Chapter 7

Common Standards in Cloud Computing

7.1 Chapter Overview

In Internet circles, everything eventually gets driven by a working group of one sort or another. A working group is an assembled, cooperative collaboration of researchers working on new research activities that would be difficult for any one member to develop alone. A working group can can exist for anywhere between a few months and many years. Working groups generally strive to create an informational document a standard, or find some resolution for problems related to a system or network. Most often, the working group attempts to assemble experts on a topic. Together, they will work intensively toward their goal. Working groups are sometimes also referred to as task groups or technical advisory groups. In this chapter, we will discuss the Open Cloud Consortium (OCC) and the Distributed Management Task Force (DMTF) as examples of cloud-related working groups. We will also discuss the most common standards currently used in cloud environments.

7.2 The Open Cloud Consortium

The purpose of the Open Cloud Consortium is to support the development of standards for cloud computing and to develop a framework for interoperability among various clouds. The OCC supports the development of benchmarks for cloud computing and is a strong proponent of open source software to be used for cloud computing. OCC manages a testing platform and a test-bed for cloud computing called the Open Cloud Test-bed. The group also sponsors workshops and other events related to cloud computing.

The OCC is organized into several different working groups. For example, the Working Group on Standards and Interoperability for Clouds

That Provide On-Demand Computing Capacity focuses on developing standards for interoperating clouds that provide on-demand computing capacity. One architecture for clouds that was popularized by a series of Google technical reports describes a *storage cloud* providing a distributed file system, a *compute cloud* supporting MapReduce, and a *data cloud* supporting table services. The open source Hadoop system follows this architecture. These types of cloud architectures support the concept of on-demand computing capacity.

There is also a Working Group on Wide Area Clouds and the Impact of Network Protocols on Clouds. The focus of this working group is on developing technology for wide area clouds, including creation of methodologies and benchmarks to be used for evaluating wide area clouds. This working group is tasked to study the applicability of variants of TCP (Transmission Control Protocol) and the use of other network protocols for clouds.

The Open Cloud Test-bed uses Cisco C-Wave and the UIC Teraflow Network for its network connections. C-Wave makes network resources available to researchers to conduct networking and applications research. It is provided at no cost to researchers and allows them access to 10G Waves (Layer-1 p2p) on a per-project allocation. It provides links to a 10GE (gigabit Ethernet) switched network backbone. The Teraflow Test-bed (TFT) is an international application network for exploring, integrating, analyzing, and detecting changes in massive and distributed data over wide-area high-performance networks. The Teraflow Test-bed analyzes streaming data with the goal of developing innovative technology for data streams at very high speeds. It is hoped that prototype technology can be deployed over the next decade to analyze 100-gigabit-per-second (Gbps) and 1,000-Gbps streams.

Both of these products use wavelengths provided by the National Lambda Rail (NLR). The NLR can support many distinct networks for the U.S. research community using the same core infrastructure. Experimental and productions networks exist side by side but are physically and operationally separate. Production networks support cutting-edge applications by providing users guaranteed levels of reliability, availability, and performance. At the same time, experimental networks enable the deployment and testing of new networking technologies, providing researchers national-scale test-beds without the limitations typically associated with production networks.

The Working Group on Information Sharing, Security, and Clouds has a primary focus on standards and standards-based architectures for sharing information between clouds. This is especially true for clouds belonging to different organizations and subject to possibly different authorities and policies. This group is also concerned with security architectures for clouds. An example is exchanging information between two clouds, each of which is HIPAA-compliant, but when each cloud is administered by a different organization.

Finally, there is an Open Cloud Test-bed Working Group that manages and operates the Open Cloud Test-bed. Currently, membership in this working group is limited to those who contribute computing, networking, or other resources to the Open Cloud Test-bed. For more information on the Open Cloud Consortium, the reader is encouraged to visit the OCC website.[1]

7.3 The Distributed Management Task Force

According to their web site, the Distributed Management Task Force

> . . . enables more effective management of millions of IT systems worldwide by bringing the IT industry together to collaborate on the development, validation and promotion of systems management standards. The group spans the industry with 160 member companies and organizations, and more than 4,000 active participants crossing 43 countries. The DMTF board of directors is led by 16 innovative, industry-leading technology companies. They include Advanced Micro Devices (AMD); Broadcom Corporation; CA, Inc.; Dell; EMC; Fujitsu; HP; Hitachi, Ltd.; IBM; Intel Corporation; Microsoft Corporation; Novell; Oracle; Sun Microsystems, Inc.; Symantec Corporation and VMware, Inc. With this deep and broad reach, DMTF creates standards that enable interoperable IT management. DMTF management standards are critical to enabling management interoperability among multi-vendor systems, tools and solutions within the enterprise.[2]

The DMTF started the Virtualization Management Initiative (VMAN). The VMAN unleashes the power of virtualization by delivering broadly supported interoperability and portability standards to virtual computing environments. VMAN enables IT managers to deploy preinstalled,

1. http://www.opencloudconsortium.org/working-groups.html.
2. http://www.dmtf.org/about, retrieved 21 Feb 2009.

preconfigured solutions across heterogeneous computing networks and to manage those applications through their entire life cycle. Management software vendors offer a broad selection of tools that support the industry standard specifications that are now a part of VMAN. This helps in lowering support and training costs for IT managers. Virtualization has enhanced the IT industry by optimizing use of existing physical resources and helping reduce the number of systems deployed and managed. This consolidation reduces hardware costs and mitigates power and cooling needs. However, even with the efficiencies gained by virtualization, this new approach does add some IT cost due to increased system management complexity.

Since the DMTF builds on existing standards for server hardware, management tool vendors can easily provide holistic management capabilities to enable IT managers to manage their virtual environments in the context of the underlying hardware. This lowers the IT learning curve, and also lowers complexity for vendors implementing this support in their solutions. With the technologies available to IT managers through the VMAN Initiative, companies now have a standardized approach to

1. Deploy virtual computer systems
2. Discover and take inventory of virtual computer systems
3. Manage the life cycle of virtual computer systems
4. Add/change/delete virtual resources
5. Monitor virtual systems for health and performance

7.3.1 Open Virtualization Format

The Open Virtualization Format (OVF) is a fairly new standard that has emerged within the VMAN Initiative. The OVF simplifies interoperability, security, and virtual machine life-cycle management by describing an open, secure, portable, efficient, and extensible format for the packaging and distribution of one or more virtual appliances. The OVF specifies procedures and technologies to permit integrity checking of the virtual machines (VM) to ensure that they have not been modified since the package was produced. This enhances security of the format and will help to alleviate security concerns of users who adopt virtual appliances produced by third parties. The OVF also provides mechanisms that support license checking for the enclosed VMs, addressing a key concern of both independent software vendors and customers. Finally, the OVF allows an installed VM to acquire information about its host virtualization platform and runtime environment, which allows the VM to localize the applications it contains and optimize its performance for the particular virtualization environment.

One key feature of the OVF is virtual machine packaging portability. Since OVF is, by design, virtualization platform-neutral, it provides the benefit of enabling platform-specific enhancements to be captured. It also supports many open virtual hard disk formats. Virtual machine properties are captured concisely using OVF metadata. OVF is optimized for secure distribution. It supports content verification and integrity checking based on industry-standard public key infrastructure and provides a basic scheme for management of software licensing.

Another benefit of the OVG is a simplified installation and deployment process. The OVF streamlines the entire installation process using metadata to validate the entire package and automatically determine whether a virtual appliance can be installed. It also supports both single-VM and multiple-VM configurations and packages containing complex, multitier services consisting of multiple interdependent VMs. Since it is vendor- and platform-independent, the OVF does not rely on the use of a specific host platform, virtualization platform, or guest operating system.

The OVF is designed to be extended as the industry moves forward with virtual appliance technology. It also supports and permits the encoding of vendor-specific metadata to support specific vertical markets. It is localizable—it supports user-visible descriptions in multiple locales, and localization of interactive processes during installation of a virtual appliance. This allows a single packaged virtual appliance to serve multiple markets.

7.4 Standards for Application Developers

The purpose of application development standards is to ensure uniform, consistent, high-quality software solutions. Programming standards are important to programmers for a variety of reasons. Some researchers have stated that, as a general rule, 80% of the lifetime cost of a piece of software goes to maintenance. Furthermore, hardly any software is maintained by the original author for its complete life cycle. Programming standards help to improve the readability of the software, allowing developers to understand new code more quickly and thoroughly. If you ship source code as a product, it is important to ensure that it is as well packaged and meets industry standards comparable to the products you compete with. For the standards to work, everyone developing solutions must conform to them. In the following sections, we discuss application standards that are commonly used across the Internet in browsers, for transferring data, sending messages, and securing data.

7.4.1 Browsers (Ajax)

Ajax, or its predecessor AJAX (Asynchronous JavaScript and XML), is a group of interrelated web development techniques used to create interactive web applications or rich Internet applications. Using Ajax, web applications can retrieve data from the server asynchronously, without interfering with the display and behavior of the browser page currently being displayed to the user. The use of Ajax has led to an increase in interactive animation on web pages. Despite its name, JavaScript and XML are not actually *required* for Ajax. Moreover, requests do not even need to be asynchronous. The original acronym AJAX has changed to the name Ajax to reflect the fact that these specific technologies are no longer required.

In many cases, related pages that coexist on a web site share much common content. Using traditional methods, such content must be reloaded every time a request is made. Using Ajax, a web application can request only the content that needs to be updated. This greatly reduces networking bandwidth usage and page load times. Using asynchronous requests allows a client browser to appear more interactive and to respond to input more quickly. Sections of pages can be reloaded individually. Users generally perceive the application to be faster and more responsive. Ajax can reduce connections to the server, since scripts and style sheets need only be requested once.

An Ajax framework helps developers create web applications that use Ajax. The framework helps them to build dynamic web pages on the client side. Data is sent to or from the server using requests, usually written in JavaScript. On the server, some processing may be required to handle these requests, for example, when finding and storing data. This is accomplished more easily with the use of a framework dedicated to process Ajax requests. One such framework, ICEfaces, is an open source Java product maintained by http://icefaces.org.

ICEfaces Ajax Application Framework

ICEfaces is an integrated Ajax application framework that enables Java EE application developers to easily create and deploy thin-client rich Internet applications in pure Java. ICEfaces is a fully featured product that enterprise developers can use to develop new or existing Java EE applications at no cost. ICEfaces is the most successful enterprise Ajax framework available under open source. The ICEfaces developer community is extremely vibrant, already exceeding 32,000 developers in 36 countries. To run ICEfaces applications, users need to download and install the following products:

- Java 2 Platform, Standard Edition
- Ant
- Tomcat
- ICEfaces
- Web browser (if you don't already have one installed)

ICEfaces leverages the entire standards-based Java EE set of tools and environments. Rich enterprise application features are developed in pure Java in a thin-client model. No Applets or proprietary browser plug-ins are required. ICEfaces applications are JavaServer Faces (JSF) applications, so Java EE application development skills apply directly and Java developers don't have to do any JavaScript-related development.

Because ICEfaces is a pure Java enterprise solution, developers can continue to work the way they normally do. They are able to leverage their existing Java integrated development environments (IDEs) and test tools for development. ICEfaces supports an array of Java Application Servers, IDEs, third-party components, and JavaScript effect libraries. ICEfaces pioneered a technique called Ajax Push. This technique enables server/application-initiated content rendering to be sent to the browser. Also, ICEfaces is the one of the most secure Ajax solutions available. Compatible with SSL (Secure Sockets Layer) protocol, it prevents cross-site scripting, malicious code injection, and unauthorized data mining. ICEfaces does not expose application logic or user data, and it is effective in preventing fake form submits and SQL (Structured Query Language) injection attacks. ICEfaces also supports third-party application server Asynchronous Request Processing (ARP) APIs provided by Sun Glassfish (Grizzly), Jetty, Apache Tomcat, and others.

7.4.2 Data (XML, JSON)

Extensible Markup Language (XML) is a specification for creating custom markup languages. It is classified as an extensible language because it allows the user to define markup elements. Its purpose is to enable sharing of structured data. XML is often used to describe structured data and to serialize objects. Various XML-based protocols exist to represent data structures for data interchange purposes. Using XML is arguably more complex than using JSON (described below), which represents data structures in simple text formatted specifically for data interchange in an uncompressed form. Both XML and JSON lack mechanisms for representing large binary data types such as images.

XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting. The benefit here is the ability to reuse that content in other applications or for other presentation environments. Most important, XML provides a basic syntax that can be used to share information among different kinds of computers, different applications, and different organizations without needing to be converted from one to another.

An XML document has two correctness levels, *well formed* and *valid*. A well-formed document conforms to the XML syntax rules. A document that is not well formed is not in XML format, and a conforming parser will not process it. A valid document is well formed and additionally conforms to semantic rules which can be user-defined or exist in an XML schema. An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic constraints imposed by XML itself. A number of standard and proprietary XML schema languages have emerged for the purpose of formally expressing such schemas, and some of these languages are themselves XML-based.

XML documents must conform to a variety of rules and naming conventions. By carefully choosing the names of XML elements, it is possible to convey the meaning of the data in the markup itself. This increases human readability while retaining the syntactic structure needed for parsing. However, this can lead to verbosity, which complicates authoring and increases file size. When creating XML, the designers decided that by leaving the names, allowable hierarchy, and meanings of the elements and attributes open and definable by a customized schema, XML could provide a syntactic foundation for the creation of purpose-specific, XML-based markup languages. The general syntax of such languages is very rigid. Documents must adhere to the general rules of XML, ensuring that all XML-aware software can at least read and understand the arrangement of information within them. The schema merely supplements the syntax rules with a predefined set of constraints.

Before the advent of generalized data description languages such as XML, software designers had to define special file formats or small languages to share data between programs. This required writing detailed specifications and special-purpose parsers and writers. XML's regular structure and strict parsing rules allow software designers to leave the task of parsing to standard tools, since XML provides a general, data model-oriented framework for the development of application-specific languages. This allows software designers to concentrate on the development of rules for their data at relatively high levels of abstraction.

JavaScript Object Notation (JSON)

JSON is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). The JSON format is specified in Internet Engineering Task Force Request for Comment (RFC) 4627. The JSON format is often used for transmitting structured data over a network connection in a process called serialization. Its main application is in Ajax web application programming, where it serves as an alternative to the XML format. JSON is based on a subset of the JavaScript programming language. It is considered to be a language-independent data format. Code for parsing and generating JSON data is readily available for a large variety of programming languages. The json.org website provides a comprehensive listing of existing JSON bindings, organized by language.

Even though JSON was intended as a data serialization format, its design as a subset of the JavaScript language poses security concerns. The use of a JavaScript interpreter to dynamically execute JSON text as JavaScript can expose a program to bad or even malicious script. JSON is also subject to cross-site request forgery attacks. This can allow JSON-encoded data to be evaluated in the context of a malicious page, possibly divulging passwords or other sensitive data. This is only a problem if the server depends on the browser's Same Origin Policy to block the delivery of the data in the case of an improper request. When the server determines the propriety of the request, there is no problem because it will only output data if the request is valid. Cookies are not adequate for determining whether a request is authorized and valid. The use of cookies is subject to cross-site request forgery and should be avoided with JSON. As you can see, JSON was built for simple tasks and can be useful, but there is some risk involved in using it—especially given the alternative solutions available today.

7.4.3 Solution Stacks (LAMP and LAPP)

LAMP

LAMP is a popular open source solution commonly used to run dynamic web sites and servers. The acronym derives from the fact that it includes **L**inux, **A**pache, **M**ySQL, and **P**HP (or Perl or Python) and is considered by many to be the platform of choice for development and deployment of high-performance web applications which require a solid and reliable foundation. The combination of these technologies is used primarily to define a web server infrastructure or for creating a programming environment for developing software. While the creators of these open source products did not intend for them all to work with each other, the LAMP combination has become popular because of its open source nature, low cost, and the wide distribution of its components (most of which come bundled with nearly all of the current Linux distributions). When used in combination, they represent a solution stack of technologies that support application servers.

Linux, Apache, PostgreSQL, and PHP(or Perl or Python)

The LAPP stack is an open source web platform that can be used to run dynamic web sites and servers. It is considered by many to be a more powerful alternative to the more popular LAMP stack. These advanced and mature components provide a rock-solid foundation for the development and deployment of high-performance web applications. LAPP offers SSL, PHP, Python, and Perl support for Apache2 and PostgreSQL. There is an administration front-end for PostgreSQL as well as web-based administration modules for configuring Apache2 and PHP. PostgreSQL password encryption is enabled by default. The PostgreSQL user is trusted when connecting over local Unix sockets. Many consider the LAPP stack a more secure out-of-the-box solution than the LAMP stack. The choice of which stack to use is made by developers based on the purpose of their application and the risks they may have to contend with when users begin working with the product.

7.5 Standards for Messaging

You probably think you know what a messaging standard is. Unfortunately, the term *messaging* means different things to different people. So does the word *standard*. People may assume you are talking about networking when you begin discussing messaging standards. The term *messaging,* however, covers a lot of ground, and not all of it is specific to networking. For our purposes here, a *message* is a unit of information that is moved from one place to another. The term *standard* also is not always clearly defined. Different entities have differing interpretations of what a standard is, and we know there are open international standards, *de facto* standards, and proprietary standards. A true standard is usually characterized by certain traits, such as being managed by an international standards body or an industry consortium, and the standard is created jointly by a community of interested parties. The Internet Engineering Task Force (IETF) is perhaps the most open standards body on the planet, because it is open to everyone. Participants can contribute, and their work is available online for free. In the following sections, we discuss the most common messaging standards used in the cloud—some of which have been used so much so that they are considered *de facto* standards.

7.5.1 Simple Message Transfer Protocol (SMTP)

Simple Message Transfer Protocol is arguably the most important protocol in use today for basic messaging. Before SMTP was created, email messages were sent using File Transfer Protocol (FTP). A sender would compose a message and transmit it to the recipient as if it were a file. While this process worked, it had its shortcomings. The FTP protocol was designed to transmit files, not messages, so it did not provide any means for recipients to identify the sender or for the sender to designate an intended recipient. If a message showed up on an FTP server, it was up to the administrator to open or print it (and sometimes even deliver it) before anyone even knew who it was supposed to be receiving it.

SMTP was designed so that sender and recipient information could be transmitted with the message. The design process didn't happen overnight, though. SMTP was initially defined in 1973 by IETF RFC 561. It has evolved over the years and has been modified by RFCs 680, 724 and 733. The current RFCs applying to SMTP are RFC 821 and RFC 822. SMTP is a two-way protocol that usually operates using TCP (Transmission Control Protocol) port 25. Though many people don't realize it, SMTP can be used to both send and receive messages. Typically, though, workstations use POP (Post Office Protocol) rather than SMTP to receive messages. SMTP is usually used for either sending a message from a workstation to a mail server or for communications between mail servers.

7.5.2 Post Office Protocol (POP)

SMTP can be used both to send and receive messages, but using SMTP for this purpose is often impractical or impossible because a client must have a constant connection to the host to receive SMTP messages. The Post Office Protocol (POP) was introduced to circumvent this situation. POP is a lightweight protocol whose single purpose is to download messages from a server. This allows a server to store messages until a client connects and requests them. Once the client connects, POP servers begin to download the messages and subsequently delete them from the server (a default setting) in order to make room for more messages. Users respond to a message that was downloaded using SMTP. The POP protocol is defined by RFC 1939 and usually functions on TCP port 110.

7.5.3 Internet Messaging Access Protocol (IMAP)

Once mail messages are downloaded with POP, they are automatically deleted from the server when the download process has finished. Thus POP users have to save their messages locally, which can present backup challenges when it is important to store or save messages. Many businesses have compulsory compliance guidelines that require saving messages. It also becomes a problem if users move from computer to computer or use mobile networking, since their messages do not automatically move where they go. To get around these problems, a standard called Internet Messaging Access Protocol was created. IMAP allows messages to be kept on the server but viewed and manipulated (usually via a browser) as though they were stored locally. IMAP is a part of the RFC 2060 specification, and functions over TCP port 143.

7.5.4 Syndication (Atom, Atom Publishing Protocol, and RSS)

Content syndication provides citizens convenient access to new content and headlines from government via RSS (Really Simple Syndication) and other online syndication standards. Governments are providing access to more and more information online. As web sites become more complex and difficult to sift through, new or timely content is often buried. Dynamically presenting “what’s new” an the top of the web site is only the first step. Sharing headlines and content through syndication standards such as RSS (the little orange [XML] button, ATOM, and others) essentially allows a government to control a small window of content across web sites that choose to display the government's headlines. Headlines may also be aggregated and displayed through "newsreaders" by citizens through standalone applications or as part of their personal web page.

Portals can automatically aggregate and combine headlines and/or lengthier content from across multiple agency web sites. This allows the value of distributed effort to be shared, which is more sustainable. Press releases may be aggregated automatically from different systems, as long as they all are required to offer an RSS feed with content tagged with similar metadata. Broader use of government information online, particularly time-sensitive democratic information, justifies the effort of production and the accountability of those tasked to make it available.

- **Benefits:** Ability to scan headlines from many sources, all in one place, through a newsreader. Time-saving awareness of new content from government, if the RSS feed or feeds are designed properly. Ability to monitor new content from across the council, as well as display feeds on their own web site. Awareness of new content position councilors as guides to government for citizens. Ability to aggregate new content or headlines from across multiple office locations and agencies. This allows a display of "joined-up" government despite structural realities. Journalists and other locally focused web sites will be among the primary feed users.
- **Limitations:** Dissemination via syndication is a new concept to governments just getting used to the idea of remote online public access to information. Governments need to accept that while they control the content of the feed, the actual display of the headlines and content will vary. Popular RSS feeds can use significant amounts of bandwidth. Details on how often or when a feed is usually updated should be offered to those grabbing the code behind the orange [XML] button, so they "ping" it once a day instead of every hour. Automated syndication requires use of a content management system. Most viable content management systems have integrated RSS functions, but the sophistication, ease of use, and documentation of these tools vary. There are three variants of RSS, as well as the emerging ATOM standard. It

is recommended that a site pick the standard most applicable to their content rather than confuse users with different feeds providing the same content.

RSS

RSS is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document includes full or summarized text, plus metadata such as publishing dates and authorship. Web feeds benefit publishers by letting them syndicate content automatically. They benefit readers who want to subscribe to timely updates from favored web sites or to aggregate feeds from many sites into one place. RSS feeds can be read using software called a reader that can be web-based, desktop-based, a mobile device, or any computerized Internet-connected device. A standardized XML file format allows the information to be published once and viewed by many different programs. The user subscribes to a feed by entering the feed's URI (often referred to informally as a URL, although technically, those two terms are not exactly synonymous) into the reader or by clicking an RSS icon in a browser that initiates the subscription process. The RSS reader checks the user's subscribed feeds regularly for new work, downloads any updates that it finds, and provides a user interface to monitor and read the feeds.

Atom and Atom Publishing Protocol (APP)

The name Atom applies to a pair of related standards. The Atom Syndication Format is an XML language used for web feeds, while the Atom Publishing Protocol (AtomPub or APP) is a simple HTTP-based protocol (HTTP is described later in this chapter) for creating and updating web resources, sometimes known as web feeds. Web feeds allow software programs to check for updates published on a web site. To provide a web feed, a site owner may use specialized software (such as a content management system) that publishes a list (or "feed") of recent articles or content in a standardized, machine-readable format. The feed can then be downloaded by web sites that syndicate content from the feed, or by feed reader programs that allow Internet users to subscribe to feeds and view their content. A feed contains entries, which may be headlines, full-text articles, excerpts, summaries, and/or links to content on a web site, along with various metadata.

The Atom format was developed as an alternative to RSS. Ben Trott, an advocate of the new format that became Atom, believed that RSS had limitations and flaws—such as lack of ongoing innovation and its necessity to remain backward compatible—and that there were advantages to a fresh design. Proponents of the new format formed the IETF Atom Publishing Format and Protocol Workgroup. The Atom syndication format was published as an IETF "proposed standard" in RFC 4287, and the Atom Publishing Protocol was published as RFC 5023.

Web feeds are used by the weblog community to share the latest entries' headlines or their full text, and even attached multimedia files. These providers allow other web sites to incorporate the weblog's "syndicated" headline or headline-and-short-summary feeds under various usage agreements. Atom and other web syndication formats are now used for many purposes, including journalism, marketing, "bug" reports, or any other activity involving periodic updates or publications. Atom also provides a standardized way to export an entire blog, or parts of it, for backup or for importing into other blogging systems.

A program known as a feed reader or aggregator can check web pages on behalf of a user and display any updated articles that it finds. It is common to find web feeds on major web sites, as well as on many smaller ones. Some web sites let people choose between RSS- or Atom-formatted web feeds; others offer only RSS or only Atom. In particular, many blog and wiki sites offer their web feeds in the Atom format.

Client-side readers and aggregators may be designed as standalone programs or as extensions to existing programs such as web browsers. Browsers are moving toward integrated feed reader functions. Such programs are available for various operating systems. Web-based feed readers and news aggregators require no software installation and make the user's feeds available on any computer with web access. Some aggregators syndicate web feeds into new feeds, e.g., taking all football-related items from several sports feeds and providing a new football feed. There are several search engines which provide search functionality over content published via these web feeds.

Web Services (REST)

REpresentational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. As such, it is not strictly a method for building "web services." The terms "representational state transfer" and "REST" were introduced in 2000 in the doctoral dissertation of Roy Fielding,[3] one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification.

REST refers to a collection of network architecture principles which outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface which transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. These two meanings can conflict as well as overlap. It is possible to design a software system in accordance with Fielding's REST architectural style without using HTTP and without interacting with the World Wide Web.[4] It is also possible to design simple XML+HTTP interfaces which do not conform to REST principles, but instead follow a model of remote procedure call. Systems which follow Fielding's REST principles are often referred to as "RESTful."

Proponents of REST argue that the web's scalability and growth are a direct result of a few key design principles. Application state and functionality are abstracted into resources. Every resource is uniquely addressable using a universal syntax for use in hypermedia links, and all resources share a uniform interface for the transfer of state between client and resource. This transfer state consists of a constrained set of well-defined operations and a constrained set of content types, optionally supporting code on demand. State transfer uses a protocol which is client-server based, stateless and cacheable, and layered. Fielding describes REST's effect on scalability thus:

> REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate

3. Roy T.Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," dissertation, University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
4. Ibid.

semantics and exchange information, and responses explicitly indicate cacheability.[5]

An important concept in REST is the existence of resources, each of which is referenced with a global identifier (e.g., a URI in HTTP). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange representations of these resources (the actual documents conveying the information). For example, a resource which is a circle may accept and return a representation which specifies a center point and radius, formatted in SVG (Scalable Vector Graphics), but may also accept and return a representation which specifies any three distinct points along the curve as a comma-separated list.

Any number of connectors (clients, servers, caches, tunnels, etc.) can mediate the request, but each does so without "seeing past" its own request (referred to as "layering," another constraint of REST and a common principle in many other parts of information and networking architecture). Thus an application can interact with a resource by knowing two things: the identifier of the resource, and the action required—it does not need to know whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between it and the server actually holding the information. The application does, however, need to understand the format of the information (representation) returned, which is typically an HTML, XML, or JSON document of some kind, although it may be an image, plain text, or any other content.

REST provides improved response time and reduced server load due to its support for the caching of representations. REST improves server scalability by reducing the need to maintain session state. This means that different servers can be used to handle different requests in a session. REST requires less client-side software to be written than other approaches, because a single browser can access any application and any resource. REST depends less on vendor software and mechanisms which layer additional messaging frameworks on top of HTTP. It provides equivalent functionality when compared to alternative approaches to communication, and it does not require a separate resource discovery mechanism, because of the use of hyperlinks in representations. REST

5. Ibid.

also provides better long-term compatibility because of the capability of document types such as HTML to evolve without breaking backwards or forwards compatibility and the ability of resources to add support for new content types as they are defined without dropping or reducing support for older content types.

One benefit that should be obvious with regard to web-based applications is that a RESTful implementation allows a user to bookmark specific "queries" (or requests) and allows those to be conveyed to others across email, instant messages, or to be injected into wikis, etc. Thus this "representation" of a path or entry point into an application state becomes highly portable. A RESTFul web service is a simple web service implemented using HTTP and the principles of REST. Such a web service can be thought of as a collection of resources comprising three aspects:

1. The URI for the web service
2. The MIME type of the data supported by the web service (often JSON, XML, or YAML, but can be anything)
3. The set of operations supported by the web service using HTTP methods, including but not limited to POST, GET, PUT, and DELETE

Members of the collection are addressed by ID using URIs of the form <baseURI>/<ID>. The ID can be any unique identifier. For example, a RESTFul web service representing a collection of cars for sale might have the URI:

```
http://example.com/resources/cars
```

If the service uses the car registration number as the ID, then a particular car might be present in the collection as

```
http://example.com/resources/cars/yxz123
```

SOAP

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on XML as its message format and usually relies on other application-layer protocols, most notably Remote Procedure Call (RPC) and HTTP for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework on which web services can be built.

As a simple example of how SOAP procedures can be used, a SOAP message can be sent to a web service-enabled web site—for example, a house price database—with the parameters needed for a search. The site returns an XML-formatted document with the resulting data (prices, location, features, etc). Because the data is returned in a standardized machine-parseable format, it may be integrated directly into a third-party site.

The SOAP architecture consists of several layers of specifications for message format, message exchange patterns (MEPs), underlying transport protocol bindings, message processing models, and protocol extensibility. SOAP is the successor of XML-RPC. SOAP makes use of an Internet application-layer protocol as a transport protocol. Critics have argued that this is an abuse of such protocols, as it is not their intended purpose and therefore not a role they fulfill well. Proponents of SOAP have drawn analogies to successful uses of protocols at various levels for tunneling other protocols.

Both SMTP and HTTP are valid application-layer protocols used as transport for SOAP, but HTTP has gained wider acceptance because it works well with today's Internet infrastructure; specifically, HTTP works well with network firewalls. SOAP may also be used over HTTPS (which is the same protocol as HTTP at the application level, but uses an encrypted transport protocol underneath) with either simple or mutual authentication; this is the advocated WS-I method to provide web service security as stated in the WS-I Basic Profile 1.1. This is a major advantage over other distributed protocols such as GIOP/IIOP or DCOM, which are normally filtered by firewalls. XML was chosen as the standard message format because of its widespread use by major corporations and open source development efforts. Additionally, a wide variety of freely available tools significantly eases the transition to a SOAP-based implementation.

Advantages of using SOAP over HTTP are that SOAP allows for easier communication through proxies and firewalls than previous remote execution technology. SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols are also usable (e.g., SMTP). SOAP is platform-independent, language-independent, and it is simple and extensible.

Because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as CORBA (Common Object Request Broker Architecture). This may not be an issue when only small messages are sent. To improve performance for the special case of XML with embedded binary objects, Message Transmission Optimization Mechanism was introduced. When relying on HTTP as a transport protocol and not using WS-Addressing or an ESB, the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other. Developers must use polling instead of notification in these common cases.

Most uses of HTTP as a transport protocol are made in ignorance of how the operation is accomplished. As a result, there is no way to know whether the method used is appropriate to the operation. The REST architecture has become a web service alternative that makes appropriate use of HTTP's defined methods.

7.5.5 Communications (HTTP, SIMPLE, and XMPP)

HTTP is a request/response communications standard based on a client/server model. A client is the end user, the server is the web site. The client making a HTTP request via a web browser or other tool sends the request to the server. The responding server is called the origin server. HTTP is not constrained to use TCP/IP and its supporting layers, although this is its most popular application on the Internet. SIMPLE, the Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions, is an instant messaging (IM) and presence protocol suite based on Session Initiation Protocol, and it is managed by the IETF. Like XMPP, SIMPLE is an open standard. Extensible Messaging and Presence Protocol (XMPP) is also an open, XML-based protocol originally aimed at near-real-time, extensible instant messaging and presence information (e.g., buddy lists) but now expanded into the broader realm of message-oriented middleware. All of these protocols are discussed in detail in the following paragraphs.

Hypertext Transfer Protocol (HTTP)

HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems. Its use for retrieving linked resources led to the establishment of the World Wide Web. HTTP development was coordinated by the World Wide Web Consortium and the Internet Engineering Task Force, culminating in the publication of a series of Requests for Comments, most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use today.

HTTP is a request/response standard between a client and a server. A client is the end-user, the server is the web site. The client making a HTTP request—using a web browser, spider, or other end-user tool—is referred to as the user agent. The responding server—which stores or creates resources such as HTML files and images—is called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels. HTTP is not constrained to using TCP/IP and its supporting layers, although this is its most popular application on the Internet. In fact, HTTP can be implemented on top of any other protocol; all it requires is reliable transport, so any protocol, on the Internet or any other network, that provides reliable transport can be used.

Typically, an HTTP client initiates a request. It establishes a TCP connection to a particular port on a host (port 80 by default). An HTTP server listening on that port waits for the client to send a request message. Upon receiving the request, the server sends back a status line such as "HTTP/1.1 200 OK" and a message of its own, the body of which is perhaps the requested resource, an error message, or some other information. Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs or, more specifically, Uniform Resource Locators, URLs) using the http: or https URI schemes.

SIMPLE

Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is an instant messaging (IM) and presence protocol suite based on the Session Initiation Protocol (SIP). Like XMPP, SIMPLE is an open standard. SIMPLE makes use of SIP for registering for presence information and receiving notifications when presence-related events occur. It is also used for sending short messages and managing a session of real-time messages between two or more participants. Implementations of the SIMPLE-based protocols can be found in SIP softphones and also hardphones.[6] The SIMPLE presence specifications can be broken up into core

6. In computing, a softphone is a software program for making telephone calls over the Internet using a general-purpose computer; a hardphone is a conventional telephone set.

protocol methods, presence information, and the handling of privacy, policy. and provisioning.

The core protocol methods provide SIP extensions for subscriptions, notifications, and publications. The methods used, **subscribe** and **notify,** are defined in RFC 3265. **Subscribe** allows a user to subscribe to an event on a server. **Notify** is the method used whenever the event arises and the server responds back to the subscriber. Another standard, RFC 3856, defines precisely how to use these methods to establish and maintain presence. Presence documents contain information encoded using XML. These documents are transported in the bodies of SIP messages.[7] Privacy, policy, and provisioning information is needed by user agents to determine who may subscribe to presence information. A framework for authorization policies controlling access to application-specific data is defined in RFC 4745 and RFC 5025. SIP defines two modes of instant messaging, the Page mode and the Session mode. Page mode makes use of the SIP method MESSAGE, as defined in RFC 3428. This mode establishes no sessions, while the Session mode based on the Message Session Relay Protocol (RFC 4975, RFC 4976) defines text-based protocol for exchanging arbitrarily sized content of any time between users.

XMPP

Extensible Messaging and Presence Protocol (XMPP) is an XML-based protocol used for near-real-time, extensible instant messaging and presence information. XMPP remains the core protocol of the Jabber Instant Messaging and Presence technology. Jabber provides a carrier-grade, best-in-class presence and messaging platform. According to a press release following its acquisition by Cisco Systems in November 2008, "Jabber's technology leverages open standards to provide a highly scalable architecture that supports the aggregation of presence information across different devices, users and applications. The technology also enables collaboration across many different presence systems such as Microsoft Office Communications Server, IBM Sametime, AOL AIM, Google and Yahoo!"

Built to be extensible, the XMPP protocol has grown to support features such as voice-over-IP and file transfer signaling. Unlike other instant messaging protocols, XMPP is an open standard. Like email, anyone who has a domain name and an Internet connection can run the Jabber server

7. RFC 3863 and RFC 4479 describe this procedure.

and chat with others. The Jabber project is open source software, available from Google at http://code.google.com/p/jabber-net.

XMPP-based software is deployed on thousands of servers across the Internet. The Internet Engineering Task Force has formalized XMPP as an approved instant messaging and presence technology under the name XMPP, and the XMPP specifications have been published as RFC 3920 and RFC 3921. Custom functionality can be built on top of XMPP, and common extensions are managed by the XMPP Software Foundation.

XMPP servers can be isolated from the public Jabber network, and robust security (via SASL and TLS) is built into the core XMPP specifications. Because the client uses HTTP, most firewalls allow users to fetch and post messages without hindrance. Thus, if the TCP port used by XMPP is blocked, a server can listen on the normal HTTP port and the traffic should pass without problems. Some web sites allow users to sign in to Jabber via their browser. Furthermore, there are open public servers, such as www.jabber80.com, which listen on standard http (port 80) and https (port 443) ports and allow connections from behind most firewalls.

7.6 Standards for Security

Security standards define the processes, procedures, and practices necessary for implementing a security program. These standards also apply to cloud-related IT activities and include specific steps that should be taken to ensure a secure environment is maintained that provides privacy and security of confidential information in a cloud environment. Security standards are based on a set of key principles intended to protect this type of trusted environment. Messaging standards, especially for security in the cloud, must also include nearly all the same considerations as any other IT security endeavor. The following protocols, while not exclusively specific to cloud security, merit coverage here. In the next few sections, we explain what they are and how they are used in the cloud environment.

7.6.1 Security (SAML OAuth, OpenID, SSL/TLS)

A basic philosophy of security is to have layers of defense, a concept known as *defense in depth*. This means having overlapping systems designed to provide security even if one system fails. An example is a firewall working in conjunction with an intrusion-detection system (IDS). Defense in depth provides security because there is no single point of failure and no single-entry vector at which an attack can occur. For this reason, a choice between implementing network security in the middle part of a network (i.e., in the cloud) or at the endpoints is a false dichotomy.[8]

No single security system is a solution by itself, so it is far better to secure all systems. This type of layered security is precisely what we are seeing develop in cloud computing. Traditionally, security was implemented at the endpoints, where the user controlled access. An organization had no choice except to put firewalls, IDSs, and antivirus software inside its own network. Today, with the advent of managed security services offered by cloud providers, additional security can be provided inside the cloud.

Security Assertion Markup Language (SAML)

SAML is an XML-based standard for communicating authentication, authorization, and attribute information among online partners. It allows businesses to securely send assertions between partner organizations regarding the identity and entitlements of a principal. The Organization for the Advancement of Structured Information Standards (OASIS) Security Services Technical Committee is in charge of defining, enhancing, and maintaining the SAML specifications.[9] SAML is built on a number of existing standards, namely, SOAP, HTTP, and XML. SAML relies on HTTP as its communications protocol and specifies the use of SOAP (currently, version 1.1). Most SAML transactions are expressed in a standardized form of XML. SAML assertions and protocols are specified using XML schema. Both SAML 1.1 and SAML 2.0 use digital signatures (based on the XML Signature standard) for authentication and message integrity. XML encryption is supported in SAML 2.0, though SAML 1.1 does not have encryption capabilities. SAML defines XML-based assertions and protocols, bindings, and profiles. The term SAML Core refers to the general syntax and semantics of SAML assertions as well as the protocol used to request and transmit those assertions from one system entity to another. SAML protocol refers to what is transmitted, not how it is transmitted. A SAML binding determines how SAML requests and responses map to standard messaging protocols. An important (synchronous) binding is the SAML SOAP binding.

SAML standardizes queries for, and responses that contain, user authentication, entitlements, and attribute information in an XML format.

8. Bruce Schnier, http://www.schneier.com/blog/archives/2006/02/security_in_the.html, 15 Feb 2006, retrieved 21 Feb 2009.
9. The reader is encouraged to consult http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

This format can then be used to request security information about a principal from a SAML authority. A SAML authority, sometimes called the asserting party, is a platform or application that can relay security information. The relying party (or assertion consumer or requesting party) is a partner site that receives the security information. The exchanged information deals with a subject's authentication status, access authorization, and attribute information. A subject is an entity in a particular domain. A person identified by an email address is a subject, as might be a printer.

SAML assertions are usually transferred from identity providers to service providers. Assertions contain statements that service providers use to make access control decisions. Three types of statements are provided by SAML: authentication statements, attribute statements, and authorization decision statements. SAML assertions contain a packet of security information in this form:

```
<saml:Assertion A...>
     <Authentication>
     ...
     </Authentication>
     <Attribute>
     ...
     </Attribute>
     <Authorization>
     ...
     </Authorization>
</saml:Assertion A>
```

The assertion shown above is interpreted as follows:

```
Assertion A, issued at time T by issuer I, regarding subject
S, provided conditions C are valid.
```

Authentication statements assert to a service provider that the principal did indeed authenticate with an identity provider at a particular time using a particular method of authentication. Other information about the authenticated principal (called the authentication context) may be disclosed in an authentication statement. An attribute statement asserts that a subject is associated with certain attributes. An attribute is simply a name–value pair. Relying parties use attributes to make access control decisions. An authorization decision statement asserts that a subject is permitted to perform action A on resource R given evidence E. The expressiveness of authorization decision statements in SAML is intentionally limited.

A SAML protocol describes how certain SAML elements (including assertions) are packaged within SAML request and response elements. It provides processing rules that SAML entities must adhere to when using these elements. Generally, a SAML protocol is a simple request–response protocol. The most important type of SAML protocol request is a query. A service provider makes a query directly to an identity provider over a secure back channel. For this reason, query messages are typically bound to SOAP. Corresponding to the three types of statements, there are three types of SAML queries: the authentication query, the attribute query, and the authorization decision query. Of these, the attribute query is perhaps most important. The result of an attribute query is a SAML response containing an assertion, which itself contains an attribute statement.

Open Authentication (OAuth)

OAuth is an open protocol, initiated by Blaine Cook and Chris Messina, to allow secure API authorization in a simple, standardized method for various types of web applications. Cook and Messina had concluded that there were no open standards for API access delegation. The OAuth discussion group was created in April 2007, for the small group of implementers to write the draft proposal for an open protocol. DeWitt Clinton of Google learned of the OAuth project and expressed interest in supporting the effort. In July 2007 the team drafted an initial specification, and it was released in October of the same year.

OAuth is a method for publishing and interacting with protected data. For developers, OAuth provides users access to their data while protecting account credentials. OAuth allows users to grant access to their information, which is shared by the service provider and consumers without sharing all of their identity. The Core designation is used to stress that this is the baseline, and other extensions and protocols can build on it.

By design, OAuth Core 1.0 does not provide many desired features (e.g., automated discovery of endpoints, language support, support for XML-RPC and SOAP, standard definition of resource access, OpenID integration, signing algorithms, etc.). This intentional lack of feature support is viewed by the authors as a significant benefit. The Core deals with fundamental aspects of the protocol, namely, to establish a mechanism for exchanging a user name and password for a token with defined rights and to provide tools to protect the token. It is important to understand that security and privacy are not guaranteed by the protocol. In fact, OAuth by itself *provides no privacy at all* and depends on other protocols such as SSL to accomplish that. OAuth can be implemented in a secure manner, however. In fact, the specification includes substantial security considerations that must be taken into account when working with sensitive data. With Oauth, sites use tokens coupled with shared secrets to access resources. Secrets, just like passwords, must be protected.

OpenID

OpenID is an open, decentralized standard for user authentication and access control that allows users to log onto many services using the same digital identity. It is a single-sign-on (SSO) method of access control. As such, it replaces the common log-in process (i.e., a log-in name and a password) by allowing users to log in once and gain access to resources across participating systems.

The original OpenID authentication protocol was developed in May 2005 by Brad Fitzpatrick, creator of the popular community web site LiveJournal. In late June 2005, discussions began between OpenID developers and other developers from an enterprise software company named NetMesh. These discussions led to further collaboration on interoperability between OpenID and NetMesh's similar Light-Weight Identity (LID) protocol. The direct result of the collaboration was the Yadis discovery protocol, which was announced on October 24, 2005.

The Yadis specification provides a general-purpose identifier for a person and any other entity, which can be used with a variety of services. It provides a syntax for a resource description document identifying services available using that identifier and an interpretation of the elements of that document. Yadis discovery protocol is used for obtaining a resource description document, given that identifier. Together these enable coexistence and interoperability of a rich variety of services using a single identifier. The identifier uses a standard syntax and a well-established namespace and requires no additional namespace administration infrastructure.

An OpenID is in the form of a unique URL and is authenticated by the entity hosting the OpenID URL. The OpenID protocol does not rely on a central authority to authenticate a user's identity. Neither the OpenID protocol nor any web sites requiring identification can mandate that a specific type of authentication be used; nonstandard forms of authentication such as smart cards, biometrics, or ordinary passwords are allowed. A typical scenario for using OpenID might be something like this: A user visits a web site that displays an OpenID log-in form somewhere on the page. Unlike a typical log-in form, which has fields for user name and password, the OpenID log-in form has only one field for the OpenID identifier (which is an OpenID URL). This form is connected to an implementation of an OpenID client library. A user will have previously registered an OpenID identifier with an OpenID identity provider. The user types this OpenID identifier into the OpenID log-in form.

The relying party then requests the web page located at that URL and reads an HTML link tag to discover the identity provider service URL. With OpenID 2.0, the client discovers the identity provider service URL by requesting the XRDS document (also called the Yadis document) with the content type **application/xrds+xml,** which may be available at the target URL but is always available for a target XRI. There are two modes by which the relying party can communicate with the identity provider: **checkid_immediate** and **checkid_setup.** In **checkid_immediate,** the relying party requests that the provider not interact with the user. All communication is relayed through the user's browser without explicitly notifying the user. In **checkid_setup,** the user communicates with the provider server directly using the same web browser as is used to access the relying party site. The second option is more popular on the web.

To start a session, the relying party and the identity provider establish a shared secret—referenced by an associate handle—which the relying party then stores. Using **checkid_setup,** the relying party redirects the user's web browser to the identity provider so that the user can authenticate with the provider. The method of authentication varies, but typically, an OpenID identity provider prompts the user for a password, then asks whether the user trusts the relying party web site to receive his or her credentials and identity details. If the user declines the identity provider's request to trust the relying party web site, the browser is redirected to the relying party with a message indicating that authentication was rejected. The site in turn refuses to authenticate the user. If the user accepts the identity provider's request to trust the relying party web site, the browser is redirected to the designated return page on the relying party web site along with the user's credentials. That relying party must then confirm that the credentials really came from the identity provider. If they had previously established a shared secret, the relying party can validate the shared secret received with the credentials against the one previously stored. In this case, the relying party is considered to be stateful, because it stores the shared secret between sessions (a process sometimes referred to as persistence). In comparison, a stateless relying party must make background requests using the **check_authentication** method to be sure that the data came from the identity provider.

After the OpenID identifier has been verified, OpenID authentication is considered successful and the user is considered logged in to the relying party web site. The web site typically then stores the OpenID identifier in the user's session. OpenID does not provide its own authentication methods, but if an identity provider uses strong authentication, OpenID can be used for secure transactions.

SSL/TLS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographically secure protocols designed to provide security and data integrity for communications over TCP/IP. TLS and SSL encrypt the segments of network connections at the transport layer. Several versions of the protocols are in general use in web browsers, email, instant messaging, and voice-over-IP. TLS is an IETF standard protocol which was last updated in RFC 5246.

The TLS protocol allows client/server applications to communicate across a network in a way specifically designed to prevent eavesdropping, tampering, and message forgery. TLS provides endpoint authentication and data confidentiality by using cryptography. TLS authentication is one-way—the server is authenticated, because the client already knows the server's identity. In this case, the client remains unauthenticated. At the browser level, this means that the browser has validated the server's certificate—more specifically, it has checked the digital signatures of the server certificate's issuing chain of Certification Authorities (CAs).

Validation does not identify the server to the end user. For true identification, the end user must verify the identification information contained in the server's certificate (and, indeed, its whole issuing CA chain). This is the only way for the end user to know the "identity" of the server, and this is the only way identity can be securely established, verifying that the URL, name, or address that is being used is specified in the server's certificate. Malicious web sites cannot use the valid certificate of another web site because they have no means to encrypt the transmission in a way that it can be decrypted with the valid certificate. Since only a trusted CA can embed a URL in the certificate, this ensures that checking the apparent URL with the URL specified in the certificate is an acceptable way of identifying the site.

TLS also supports a more secure bilateral connection mode whereby both ends of the connection can be assured that they are communicating with whom they believe they are connected. This is known as mutual (assured) authentication. Mutual authentication requires the TLS client-side to also maintain a certificate. TLS involves three basic phases:

1. Peer negotiation for algorithm support
2. Key exchange and authentication
3. Symmetric cipher encryption and message authentication

During the first phase, the client and server negotiate cipher suites, which determine which ciphers are used; makes a decision on the key exchange and authentication algorithms to be used; and determines the message authentication codes. The key exchange and authentication algorithms are typically public key algorithms. The message authentication codes are made up from cryptographic hash functions. Once these decisions are made, data transfer may begin.

7.7 Chapter Summary

In this chapter we have discussed some of the more prevalent standards used in cloud computing. Although we have not analyzed each standard in depth, you should now have a feel for how and why each standard is used and, more important, an understanding of why they have evolved. Standards are important, to be sure, but most of these standards evolved from individuals taking a chance on a new innovation. As these innovative techniques became acceptable to users and implementers, more support for the technique followed. At some point, enough support was present to make the innovation be considered a “standard,” and groups formalized protocols or rules for using it. Such a “standard” is used until more new innovation takes us elsewhere.