

# Computer

Innovative Technology for Computer Professionals

MARCH 2011

<http://www.computer.org>

## CLOUD COMPUTING

FACE RECOGNITION TECHNOLOGY, P. 84

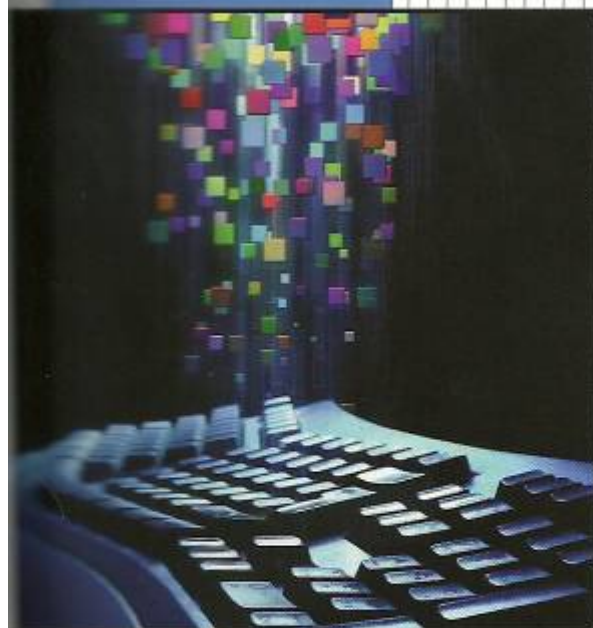
WIKIPEDIA: PAST, PRESENT, AND FUTURE, P. 87

RESOURCE ACCOUNTING, P. 91



IEEE  
computer  
society





# Building a Cloud Computing Platform for New Possibilities

Yousef A. Khalidi, Microsoft

**The cloud represents a drastic shift, not only in terms of the new economics it brings to bear, but also in terms of the new possibilities it enables. Realizing the full benefits of this new paradigm requires rethinking the way we build applications.**

**W**hen cars emerged in the early 20th century, they were initially called horseless carriages, an indication that people defined this new concept in the context of existing technology. Incredibly, automotive engineers even designed whip holders into the first models before realizing they were no longer necessary.

Early designers failed, at first, to understand the possibilities of the new paradigm, such as building for higher speeds or greater safety. When Karl Benz, arguably the inventor of the automobile, attempted to estimate the long-term market opportunity, he concluded there could never be more than 1 million cars because of their high cost and the shortage of capable chauffeurs.<sup>1</sup>

But by the 1920s, the number of cars had already reached 8 million, and today more than 600 million are on the road, proving Benz wrong hundreds of times over. What he and other early pioneers failed to realize was that profound reductions in both cost and complexity, plus a dramatic increase in the car's importance to daily life, would overwhelm prior constraints and bring automobiles to the masses.

Today, computing is going through a similar change with the shift from client/server architectures to cloud computing. The cloud promises not just cheaper computing but also that it will be faster, easier, more flexible, and more effective.

Many formerly impractical things suddenly become very practical. Before cars were widely adopted, for example, it was possible, but impractical, for people to live miles away from the city where they worked. Now employees routinely commute over what would have been unrealistic distances, creating new possibilities for business and economic development outside the confines of the urban landscape.

Similarly, the emergence of cloud computing should have a profound effect on the way people think about the possibilities of computing. When it's possible to consume a platform-as-a-service (PaaS) solution from any datacenter in any location, how will that redefine the computing landscape?

It's helpful, of course, to understand what such a platform would look like.

## DESIGN PRINCIPLES

Building a platform that delivers on cloud computing's full promise requires defining what success looks like. Most people in the industry agree on the cloud's two primary benefits: increasing agility and reducing costs. But as in most engineering challenges, the devil is in the details.

To increase developer agility, a cloud platform should provide a rich set of services, such as storage, queuing, and identity services that simplify and accelerate the program-



## PUTTING PRINCIPLES INTO PRACTICE: WINDOWS AZURE

**W**indows Azure ([www.microsoft.com/windowsazure](http://www.microsoft.com/windowsazure)) is Microsoft's platform-as-a-service offering. It includes an operating system and a set of developer services that provide the functionality to build applications spanning the consumer Web to enterprise scenarios. Windows Azure and SQL Azure are this platform's key components.

Windows Azure provides developers with on-demand compute, storage, networking, content delivery, and middleware capabilities to host, scale, and manage applications on the Internet through Microsoft datacenters. To build applications and services on this platform, developers can use their existing expertise with standard programming tools. Azure supports popular standards, protocols, languages, and development environments, including SOAP, REST, XML, .NET Framework, C#, Java, PHP, and Ruby.

The Microsoft SQL Azure database is a cloud-based relational database service built on SQL Server technologies. It's a highly available, scalable, multitenant database service hosted by Microsoft in the cloud. Customers can use existing knowledge in T-SQL development and a familiar relational data model for symmetry with existing onsite databases. Additionally, customers can quickly become productive on SQL Azure by using the same development and management tools they use for onsite databases.

Windows Azure is designed to abstract away much of the infrastructure that typically underlies applications, allowing developers to focus on building applications. It provides a virtualized compute and network infrastructure, and each physical server runs a copy of a hypervisor that divides it into a set of isolated virtual machines. One virtual machine runs the trusted components used by the platform to manage the other virtual machines. The trusted virtual machine also runs network virtualization software that includes a packet filter to ensure that untrusted virtual machines can't generate spoofed traffic or receive traffic not addressed to them.

All services running on the base platform treat each machine as a cache of programs and state. Services expect and can handle failures by storing and retrieving their data from other nodes, such as SQL

Azure or the underlying Windows Azure storage system. The Fabric Controller (FC) platform component manages and automates hardware and application life cycles. A highly redundant core component of the platform that maintains a repository of all hardware and software assets for a given hardware cluster, the FC is responsible for two main functions:

- *Managing hardware assets.* The FC keeps track of a given cluster's physical inventory; it's responsible for booting the correct software images on each machine, configuring the network infrastructure, and reacting to and recovering from hardware failures.
- *Managing software assets.* The FC has a repository of all applications running on the cluster, including the desired configuration and resource needs for each application. Based on its knowledge of hardware assets, the FC maps the desired application needs to available hardware resources. It configures the various hardware and platform elements such as packet filters as needed. The FC cooperates with applications to recover from failures automatically, and it aids in application life-cycle management functions such as rolling upgrades.

The FC carries out its functions based on developer-supplied application models. In essence, the FC maps declarative application specifications to available hardware resources. The application model includes each application's topology, pointers to required software and related configuration settings, and a description of the application resource needs. The FC attains and maintains the desired state for each application over the shared hardware infrastructure.

Finally, Windows Azure provides a set of services and related development and management tools, including the SQL Azure database service, a storage service, monitoring and logging services, metering and billing services, and a rich developer environment with support for Microsoft's Visual Studio and Eclipse.

mer's job. A rich set of highly available platform services not only increases programmer productivity but can also result in more efficient sharing of services among different applications.

Another important requirement is to free the programmer from dealing with mundane, low-level issues such as hardware configuration, network management, or operating system (OS) patches. Application deployment should be simple and quick. For example, changing the application structure and the communication pattern between various application components shouldn't require tinkering with the network—the programmer should be free from such details.

Similarly, application managers must be able to respond to load changes, frequent software changes, and configuration updates quickly and reliably. They shouldn't spend their time allocating hardware, booting machines, configuring OS images, or managing low-level network equipment. In traditional IT organizations, these activities are time-consuming and distract from the main goal of managing the application.

As for reducing costs, an essential element is automation, which should include everything from server allocation to OS deployment to application life-cycle management (deploying, restarting from failures, and so on). Human intervention should be kept to a minimum. Anything that can be automated should be automated—including the anticipation and mitigation of software and hardware component failures.

Another cost reduction technique is to use standard server hardware with standard networking and to avoid specialized setups, such as separate storage area networks or expensive high-end systems. Attaining the goal of resource elasticity requires sharing large pools of virtually uniform hardware. Automating the management of these large hardware pools further reduces costs.

Finally, to maximize both agility and cost reduction, the application must be written to scale out horizontally, with a message-passing style of communication between its various components. Capacity is added or removed as



remote compute units, instead of adding memory and CPUs to each server.

Given these requirements for realizing the cloud's benefits, the fully optimized cloud platform should have the following architectural features:

- a virtualized compute fabric, in which each compute server runs a hypervisor that can allocate resources efficiently and dynamically and isolate them into virtual machines;
- a virtualized network that securely enables collections of virtual machines to be grouped together and isolates unwanted traffic from the rest of the shared hardware pool;
- data, software, and other state at each server node viewable as a cache;
- a scale-out model, in which failures are expected and state can be reconstructed from elsewhere in the network, such as from other nodes running a storage service;
- automation for virtualized compute fabrics, virtualized networks, OS deployment and updates, application deployments, failure handling and application recovery, and application updates;
- a metamodel that lets the platform automate application life-cycle management; and
- rich platform services, accessible through network protocols.

At a minimum, the cloud platform should provide reliable and scalable storage services, including transactional databases, monitoring and logging services, identity services, and metering and billing services.

## HYBRID SOLUTIONS

Many organizations have data and applications that are impractical to move to the cloud today for myriad reasons: compliance, legal issues, internal controls, economics, and so on. In some cases, the costs of moving—financial and otherwise—simply overwhelm the benefits. Naturally, over time, the number of things that are technically impractical to move will diminish, but practice always lags behind technology.

At this inflection point, it's important to recognize the necessity of hybrid models, which focus on aspects of applications that can move to the cloud sooner than others. It wouldn't be easy to move an entire ERP system, for example, because it shares sensitive data with other onsite applications. However, it's possible to split an online shopping site so that its Web-facing front end goes to the cloud, but its check-out process triggers a secure connection at the datacenter.

The criteria for deciding which workloads or applications to move to the cloud today fall into the following general categories:

- *Business case.* For a given workload and application, is it more economical to move to the cloud? Will the increased agility and elasticity overcome the cost of moving the application? Will moving to the cloud enable new business scenarios?
- *Technical readiness.* Is the application "cloud ready"? The application must retain data in a durable replicated store or replicate it. State in general can be stored locally, but any instance is only a cache. The application shouldn't require lengthy installation or configuration, or any human intervention to start. It must scale horizontally by sharing load across multiple instances (scale out), not vertically within an ever-bigger single instance (scale up).
- *Dependencies.* The application must run without special hardware needs; any dependencies on other applications or data must be addressed before moving to the cloud. Communication latency between the ap-

## All dependencies and regulatory requirements must also be addressed before moving an application.

plication and dependent onsite resources must also be addressed.

- *Regulation, compliance, and legal considerations, particularly with respect to data.* A public cloud might not be able to host some data due to regulatory issues, for example.

Considering and resolving the core technical issues of making the application "cloud ready" are necessary but not sufficient steps. All dependencies and regulatory requirements must also be addressed before moving an application. Many cases might require careful decomposition, keeping some part of the application onsite and the remainder in the cloud.

## EMBRACING A NEW APPLICATION MODEL

PaaS solutions provide a complete application development and hosting site delivered as a cloud service.<sup>2</sup>

In addition to managing the underlying infrastructure and offering a metered-by-use cost model, PaaS also facilitates application development, testing, deployment, and ongoing maintenance, liberating the customer to focus on managing the application instead of the underlying infrastructure. Customers can write their applications directly to the PaaS platform and then elastically scale out as and when needed. To make this possible, the PaaS programming model imposes some rules on applications:



## ADOPTION SCENARIOS

**A**lthough the Windows Azure platform has only been commercially available since January 2010, some common adoption scenarios are already emerging:

- *Web extensions.* Industries such as manufacturing, transportation and logistics, and professional services increasingly rely on interactions with their own distributed sites and suppliers to ensure business success. To enable better interaction and collaboration, customers can use the platform to add Web functionality to existing business databases and applications.
- *Marketing sites.* Short-term marketing campaigns require bursts of extra capacity. Using this platform not only makes these requirements for temporary capacity affordable, it also enables scenarios that would otherwise be cost-prohibitive due to the massive computing and related capital investment costs.
- *Departmental applications.* Business teams in media and entertainment, consumer goods, and distribution and retail rely on internal applications to perform functions such as developing segmentation models and planning tools. These applications require a database and a browser- or Excel-based UI with a business rules engine. Using this platform also enables consolidation of existing departmental databases, such as Access, MySQL, and SQL Express into a more managed and reliable cloud environment.
- *High-performance computing applications.* HPC workloads use pools of computing power to solve advanced computational equations or problems. In HPC scenarios, the "application" is typically a complex mathematical calculation. The most common industries for these applications are financial services, life sciences, media and entertainment, and manufacturing.
- *Data collaboration and content distribution.* The Windows Azure platform makes it possible to efficiently store and distribute massive amounts of data and digital content such as census data, a research catalog, or when multiple parties reference, share, and collaborate on updates to inventory data. In this scenario, the information is managed centrally, with many users accessing it on a consistent basis. Similarly, customers might need to distribute large amounts of data to a network of users but on a less frequent basis.

We expect to see a broader set of common adoption scenarios emerge as the natural lag between practice and possibility diminishes.

- A PaaS application must comprise one or more roles. Every application can be broken down into logical divisions; a role is a formalization of these divisions. It includes a specific set of code and defines the environment in which the code runs. The different types of roles can be optimized for specific functions: front end, back end, or otherwise.
- A PaaS application should run two or more instances of each role. These instances are interchangeable, and every role instance runs the exact same code. The expectation is that an instance shouldn't save

any data locally—such as session state—that would make it unique.

- A PaaS application provides a description of its topology to the platform, along with its resource requirements. This topology includes the application's roles, the connectivity needs among these roles, and each node's required number of instances. In turn, the platform uses this description to automatically allocate the necessary hardware resources, configure the hardware and software components, monitor the application, and, when necessary, recover failed components.
- A PaaS application behaves correctly when any role instance fails. In case of failure, users aren't affected as they continue to be served through the online instances. Any instance-maintained data or state must be treated only as a cache, and after a failure or a restart, the application must be able to reconstruct the state from elsewhere. Although application performance might be affected, the application will continue to run.

The degree to which these rules are enforced might vary, but PaaS offerings do assume that applications obey all these rules. Adopting the PaaS programming model as well as these rules provides developers with many benefits in the areas of management, availability, and scalability.

In PaaS offerings, the platform itself automatically handles most administrative tasks, such as applying OS patches, installing new versions of system software, and configuring the network. The goal here is to reduce the effort involved in administering the application environment.

## Availability

PaaS programming models help improve application availability in several ways:

- *Protect against hardware failures.* Because every application comprises multiple instances of each role, hardware failures—a disk crash, network fault, or server death—won't take down the application. To help, different instances of the same application are placed not randomly but in different fault domains. A fault domain is a set of hardware components—computers, switches, and more—that share a single point of failure. Since the platform understands the application's topology, it can effectively allocate the application across fault domains. The application might temporarily lose some instances, but it will continue to behave correctly.
- *Protect against software failures.* The system monitoring and management provided in PaaS also detects failures caused by software. The application can cooperate by monitoring and reporting internal system



health status. If the code in an instance crashes or the virtual machine in which it's running goes down, the system will restart the role. The system will also automatically reconfigure the network appropriately and optionally inform the rest of the application of the change. Although any work the instance was doing when it failed will be lost, the new instance will become part of the application as soon as it starts running.

- *Update applications with no application downtime.* Whether for routine maintenance, to install a new version, or even to revert to an older version, every application must be updated. An application built with a PaaS programming model can be updated while it's running. To allow this, different instances for each of an application's roles are placed in different update domains. When a new version of the application needs to be deployed, the platform can shut down the instances in one update domain, update the code, and then create new instances from that new code.
- *Update the OS and other supporting software with no application downtime.* The platform assumes that every application follows the four application rules, so it knows that it can shut down some of an application's instances when required, update the underlying system software, and then start new instances. By doing this in chunks, never shutting down all of a role's instances at the same time, the OS and other software can be updated beneath a continuously running application.

Whether it's planned or not, today's applications usually have downtime for OS patching, application upgrades, hardware failures, and other reasons. PaaS programming models are designed to keep applications continuously available, even in the face of software upgrades and hardware failures.

### Scalability

Applications designed to handle multiple users concurrently are a natural fit for the cloud. A limitation of traditional programming models is that they weren't explicitly designed to support Internet-scale applications. Developers can use PaaS programming models to build the scalable applications that massive cloud datacenters can support. Just as important, these programming models also let applications scale down when necessary, using just the resources they need.


PaaS programming models help developers build more scalable applications in three main ways:

- Automatically creating and maintaining a specified number of role instances. This makes application scalability quite straightforward, especially since cloud

platforms run in very large, uniform datacenters where getting the level of scalability an application needs isn't generally a problem.

- Providing a way to modify the number of executing role instances for a running application by offering a Web portal or an API to allow changing the desired number of instances for each role while an application is running.
- Providing highly available and scalable platform services, such as storage, relational database management systems (RDBMSs), caching, and networking services for applications.

These services free the developer from having to implement or manage complicated platform services. For example, to use a database in a traditional system, a database administrator must install RDBMS instances, configure the instances for high availability and backup,



**Developers can use PaaS programming models to build the scalable applications that massive cloud datacenters can support.**

and monitor the database for any errors. In a PaaS platform, the RDBMS is provided as a service—the developer needs only to declare the intention of using a database table, and the platform takes care of the rest.

In addition to the many benefits developers enjoy when PaaS abstracts away the infrastructure layer, the new model also delivers many benefits at the platform layer—namely, in the form of a rich set of services that cloud applications can simply call up. With these capabilities built into the platform, developers need not implement these services themselves, and operators need not run instances of them: the platform can share resources underneath these services, maximizing hardware utilization and driving down costs.

**W**e're in the cloud's early days. Realizing the full benefits of this new paradigm will require rethinking the way we build applications. Because it will take time for practice to catch up with technology, in the meantime we must have a disciplined and thoughtful approach to embracing hybrid models. But ultimately, this shift opens up a new dimension of possibilities in much the same way as the wide adoption of automobiles fundamentally altered 20th-century life.

If we refuse to define this new paradigm in the context of existing technology—when we break free of the whip-holder mentality—businesses and IT departments



can shift their focus from managing the technology to innovating, seizing new opportunities, and imagining new scenarios that will move their organizations forward in unprecedented ways. **C**

## References

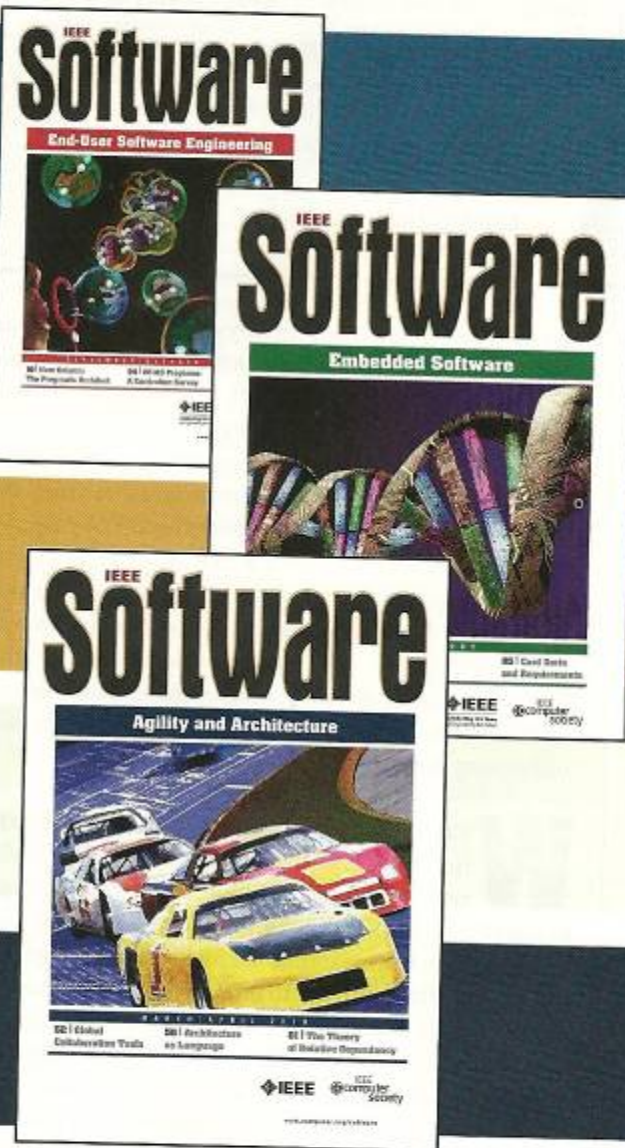
1. W. Horton, "Horseless Carriage Thinking," 1998; <http://www.designingwbt.com/content/hct/hct.pdf>.
2. P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, NIST special publication 800-145, Nat'l Inst.

Standards and Tech., 2011; [http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145\\_cloud-definition.pdf](http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf).

*Yousef A. Khalidi is a Distinguished Engineer on the Windows Azure team at Microsoft. Khalidi received a PhD in computer science from Georgia Tech and is the holder of 32 patents. Contact him at [yousef.khalidi@microsoft.com](mailto:yousef.khalidi@microsoft.com).*



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.



*IEEE Software* offers pioneering ideas, expert analyses, and thoughtful insights for software professionals who need to keep up with rapid technology change. It's the authority on translating software theory into practice.

[www.computer.org/software/SUBSCRIBE](http://www.computer.org/software/SUBSCRIBE)

# SUBSCRIBE TODAY