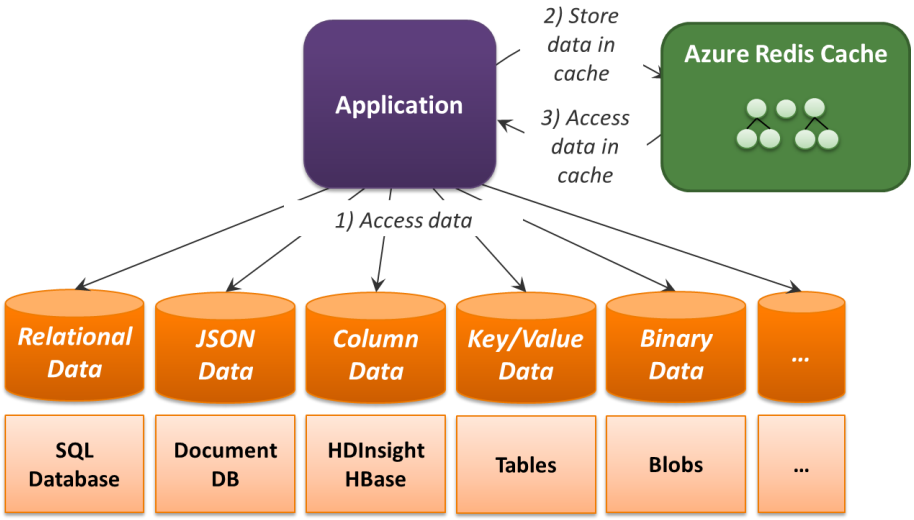

Figure 11: Azure Redis Cache keeps a quickly accessible copy of an application's data in memory.

With Redis Cache, an application can access data from any Azure PaaS data service, as usual (step 1). The application can then store a copy of that or other data in Redis Cache (step 2). When the application needs the data in the future, it can access the copy in this in-memory cache (step 3) rather than going back to the PaaS data service. Doing this can be significantly faster, letting applications have better response time and handle more simultaneous users.

In fact, an application that serves largely read-only data, such as a web site that provides a large amount of readable content, might serve most of that data from a combination of Redis Cache and Azure Search. Both provide accessible copies of data that can be accessed very quickly, yet the application still retains the ability to access the original data as needed in whatever PaaS data service stores that data.

PaaS and Lock-In

One of the traditional concerns developers have had in using PaaS technology—compute or data—is the fear of cloud lock-in. If a PaaS technology is available only in the public cloud, won't using that technology lock the application into that public platform?

With PaaS on Azure, this need not be true. Microsoft is bringing more and more PaaS technologies to on-premises datacenters. With PaaS compute, for example, Service Fabric is part of Windows Server 2016—it will run just as well on your own servers as on Azure. And with the advent of Azure Stack, App Service, Tables, and Blobs will be available for on-premises datacenters (with more services to come).

In the cloud, building new applications using PaaS compute and PaaS data makes sense. As these technologies become available in non-cloud datacenters, expect to see the PaaS approach also get more popular for on-premises applications.

Scenarios

Understanding the basics of Azure's PaaS data services is important, but it's not enough. It's also important to think about how these services can be used in new applications. This section walks through several different scenarios, each looking at how applications built on PaaS compute might use PaaS data services.

Creating an Employee-Facing Application

Most organizations prefer to buy applications rather than build them. With the increasing popularity of Software as a Service (SaaS), doing this has become even easier. Yet many organizations, especially large enterprises, also have a significant number of custom-built applications used by their employees. Sometimes, an organization creates these internally facing applications because it's easier than buying them, as with a simple web solution for requesting vacation time. Often, though, organizations create custom applications because they can't get what they need from a vendor. Think about an enterprise that automates a proprietary internal business process, for instance, creating a custom application for its employees to use. Software like this provides real business value—it might even be one of the main ways a firm differentiates itself from its competitors—and so employee-facing applications can be very important.

For all of the reasons described earlier, it typically makes sense to build these new applications using the PaaS approach. For PaaS compute, a good place to start is App Service, which is designed to let development teams create applications quickly and easily. For data, those applications might use any of Azure's PaaS data services. Figure 12 shows how this looks.

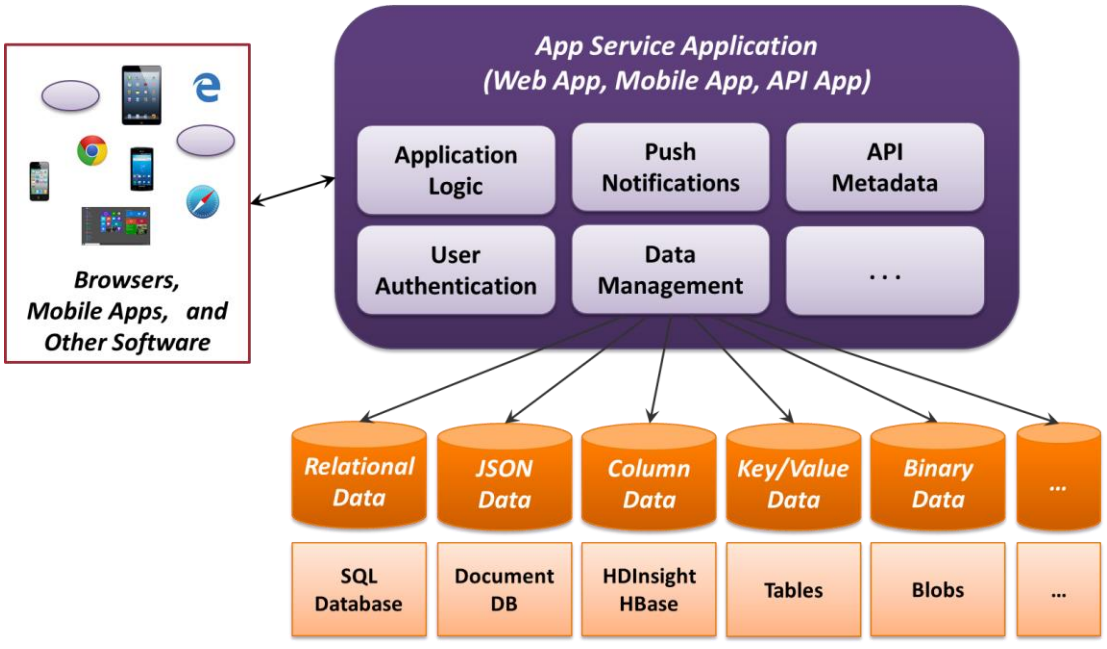

Figure 12: An App Service application can use any of Azure's PaaS data services.

App Service is a collection of capabilities for creating a cloud application. Depending on which ones you use, your application might be referred to in different ways:

- Web Apps have browser clients and use capabilities that include Application Logic (for which App Service provides several options, including .NET, Java, and more), User Authentication, and Data Management.
- Mobile Apps have mobile clients (and despite their name, they run in the cloud.) A Mobile App is a Web App, and so it might use the App Service capabilities for Application Logic, User Authentication, and Data Management. It will also use some of the mobile-centric capabilities, such as Push Notifications.
- API Apps have other software as their clients. An API App is also a Web App, so it can use the basic capabilities just described. It also uses the API Metadata capability, which helps create RESTful APIs that other software can call.

A single application can combine capabilities from all three categories, functioning as a Web App, a Mobile App, and an API App at the same time. And as just described, any of these can use the Data Management capability that App Service provides. This feature provides built-in support for accessing several of Azure's PaaS data services, including SQL Database, DocumentDB, and Tables. It also supports accessing PaaS data services provided by other organizations on Azure, including a MySQL service and a Mongo DB service.

For employee-facing applications, choosing the right PaaS data services depends on what your application needs to do. Here are some things to think about:

- Because SQL Database uses the familiar relational model and provides a broad set of services, it's probably the first option to consider. Massive scale can be a little more work to achieve with SQL Database than with, say, DocumentDB, but internally facing applications don't typically need this—they rarely have tens of thousands of simultaneous users.
- If the structure of the data your application is working with is quite varied, and especially if it will change frequently, consider using DocumentDB. This kind of rapid change isn't common with, say, core banking applications, but it can be with applications that process telemetry data from an ever-changing set of devices or anything that works with diverse user-generated content. And don't forget the influence of fashion. While it's not a particularly good motivation, JSON data stores are in vogue today, so plenty of developers want to learn how to use them.
- Whatever choice you make for the application's main operational data, it's always a good idea to consider using Blobs for binary data. Access is simple, the service offers built-in replication across data centers, and it's cheap.

Finally, regardless of the choice you make for operational PaaS data stores, remember the potential value of Azure Search. Whether you use it to more easily search diverse data stores or as a way to provide a better interface to your users, Search can help you build a better application. And if your application needs fast access to data that's read frequently, using Redis Cache can help.

Creating a Customer-Facing Application

App Service is very likely Azure's best PaaS option for creating an employee-facing application. It can also be a good choice for creating customer-facing applications (although as described later, Service Fabric might also be a good option here.)

Yet customer-facing applications commonly need to be more scalable than those used by an organization's own employees. Because of this, PaaS data services like DocumentDB and HDInsight HBase can be more appealing for customer-facing applications. And if the data stored by the application is relatively simple, Tables can be a scalable and inexpensive choice. For instance, a global consumer electronics manufacturer that wants to create a product registration application for its customers might choose Tables to store this data. And no matter what the application does, Tables are likely to be a good choice for storing performance data and other metrics generated by the application itself.

It's also important to consider other Azure PaaS data services. For example, a web site supporting a major sporting event—the Olympics, the Super Bowl, the World Cup—clearly needs excellent performance under heavy load. Using Redis Cache might be the best way to achieve this, especially with read-mostly data. Applications like this also probably need to give their users a way to search for things like player information and highlight videos. Azure Search makes this straightforward to provide.

Creating a Cloud Backend for a Mobile Application

As described earlier, App Service provides capabilities that help developers who are creating cloud applications connect to mobile clients. But think about the reverse scenario: Suppose you're creating a mobile app for iOS or Android or Windows, and you'd like to use a few cloud services. This is a common situation, and there's a name for cloud technologies that address this: Mobile Backend as a Service (MBaaS). Along with helping cloud applications talk with mobile clients, App Service also supports MBaaS scenarios.

For example, suppose you're creating a mobile app that needs some cloud storage. App Service includes client SDKs to help your app access the Data Management capabilities this PaaS platform provides. It also provides a simple graphical tool for creating a connection to a PaaS data service. Using these, the mobile app can store data in whatever Azure PaaS data service makes the most sense.

To make doing this even more attractive, App Service provides Easy Tables. This option lets a mobile developer graphically define tables in SQL Database, complete with schemas, then use them from a mobile app. Given that mobile developers aren't always deep into cloud software, this simple option argues for choosing SQL Database over the other Azure PaaS data services, just because it's so straightforward to use.

Alternatively, some applications might be a better match for a NoSQL service. Maybe a relational schema is too confining, for example, or perhaps the development team wants to avoid mapping the application's data to relational tables. For cases like this, App Services Mobile Apps provide built-in support for DocumentDB and other NoSQL PaaS data services, although (today, at least) without support for Easy Tables.

Creating a Cloud Application with a Microservices Architecture

The newest of Azure's PaaS compute services is Service Fabric. The most obvious difference between this and Azure's other PaaS compute services is that Service Fabric is designed explicitly for applications created using a microservices architecture.

In Service Fabric, a microservice contains logic and (optionally) state that can be independently created, versioned, deployed, and scaled. Each microservice interacts with other microservices through well-defined interfaces using REST or other protocols, and each one is typically created by a relatively small development team. Creating applications as a group of interacting microservices can help make those applications more scalable, more reliable,

and easier to maintain. This approach can also bring some complexity, however, which is why it's most often used by independent software vendors (ISVs) and more technically sophisticated enterprises, such as financial services firms.

Microservices and polyglot persistence fit together very well. The reason for this is that, in general, the development team building each microservice can choose whatever persistence mechanism is the best fit for them. A single Service Fabric application, for example, might use several different options, as Figure 13 shows.

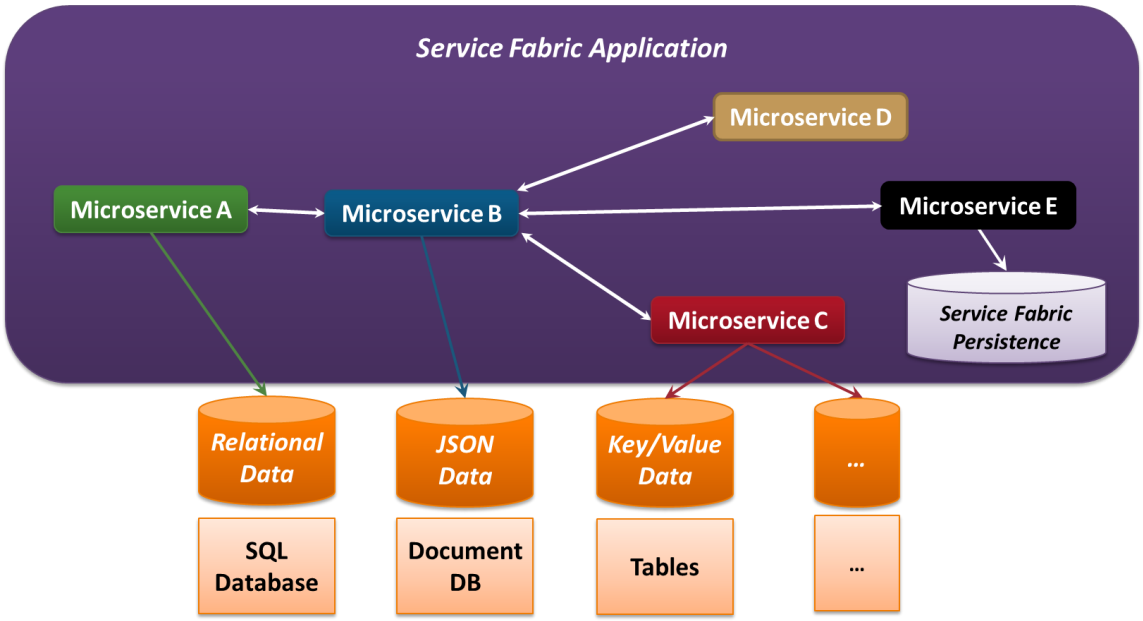

Figure 13: Each microservice in a Service Fabric application can potentially use whatever persistence mechanism its creators think is most appropriate.

Microservices A, B, and C are *stateless*—they write any state that must be maintained between calls from clients to a persistence service. In this example, these three microservices all use different PaaS data services, each chosen by the microservice's creators as the best choice for whatever the microservice is doing. Microservice D is also stateless, but it doesn't need to store any data between calls from its clients. (Perhaps it just does a calculation on a value passed to it by its caller, then returns a result.) Microservice E uses yet another option: it relies on the built-in persistence provided by Service Fabric itself. This microservice is an example of what Service Fabric calls a *stateful* service.

Stateful services represent another persistence option that hasn't been mentioned so far. This option is available only to code written using Service Fabric, and it makes sense when applications need very fast reads. Rather than going to an external PaaS data service to fetch data, as a stateless service does, a stateful service has this data available locally for faster access. That data isn't directly accessible by other applications, however, as it would be if it were stored in a PaaS data service, nor can it be read with SQL or another query language. Still, stateful services can be the right choice for some microservices created using Service Fabric.

Creating a Multi-Tenant SaaS Application

More and more, the people who purchase applications look first for a cloud solution. Rather than buy, install, and manage on-premises software, they'd prefer a SaaS application. And since buyer preferences are shifting, so are the views of ISVs—they're moving en masse toward SaaS.

A SaaS application has many similarities to a customer-facing application created by an enterprise. Both typically must handle large numbers of simultaneous users, for instance, so scalability is important. Both must also be up all the time, so availability is critical.

There are also differences, however. One of the most important is that SaaS applications are commonly multi-tenant. This means that a single application must support many customers, keeping each customer's data safe from access by other customers. If a SaaS application chooses to use relational data, as many business applications do, one option is to run a separate DBMS instance in its own VM for each tenant, i.e., each customer. This is hard to scale, however, and so using a PaaS data service can make more sense.

SQL Database is a good fit for this scenario. By creating a separate database for each customer, an ISV can rely on Microsoft to keep each customer's data secure. This is a simple and potentially attractive solution, but it can get expensive. SQL Database pricing is based on the throughput available for a database, not on how much data it stores. This means that having a high-throughput database available for every customer of a multi-tenant application can get pricey. The ISV would need to pay for the maximum throughput defined for each database regardless of the actual load the customer is currently placing on that database.

To help control an ISV's costs, SQL Database provides *elastic database pools*. This option lets a fixed amount of throughput be shared across a group of databases. Doing this lets an ISV avoid paying for many high-throughput databases when in fact the variations in customer load don't require this. Using this option can be an important part of creating cost-effective multi-tenant SaaS applications with SQL Database.

Still, some SaaS applications, especially those that must work with very large amounts of data, will likely be better off with a NoSQL solution such as DocumentDB or HBase. And whatever the creators of a SaaS application choose for customer data, the application might also use other PaaS data services. It could use lower-cost Tables to keep track of performance and other logging information, Blobs for binary data, and more. Polyglot persistence can be useful here, too, and so as always, it's worth thinking about which PaaS data services are the best fit for a particular kind of data.

Creating an Integration Application

Most applications need to talk with other applications. Providing this kind of integration presents some unique challenges, however, and so integration applications aren't exactly like other applications. Because of this, App Service provides specialized capabilities for integration with what it calls Logic Apps.

Logic Apps are built on workflows. A workflow is a way of creating logic out of discrete steps, with the potential for pauses (sometimes long pauses, like a few days) between these steps. Like other apps, Logic Apps commonly need to work with persistent data, and so they might use PaaS data services. Figure 14 shows an example Logic App that does this.

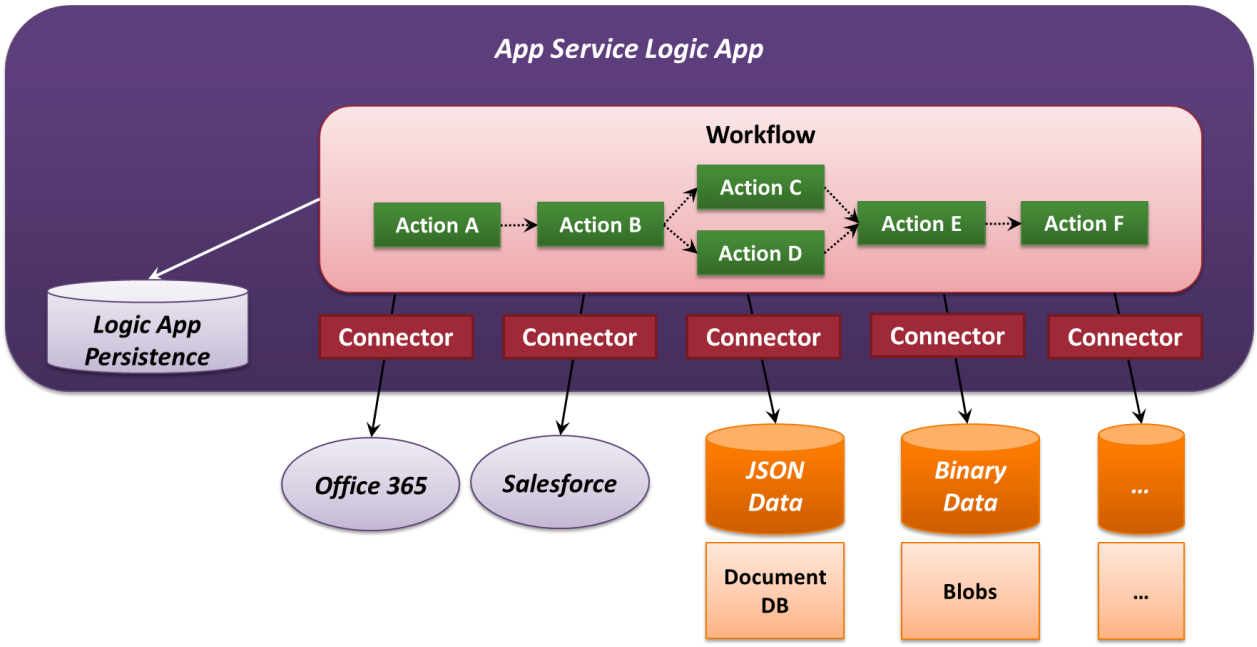

Figure 14: An integration application created as an App Service Logic App can use connectors to access SaaS applications, Azure PaaS data services, and more.

As the figure shows, the workflow in a Logic App consists of some number of *actions*. Each action does something, then stores the workflow's current state in a persistence mechanism provided by the Logic App itself. When the next action in the workflow starts, it reads this state, then executes the next step in whatever this workflow is doing.

Since a main purpose of Logic Apps is integration, they can also connect with the outside world. This might be done using App Service's API Apps, or it can use more specialized components called *connectors*. Connectors are available for many SaaS applications, such as Office 365 and Salesforce.com, as well as for many PaaS data services. (In fact, there are dozens and dozens of connectors available today, letting Logic Apps connect with lots of different things.) Once again, the importance of polyglot persistence is evident, as each action in a workflow can access and use whatever PaaS data service is best suited for its purpose.

Creating an Analytics Application

So far, the focus has been entirely on Azure's PaaS services for operational data. These services can support many kinds of applications, including banking systems, SaaS offerings, e-commerce web sites, integration applications, and lots more. But what happens when it's time to analyze this data? You'll need to build analytics applications, which typically process large amounts of historical data. In a PaaS world, where should this data be stored?

The answer is simple: Just as operational data is stored in operational PaaS data services, analytical data can be stored in analytical PaaS data services. Figure 15 illustrates this idea.

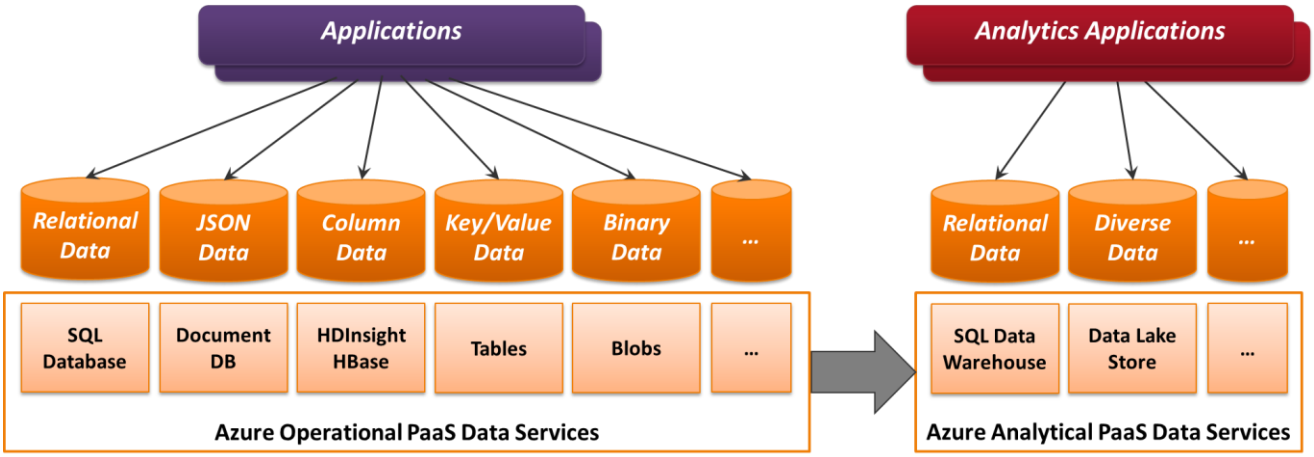

Figure 15: Azure provides PaaS data services for analytical data as well as for operational data.

It's important to realize that these analytical data services are right there in the cloud with Azure's operational data services. This makes moving data into them simpler than it would be with on-premises analytical data technologies. Also, remember that these analytical technologies are provided as PaaS services, with all of the advantages that implies.

As the figure shows, Azure provides a few choices for analytical PaaS data services, including the following:

- Azure SQL Data Warehouse, which provides a relational data warehouse in the cloud. This service can also allow access to non-relational data using a technology called PolyBase. Data stored here can be accessed directly via Power BI, Microsoft's interactive tool for business intelligence, or by other software.
- Azure Data Lake Store, which is an implementation of the Hadoop Distributed File System (HDFS). HDFS can store all kinds of data, and so Azure Data Lake Store is intended to hold very large amounts of pretty much anything. While polyglot persistence can make an application better, it can also complicate data analysis, since the application's operational data can be spread across different kinds of stores. By providing a common place to store all kinds of data for analysis, Azure Data Lake Store helps address this issue.

If you're creating an operational application, it's useful to keep in mind how you plan to analyze the historical data your application will create. If you expect to use primarily relational tools to do this, for instance, this might push you toward using SQL Database as your PaaS data service. Alternatively, if your application will generate very large amounts of data that will need to be analyzed using Hadoop or Azure Data Lake Analytics, you might choose a non-relational option for your operational store, such as DocumentDB or HBase.

Conclusion

Cloud computing changes how we create applications. One of the most important of these changes is that it encourages a PaaS view of the platform we build on, including data services. Among other things, this makes polyglot persistence simpler, letting developers more easily match their data with the best persistence option for that data.

As cloud computing and a PaaS approach continue to grow in importance, we all need to think about data and data services in a way that matches this new world. Nothing else makes sense.

About the Author

David Chappell is Principal of Chappell & Associates (http://www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.