When to Use Standards-Based APIs (Part 1)

IN MY FIRST FEW COLUMNS, I EXPLORED A VARIETY OF CLOUD STANDARDS TOPICS OF CURRENT INTEREST. The subjects of these columns have ranged from basic definitions of cloud terminology to surveys of recent activity to an assessment of the types of organizations that come together to create standards. I've also made several suggestions and observations as to how such organizations can change their working styles to adapt to the fast-changing pace of cloud development.

In this two-part column, I'd like to look at the role of application programming interfaces (APIs) in cloud computing, in particular, when to use and when not to use standards-based APIs in your work. By looking at situations in which standards are and aren't helpful, or at least when they are or aren't needed, we can arrive more clearly at an understanding of their important features and roles in cloud infrastructures and ecosystems. I'll try to make this discussion practical by including tools that are useful for various types of cloud applications as well as some specific open source examples.

ALAN SILL

Texas Tech University,
alan.sill@standards-now.org

The Purpose of APIs

APIs have come to be at the center of cloud computing thinking because their use as a central architectural design feature is well-matched to intrinsic prevailing assumptions about multiple instances, separation of design, and scaling methods currently available for distributed computing. To understand these assumptions, let's think, for example, of the alternative, in a hypothetical theoretical setting.

If all of the world's computing resources were concentrated in, let's say, a single central resource that had enormous bandwidth in and out, and tremendous computing capability to perform operations within a single massive memory space, the need for API-centric computing would be less than it is now and perhaps nonexistent. We might collectively imagine architectures in which different stages of computation would be organized and collected in separate workflow components, but the exercise would largely be academic. In practice, a programmer could compose a problem, put it into the megamachine for rapid computation, and get the results. Any API would be for the convenience of a particular community to make the composition of the input and formulation of the computational problem easy, and equivalently, to make it easy to retrieve the answer in simple terms, no more complex than the problem requires.

The situation I've just described stands in stark contrast to the actual state of large-scale worldwide computing. There are now many billions of points of computation in the world's current infrastructure, some supporting multiple containers and/or virtual machines, and the number of separate computer sales has reached several per person in many countries in the world. (See, for example, www.statisticbrain.com/computer-sales-statistics for current numbers.)

Many of these computing devices are in datacenters rather than homes and offices, but with the advent of the Internet of Things and other de-

2325-6095/15/$31.00 © 2015 IEEE

vice-centric computing deployment, the balance might tip in the other direction soon. Overall, we can anticipate that in the future, computing will continue to move even more toward multiple processes and devices that require organized points of interface and connection.

In this situation, APIs take on a central role in providing organized methods to allow different computing processing steps and operations to communicate. Well-designed APIs are therefore crucial to using the worldwide arrays of computing devices to separate the functions involved in performing computations, storing and moving data between locations, administering and organizing its use, limiting and protecting access, and so on.

In Clouds, Everything Has an Interface

Resources in cloud settings are often referred to collectively as "the cloud" as if it were one entity. In fact, there are many simultaneous clouds, each made up of multiple components that generally interact using standardized interfaces. Here, the term "standardized" is used in its loosest possible form, not referring specifically to the formal standards-based APIs that I explore later in this column. But even at the most basic level, clouds have to do with multiplicity, with the ability to deploy multiple instances and scale them on demand as well as hook them up with other components and cloud services.

Even if you write everything in your cloud deployment yourself, making no other reference to the API design tools and patterns further explored in this column, you'll have to decide on methods to connect, scale, and talk between the multiple components of your cloud deployment that follow a regular or at least predictable design.

Clouds therefore use interfaces in ways that single programs or self-contained information technology systems don't. They're intrinsically built around the idea of an interface, and to some degree, in clouds, everything's essentially available only via interfaces presented to other components.

People talk about APIs now in terms that clearly differentiate them from the underlying services to which they provide access. Third-party providers have sprung up to offer API design as a separate service that can be adapted to multiple underlying services, data structures, and functionality. The API has clearly come into its own as an aspect of programming that merits specific design and attention.

This situation has led to complaints from some who program, build, and support clouds and who would like to see innovation happen deeper within the software stack, and not just at the API level.[1] APIs are interesting, according to this viewpoint, but the core functionality of clouds is provided by the features of the components themselves as well as by the way they're accessed.

Thus, there are two ideas that must be reconciled in any particular cloud software deployment: the API-centric view and the functionality-first model. Of these, the latter is closest to the service-oriented architecture (SOA) roots of cloud computing, and the former, which puts APIs at the center of everything, is the newer evolution.

When to Use Standards-Based APIs

A standard in this context is simply another tool for organizing access to computational resources and data. The use of a standards-based API is simple to understand in this context: you should use standards when you need to communicate in an organized way with a repeatable pattern.

Even at the most basic level, clouds have to do with multiplicity, with the ability to deploy multiple instances and scale them on demand.

As I described in a previous column,[2] the paradigm of standards in information technology traces back to the beginnings of the industrial era and formed much of the technological basis for the period now known as the Industrial Revolution. As mentioned there, cross-fertilization of industrial engineering standards allowed invention in many other fields to spin up quickly, with advances in one discipline feeding quickly into adoption and progress in others, and I believe we're in a period that will be looked back on in the future as one of similar rapid cross-disciplinary expansion.

As I mentioned in that column, the "P" in API refers to programming—that is, to active design by a person or, in the not-so-distant future scenario, an automated system capable of learning from human patterns of application use. API design is therefore largely influenced by the need for programmers (human or automated) to be able to manage the details of boundary-level interactions between computing functions, even in highly automated and complex settings. Standards should be viewed as a tool for organizing our interactions at boundaries between separate processes or components of a cloud computing interaction.

Standards should be viewed as a tool for organizing our interactions at boundaries between separate processes or components.

Standards can also help communities form to tackle different aspects of particular problems. I explored this topic in a previous column,[3] and invite any interested communities to submit articles documenting their efforts to this magazine.

When Not to Use Standards-Based APIs

There are situations in which the use of a standards-based API could be a waste of time. The work might be temporary, or designed to be entirely local, or the characteristics of the internal code could be so unique as to make the application of a standards-based process either superfluous or optional. These situations, however, are in my experience and opinion much rarer and more restricted than people generally assume, and programmers and cloud users might use standards much more often than they realize. Nonetheless, let's look at some situations in which use of standards-based processes including APIs doesn't make sense.

When You Intend Never to Use the Results Again

The work might be temporary, as noted, or not intended to be shared either in input and/or output with other people, processes, or interactions, or otherwise a throw-away effort in situations that don't require reuse. In some circumstances, whether to use a preexisting pattern or something unique that doesn't have reusability as a core consideration is a toss-up decision.

When No One Will See the Work but You

You might just be writing a simple copy-and-paste intermediate result or otherwise generating a computational step, permanent or temporary set of data, or otherwise one-off situation that doesn't need to be shared with others. Note that this situation isn't logically identical to that of completely temporary considerations. The work might require reuse, but you're the only designer and user.

When the Work Is Dominated by Unique Considerations within a Small Community

A further evolution of the previous scenarios might occur in situations in which a small group of people agrees to exchange information among themselves without reference to a previous external method to organize this information. This step, of course, is one way that standards eventually come about: if the method becomes popular, the computational steps and/or data might need to be shared more broadly, and a successful shared technique could provide the basis for developing or using a more general standard.

When There's No API to Reuse That Fits the Need

Innovation is an important ingredient of progress. If you and/or any related community of current or potential future users are sure that your situation is unique, it might be worth developing a common API and even developing it into a standard. Creating an API ahead of time with characteristics that meet a particular need might, or might not, be required in a particular situation.

Examples of API Design for Data Access

This discussion might sound a little abstract, so this is a good moment to illustrate these concepts with a few specific examples. Let's start with the simple situation of providing access to a dataset using cloud-like features.

As a long-time database practitioner and user, I'm often frustrated when people present complex data to me in the form of a simple spreadsheet. Although spreadsheets are useful tools in some situations, from my point of view, any time the dataset grows in size or complexity, it becomes difficult to do anything truly interesting or flexible within the limitations of that presentation and access paradigm. It's also difficult, although not impossible, to scale methods to access datasets when they're locked up in spreadsheets.

To make progress, one can upload the spreadsheet to more general cloud-based tools, and even translate them to a variety of SQL-based or No-SQL query frameworks, but jumping all the way to the use of such tools directly loses the value of this simple data access scenario. There are also other things you can do as intermediate steps to illustrate some useful API design and usage concepts. For this reason, let's start from a simple common data format in a stand-alone setting, and work our way up to a full cloud-based setting, adding increasingly sophisticated features along the way.

Example 1: Access Spreadsheet as a Database

The simplest thing to do to get general querying and introspection control over spreadsheet data, and something I do often, is to access the data in the spreadsheet as if it were in a database. Contrary to what many people think, this doesn't require that you design a set of database tables or even install or operate a database program. You can do the basics with just an adapter designed for this purpose and any database query tool.

One way to do this is to present access to comma separated variable (CSV) formatted data that can be written by any spreadsheet program as a set of database tables through a specially written Java database connector (JDBC). This makes it easy to explore the spreadsheet data using SQL queries in a more flexible way than through the internal search and macroprogramming methods of most spreadsheet programs. There are many such adapters commercially available. An open source example is the CsvJDBC tool (http://csvjdbc.sourceforge.net), and another is the JDBC-csv-driver project (https://github.com/jprante/jdbc-driver-csv).

Standards are involved from the very outset in this example. The CSV format, while not formally standardized, is basically an evolution of commonly used patterns for export and import of tabular data. Various approaches have been used over time for such tasks. A common starting point for understanding this format is provided by RFC 4180, which is an informational-class publication from the Internet Engineering Task Force (IETF).[4] RFC 4180 documents various approaches to internal formatting of CSV files and their use as an Internet media type. Such media types are described in Internet parlance as multipurpose Internet mail extensions (MIME), and RFC 4180 also formally registers the "text/csv" MIME type.

After many years of use, this document inspired RFC 7111, an additional informational publication relevant to cloud and Web usage patterns that defines uniform resource identifier (URI) fragment identifiers for text/csv entities.[5] These fragment identifiers make it possible to refer to parts of a text/csv MIME entity individually by row, column, or cell, and can support implementations that retrieve specific portions of a CSV entity. In Web and cloud contexts, the combination of RFC 4180 and RFC 7111 can provide a simple, reusable pattern for machine access to lightly formatted CSV data.

Standards are also involved in the database programming aspects of this example. Although Java is a product of a particular company (originally Sun Microsystems and now Oracle), many aspects of its continuing development are subject to a community process, and JDBC is a specification available through that process. (See www.jcp.org and www.oracle.com/technetwork/java/javase/jdbc/index.html for more information.)

SQL is a published standard of the International Organization for Standardization (ISO), most recently updated in 2011,[6] that originated in earlier specifications published by the American National Standards Institute (ANSI) in 1986 and was then further updated with a variety of extensions before its submission to ISO for adoption as an international standard.

I've used this pattern many times to throw a quick database access layer on top of a complex spreadsheet or set of related spreadsheets and write simple access queries to derive and format results without resorting to the intricacies of spreadsheet macroprogramming. The fact that tools of this nature can be made available and useful, even in quick-and-dirty utility examples of this kind, is a direct result of these standards' existence.

No significant processing layer is present in this example; it's a simple case of using standards-based interfaces and related common methods to provide options for more flexible patterns for access to data.

Example 2: Give Your Data an API

The next step in handling simple datasets would be to use tools that add some local processing capabilities. A useful example is provided by the OpenRefine project (http://openrefine.org), an open source tool that deploys a small local webserver provisioned with some advanced tools for importing local datasets and accessing other remote data via the Web. These tools address problems that can arise from accessing messy, inconsistently formatted, or incomplete data that are sometimes found in such loose collections or mashups. Using them, you can import CSV files as in the previous example and combine these with other datasets pulled from the Web in a variety of formats.

OpenRefine provides tools, macros, and methods aimed at interactive exploration of such data, and can also save such processing steps in JavaScript form for later repetition or automation. It thus represents a step toward the full-scale automation, distributed data access, and API methods you might find in more sophisticated cloud tools. I've used OpenRefine many times to grab and explore complex datasets, including those from the local database access example, and sometimes tools of this nature provide all of the API features you might need to gain programmatic access to such data.

Another interesting open source tool in this category that can help illustrate the separation of the concepts of API design from those of internal code characteristics is provided by the API-in-a-Box project (https://github.com/switzersc/api-in-a-box). This project leverages several modern tools and concepts at once by providing a framework that sets up an elastic search container, an API container that supplies RESTful endpoints to list, search, and return results on resources, and an interface that lets you store the data to serve as the source for these resources in the form of CSV files, just like those in the previous examples, that you upload to a GitHub repository of your choice.

The API-in-a-Box and OpenRefine examples have now elevated our humble CSV-formatted data files to the point that they're nearly indistinguishable from fully cloud-based data. In combination, these tools allow you to set up and use RESTful hypermedia API frameworks to access your own or other data with a cloud-friendly search framework at your disposal. As standards for container design and deployment and related standards and techniques for cloud-based data access evolve, we might see many further examples of this nature.

Example 3: Full Cloud-Based Data Management

The examples given above followed a simple CSV format through several variations, ending with a tool that allows you to treat such data as you would using cloud-based methods. Once you understand your sources and these simple tools, the next approach to generalizing data access tasks might be to put relevant original datasets into cloud-based locations and write a custom interface for each dataset. The API-in-a-Box approach provides a simple user-driven example, but most cloud-based data storage frameworks are equipped with their own built-in API methods.

Further discussion on this topic is out of scope for this column, but it suffices to say that the range of applicable API tools, standards, and methods is wide and varied, and we'll return to this topic in the future.

IN THE SECOND OF THIS TWO-PART COLUMN, I'LL DISCUSS TOPICS RELATED TO DATACENTER INFRASTRUCTURE CONTROL (BOTH PHYSICAL AND PROGRAMMATIC), AND GIVE SOME RELATED EXAMPLES. I'll also deliver on an earlier promise to discuss application binary interfaces (ABIs) in contrast to APIs, and dive into details of both types of these interfaces that are becoming important in cloud computing applications.

Meanwhile, please respond with your opinions on this topic or on those explored in previous columns. Let us know what you think, and please also include any news you think the community should know about the general areas of cloud standards, compliance, or related topics. We're always open to article submissions, and I'm happy to review ideas for such submissions, or topics for proposed guest columns. I can be reached for this purpose at alan.sill@standards-now.org.

References

1. S. Petersen, "OpenStack Making Progress, but Manageability Remains a Concern," *eWeek*, 24 May 2015; http://www.eweek.com/cloud/openstack-making-progress-but-manageability-remains-a-concern.html.
2. A. Sill, "Invention, Innovation, and New APIs," *IEEE Cloud Computing*, vol. 2, no. 2, 2015, pp. 6–9.
3. A. Sill, "The Role of Communities in Developing Cloud Standards," *IEEE Cloud Computing*, vol. 1, no. 2, 2014, pp. 16–19.
4. Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, IETF RFC 4180, Oct 2005; https://tools.ietf.org/html/rfc4180.
5. M. Hausenblas, E. Wilde, and J. Tennison, *URI Fragment Identifiers for the text/csv Media Type*, IETF RFC 7111, Jan. 2014; https://tools.ietf.org/html/rfc7111.
6. *Information Technology—Database Languages—SQL—Part 1: Framework (SQL/Framework)*, ISO/IEC 9075-1, 4th ed., Int'l Organization for Standardization, 2011.

ALAN SILL *is interim senior director at the High Performance Computing Center at Texas Tech University, where he's also site director for the US National Science Foundation Cloud and Autonomic Computing Center and adjunct professor of physics. Sill has a PhD in particle physics from American University and is an active member of IEEE, the Distributed Management Task Force, and TeleManagement Forum, and he serves as president for the Open Grid Forum. He's a member of several cloud standards working groups and national and international standards roadmap committees, and he remains active in particle physics and advanced computing research. For further details, visit http://cac.ttu.edu or contact him at alan.sill@standards-now.org.*