OPEN DATA CENTER ALLIANCE℠
DEVELOPING CLOUD-CAPABLE APPLICATIONS
WHITE PAPER REV. 1.1

Jan Drake, Arun Jacob, Nigel Simpson, Scott Thompson
TABLE OF CONTENTS
Legal Notice ........................................................................................................... 3
Introduction ........................................................................................................... 4
The Seven Keys To Cloud Application Architecture ........................................ 6
Example Cloud Application Architecture ......................................................... 7
    Figure 1. Sample Cloud Architecture ............................................................. 7
Example Cloud Deployment Architectures ....................................................... 8
    Figure 2. Single Data Center/Provider Cloud Deployment Architecture ........... 8
Multiple Data Centers/Providers ................................................................. 9
    Figure 3. Multiple Data Centers/Providers Cloud Deployment Architecture ........ 9
Cloudbursting Example .................................................................................. 10
    Figure 4. Cloudbursting Example ................................................................. 10
Recommendations ...................................................................................... 10
Conclusion ...................................................................................................... 12
Appendix A – Glossary of Terms ..................................................................... 13
LEGAL NOTICE

Copyright © 2011-2012 Open Data Center Alliance, Inc. ALL RIGHTS RESERVED.

This Open Data Center Alliance® Developing Cloud-Capable Applications White Paper is proprietary to the Open Data Center Alliance, Inc.

NOTICE TO NON-OPEN DATA CENTER ALLIANCE PARTICIPANTS: Non-ODCA participants only have the right to review, and make reference or cite, this document. Any such references or citations to this document must give the Open Data Center Alliance, Inc. full attribution and must acknowledge the Open Data Center Alliance, Inc.’s copyright in this document. Such users are not permitted to revise, alter, modify, make any derivatives of, or otherwise amend this document in any way.

NOTICE TO OPEN DATA CENTER ALLIANCE PARTICIPANTS: Use of this document by ODCA participants is subject to the Open Data Center Alliance’s bylaws and its other policies and procedures.

OPEN DATA CENTER ALLIANCESM, ODCA®SM, and ODC AllianceSM, are service marks of, and the OPEN DATA CENTER ALLIANCE logo® is a registered service mark, owned by Open Data Center Alliance, Inc. and all rights are reserved therein. Unauthorized use is strictly prohibited. THIS DOCUMENT DOES NOT GRANT ANY USER OF THIS DOCUMENT ANY RIGHTS TO USE ANY OF THE OPEN DATA CENTER ALLIANCE’S TRADEMARKS.

Any other service marks, trademarks, and trade names referenced herein are those of their respective owners.

This document and its contents are provided “AS IS” and are to be used subject to all of the limitations set forth herein.

Users of this document should not reference any initial or recommended methodology, metric, requirements, or other criteria that may be contained in this document or in any other document distributed by the Open Data Center Alliance, Inc. (“Initial Models”) in any way that implies the user and/or its products or services are in compliance with, or have undergone any testing or certification to demonstrate compliance with, any of these Initial Models.

Any proposals or recommendations contained in this document or in any other document distributed by the Open Data Center Alliance, Inc., including, without limitation, the scope and content of any proposed methodology, metric, requirements, or other criteria, do not mean the Open Data Center Alliance, Inc. will necessarily be required in the future to develop any certification or compliance or testing programs to verify any future implementation or compliance with such proposals or recommendations.

Published November, 2012
INTRODUCTION

For enterprises and consumers to realize the benefits of cloud computing a new approach to application development is required. Whether you are a CIO, a Technology Manager, an Architect, or a Developer, you could benefit from the perspective on developing cloud applications presented here.

Cloud and the Enterprise

The potential benefits for Enterprise adoption of cloud technologies are immense. The direct and related potential benefits include:

- Decreases in infrastructure costs
- Reduced time to market
- Flexibility in infrastructure investments beyond today's virtualization solutions
- Opportunity to rapidly adopt and apply game-changing technologies to the Enterprise
- Enhanced partnering opportunities due to increased business interoperability
- Full compatibility with the trend away from PCs to portable and purpose-specific devices

It is beyond the scope of this paper to address each of the above in detail; however, whether you are building an application in-house, utilizing a vendor to create bespoke software, or evaluating vendors for the cloud compatibility of their solutions, this paper will help you understand the application approaches required to leverage the cloud.

CIOs and Technology Managers Take Heed

The most important thing for CIOs and Technology Managers to realize is that every significant transition in technology has resulted in a significant rewrite or re-implementation of the technologies that your businesses depend on: simple migration is a myth. There have been many of these transitions and vendors, to various degrees have sought to shield businesses from the potentially disruptive aspects of these technology shifts. By and large, however, the reality has been that they have passed their transition costs directly to Enterprises as they have retooled.

The good news for CIOs and Technology Managers is that technology capabilities and the technology ecology around them have evolved to the point where Enterprises now have the potential to reduce the impact to the bottom line of these technology shifts. Achieving this goal requires a multi-step approach. In this paper we will focus on one important component: investing in proper application architectures (whether built or acquired) in lieu of the traditional approaches...
found in the Enterprise today. We look forward to addressing the greater picture around this in other papers. As always, anyone reading these publications is welcome to contact the ODCA and its contributors for further information.

Development Managers and Architects Take Note
The opportunity to dramatically impact the businesses that you manage and for which you create technology has been amplified by the concepts and technologies of “the cloud.” For those of you with traditional enterprise development teams yet another shift in thinking is required. We say “another” because you've been through this before. The development history of Enterprise teams is littered with stories of technology transitions and their associated complications. Our hope is that this paper will help you minimize the impact of future transitions by providing you with perspective to guide your development teams and to evaluate vendor solutions, as you construct solutions for the Enterprise that incorporate the concepts of cloud computing.

A Brief Look at The Point of Technology Transitions
Each transition in technology architectures has brought benefits, with the trend being from tightly coupled, vertical systems to loosely coupled, highly distributed systems. Generally speaking, with each transition, the Enterprise gains more options and, therefore, increased flexibility in its business, as system replacement or augmentation becomes less of a “rip and replace” and more of a “let's slot this in instead” approach. The traditional state of the art in decoupling has been Service Oriented Architecture (SOA) and what is usually referred to as the Enterprise Bus (a centralized go-between for disparate systems). When the systems that an Enterprise invests in and interacts with are distributed in nature, they are, perforce, decoupled, and can provide additional benefits such as decreased time to market, higher availability, lower Mean Time To Recovery (MTTR), and lower cost of delivery and operations at scale. Cloud application architecture takes this to a new level.

Application Delivery Strategy: Infrastructure or Platform?
One more important consideration, before we dive into cloud application architecture itself, is application delivery strategy: will you deliver your applications with Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or both?

The trade-offs between them are clear:

- IaaS provides ultimate flexibility at the cost of increased operations and deployment complexity, due to custom deployment and configuration of the infrastructure for each individual application. With this delivery approach, you design an entire operational infrastructure for each application every time.

- PaaS provides a pre-designed, operationalized, scalable, and reusable container for your applications, which dramatically decreases operations and deployment complexity. With this delivery approach, you design the operational infrastructure once and reuse instances of it for each application.

The likelihood is that your business will, to some level, need to support both. Generally, we encourage using PaaS and falling back to IaaS only when the needs of the application actually require it. Now that we've given some framing to the topic, let's move on to actual cloud application architecture.

The Architectural End Game
As you develop new applications for the cloud, architecture plays a pivotal role. Developing successfully for the cloud requires special considerations; specifically, the architecture should allow the application to run anywhere, and be composable, elastic, extensible, and evolvable.

- **Run Anywhere** means that the application is able to run on any cloud provider and on any device.

- **Composable** means that the application is designed into discrete services that can be utilized individually, as well as composed together, to create more complex applications: think “mashup.”
• **Elastic** is defined as the ability to scale up but also to scale down based on the load.

• **Extensible** is defined as the ability to easily grow the application over time.

• **Evolvable** is defined as the ability to replace existing underlying technology or vendor decisions with others, as the needs of the business and the market change, with minimal impact to the business.

**THE SEVEN KEYS TO CLOUD APPLICATION ARCHITECTURE**

1. **Everything is a Service** – Compute, network, and storage are now software services, along with cloud applications. Increasingly, vendor capabilities are being delivered in Software as a Service (SaaS) modes, rather than as on-premise solutions. Thinking of your application as a provider and consumer of lightweight web services is the first key. This will lead you to partition your application and its capabilities into more granular components that can be implemented, tested, and scaled separately. It will also lead you rapidly to the understanding that when you build a cloud application you are inherently building a distributed system.

2. **Use RESTful APIs** – When everything is a service, APIs become the building blocks and leverage points of your cloud application. Deconstructing your application into separate web service APIs enables easy re-use and publishing of your application’s capabilities for integration into other cloud apps, while also enabling you to invest in and scale appropriately each individual service. Additionally, well-designed APIs shield your application from underlying technology implementations (something we strongly recommend and address further below) as well as vendor-specific implementations. A world-class cloud application architecture is constructed from API primitives which address the simplest and most direct use-cases in order to enable flexible assembly into solutions. This should inform not only your cloud application architecture but also your evaluation of vendor solutions. Encapsulating technologies you create or purchase behind RESTful APIs enables an architecture and application that can evolve.

   Increasingly, web services are built RESTfully using JavaScript Object Notation (JSON) payloads instead of XML. This combination gives providers and consumers an easy-to-understand, easy-to-code standard that provides much higher and lighter weight interoperability than previous standards such as Simple Object Access Protocol (SOAP). We recommend using RESTful interfaces with JSON payloads as an application standard. Links to more information on these and other technologies are provided at the end of this paper.

3. **Separate Compute and Persistence** – Designing for complete separation of compute and persistence provides deployment and scaling flexibility. If we make sure our cloud applications use web services for ALL data storage (including things we normally might not consider, such as log files and debugging streams), we enable the infrastructure around them to be managed and invested in separately, shield the application from underlying changes in storage technology, and decrease deployment overhead. This means that our compute instances can be deployed, destroyed, or relocated as needed. Absolutely nothing is stored locally on the compute instance primitives running your cloud application. A classic example is an application that uses Amazon’s S3 for file storage: the application can be run from the desktop, a mobile device, a private cloud, or a public cloud and still function, provided access to S3 is available (or fail gracefully if it is not). With this in place, your application is poised to run in your cloud, public clouds, or both.

4. **Design for Failure** – We always do our best to avoid failures; however, at the same time, we must expect them to happen. Components will fail, services will become unavailable, latencies will increase, but your business must continue to run and your customers must still be served. Designing applications to survive failures in a distributed environment affects decisions everywhere in the architecture but especially in the user experience at the end-point of your application. Focus on graceful degradation and recovery when services are unavailable.
5. **Architect for Resilience** – Over the years, many products have been delivered with architectures that focus on Mean Time Between Failures (MTBF). MTBF focused architectures tend to focus on trying to reduce the frequency of errors (preferably to zero). As a result, when failures do occur, identifying and resolving them takes an unpredictable amount of time. In a service oriented world, a focus on Mean Time To Recovery (MTTR) provides greater benefits to cloud applications as such an architecture focuses on design decisions that enable the rapid identification and resolution of problems in preference to, but not replacement of, prevention. In essence, we trade an attempt at perfection for predictable recovery times.

6. **Operationalize Everything** – The entire set of services should be supportable, properly instrumented, easy to maintain, easy to troubleshoot and require the least number of people to operate. We strongly recommend adopting a DevOps model in your organization, if possible, as it provides a feedback loop that improves the quality and operational supportability of the applications being developed. Additionally, we are now in a Big Data world – instrument everything, log everything, analyze everything; you’ll learn a lot about how your application behaves in a distributed environment and how to improve it operationally.

7. **Security at Every Layer** – In the cloud, and more specifically in public clouds, the perimeter security approach, which is typically used by enterprises, is not sufficient. In a public cloud, it is not clear where your data resides, who has access to it, or how it travels; therefore, a more comprehensive approach is needed. Security is needed everywhere. Examples are encrypted transport into the cloud, secure coding and access control inside your application and encryption of data at rest. You also need to ensure that each and every API and data source is penetration tested and thoroughly analyzed.

**EXAMPLE CLOUD APPLICATION ARCHITECTURE**

The diagram below (Figure 1) shows how a simple cloud application might be constructed using a Data as a Service (DaaS) layer and Software as a Service (SaaS) to deliver functionality to a client.

![Figure 1. Sample Cloud Architecture](image-url)
**EXAMPLE CLOUD DEPLOYMENT ARCHITECTURES**

**Single Data Center/Provider**

The simple diagram below (Figure 2) shows a cloud application consisting of a couple of stateless web services utilizing a data services tier in a single data center deployment. As you’ll see in the next figure, providing complete separation of compute and persistence prepares the application for execution anywhere.

![Figure 2. Single Data Center/Provider Cloud Deployment Architecture](image-url)
MULTIPLE DATA CENTERS/PROVIDERS

In the diagram below (Figure 3), we show a high availability, disaster recovery model in a multiple data center environment with ephemeral compute instances talking to local data services built on data technologies that are intended to support replication. Note that the applications are completely unaware of the complexities of the underlying data infrastructure due to the data services layer they depend on.

Multiple Data Center Model
- Global Load Balancer Manages Request Traffic
- App Services Also Use Global Load Balancer/Load Balancer to Access Data Services
- Active/Active and Active/Failover Managed by Global Load Balancer
- High Availability w/Disaster Recovery
- Easily Add a Public Cloud Provider

Figure 3. Multiple Data Centers/Providers Cloud Deployment Architecture
CLOUDBURSTING EXAMPLE

In the diagram below (Figure 4), you can again see the flexibility gained by complete separation of compute and persistence.

Cloudbursting Model
- Extra Compute Power Rented from Third Party
- Global Load Balancer Manages Request Traffic
- Easily Add a Public Cloud Provider

Client

Global Load Balancer

Geo-Aware Load Balanced Requests

Data Center #1

Application Stack

Stateless Compute Nodes

Service A

Service B

Data Services Stack

Key/Value Store Service

Document Store Service

Cassandra/ Membase Cluster

Mongo/ Couch Cluster

Third Party Cloud Provider

Application Stack

Stateless Compute Nodes

Service B

Service A

Figure 4. Cloudbursting Example

RECOMMENDATIONS

There are many options before you as you move to a cloud ready architecture. In order to kick-start your thinking, we have provided a set of recommendations to consider when designing your cloud application.

IaaS or PaaS – As we mentioned above, when considering a specific architecture type, there is often a choice to be made between an Infrastructure as a Service (IaaS) model or Platform as a Service (PaaS) model. IaaS provides maximum flexibility but higher operational overhead, while PaaS provides less flexibility but significantly less management overhead. We believe that PaaS is a better solution in the long run.
**SaaS** – Consider using Software as a Service offerings to rapidly bootstrap your application and/or provide capabilities you don’t want or need to build. Using SaaS can be an excellent way to decrease your time to market for a reasonable investment. However, beware vendor lock-in here: look for a solution that has multiple vendors using RESTful APIs to provide their capabilities and make sure you encapsulate the integration on your side to enable you to replace your SaaS provider as necessary. Also, do not use vendor specific data structures within your own application architecture (i.e., a JSON customer object definition). It is almost always worth translating to your own internal data structures to reduce any semantic and data structure dependencies and enable a rapid transition to another vendor.

**Plan for Hybrid Clouds** – As you build your application for a cloud, it is incremental work to architect it to take advantage of the public cloud. Public clouds are rapidly becoming a commodity that may be appropriate for certain of your applications. Specifically, remember #3 above: insist on ephemeral compute instances and focus on ensuring that they don’t have any persistence on them whatsoever so you can destroy them, relocate them, or scale them without any concern about data loss or security.

**Leverage Open Source and Open Standards** – Consider using standard open source stacks and vendor solutions based on open standards wherever possible. On the open source side, take advantage of software that is used by major providers such as Facebook, Yahoo, and Google and is supported by a large talent pool of developers. On the open standards side, using vendors and products that align with an open standard increases the interoperability of the solutions you are assembling. Proper use of open source and open standards software enables you to focus on business differentiating features in your application, rather than on plumbing. The above said, each user should always analyze the legal implications of all open source applications.

**Guidelines for Vendor Evaluation** – If, instead of building, you are assembling or buying a cloud application, you can evaluate them for some of the same characteristics. Insist on RESTful interfaces for integration and automation and if possible, choose vendors that expose their full capabilities via their API and that build their product user experience on top of their API. Look for vendors that support Open Standards or use open source technology that has an active community behind it. Consider, if possible, an open source plus support model instead of proprietary software licenses, especially when adopting emerging technology, as this will reduce your risk if a vendor discontinues a product or simply disappears from the marketplace. The above said, each user should always analyze the legal implications of all open source applications.

**Create or Utilize Data Services** – This is key for separating compute and persistence. By creating data services for your applications to depend on, you shield the application from the underlying implementation, isolate and reduce the investments required for high availability and disaster recovery, and enable ephemeral compute instances, which allows your application to run anywhere. Exposing these services via RESTful interfaces provides you with the ability to evolve the underlying technologies in response to the rapidly changing data storage and analytics market.

**Use NoSQL Data Stores** – Relational databases are the default storage platform for most small to medium scale application stacks. Relational databases, which have great value when providing transaction boundaries in operations that need them, are often misused for storing and accessing large volumes of non-transactional data. In order to achieve the performance that your application will need as it scales up, you should consider the requirements of the data that you are storing, and determine whether you can leverage NoSQL data stores. NoSQL data stores are non-relational data stores – they do not provide transaction boundaries or the associated transaction rollback capabilities that relational databases do. However, they have many benefits relational databases can’t provide, such as linear scaling, and built in replication. Most of the scaling and performance pain points encountered with relational databases come from attempts to store data that really has no transactional requirements. Server Log data and user configuration data are examples of data that doesn’t need the ACID capabilities of a relational store.

Non-transactional data does not have to be immediately consistent – more value is placed on having that data available and partition tolerant, with eventual consistency being assured within milliseconds. When you can relax the need for immediate consistency that a relational database provides, you can achieve much greater scale at a much lower cost.
In order to choose the most appropriate NoSQL platform for your application, you need to understand the real consistency and latency needs of the application. Specifically, you must understand the CAP (Consistency, Availability, and Partitioning) requirements of your data, and choose a platform that meets those requirements. Please see footnotes\(^1\) for links on this.

**Log Everything** – In today’s world, we have the ability to conduct analysis on large data sets efficiently, easily and quickly. Remember that you don’t know what you don’t know and that part of the promise of Big Data is the ability to rapidly discover patterns you didn’t know were there. Typically these patterns are uncovered via standard machine learning methods that enable identification and classification of the data. The real power of being able to capture and store large volumes of log data is that you have enough raw data to run a variety of different classification experiments that will allow you to quickly and deterministically understand these patterns. The more you log, the more you can extract value from the data collected, which can enable greater insight, strong business intelligence, and things about your application you didn’t know.

**Standardize on OAuth** – OAuth\(^2\) is a security protocol that the industry is quickly adopting as a standard authorization and authentication protocol. We recommend using it both internally and externally to provide an extensible, entitlements-based, authentication and authorization system for your web services and possibly your applications themselves.

**Staff Appropriately (Dev/Ops Model)** – We strongly recommend adopting a DevOps model\(^3\), a term that combines the words “developer” and “operations.” This organizational model brings together developers and IT operations staff in the design, development, and operation of the cloud applications you bring to market. This joint working relationship ultimately leads to higher quality software by educating developers on what it takes to create an operationally supportable application, while immersing them in the experience of supporting the application directly.

**Automate Everything** – The tool chain around cloud application development and operations forms the spine in support of application delivery, deployment, evolution, and operational support. Everything from continuous integration and build management to deployment, operations, provisioning and analytics should be automated to streamline design and delivery into the cloud and to eliminate human error.

**API Standards** – Have some, better yet: Adopt formal RESTful API standards, based on industry standards, and enforce them, as this will increase the understandability and interoperability of the services you take to the cloud. We have included a few links to relevant articles in the footnotes and would also direct you to the “REST API Design Rulebook” by Mark Masse, which will be published shortly and is a more formal addressing of the topic based on standards work done by an ODCA member organization.\(^4\)

**CONCLUSION**

We are in the midst of another technology transition, due to the concepts and capabilities of cloud technology: a transition to a truly distributed technology world for developers and businesses alike.

Technology managers, architects, and developers have the opportunity to deliver on the promise of this transition by adopting new approaches and techniques that provide leverage for themselves and their businesses.

We hope the information above serves as a starting point for your exploration and adoption of new application architectures within the cloud.

---


\(^2\) See glossary for definition

\(^3\) [http://dev2ops.org/blog/2010/11/7/devops-is-not-a-technology-problem-devops-is-a-business-prob.html](http://dev2ops.org/blog/2010/11/7/devops-is-not-a-technology-problem-devops-is-a-business-prob.html)

APPENDIX A — GLOSSARY OF TERMS

API — An abbreviation of “Application Program Interface,” it’s a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer then puts the blocks together. Most operating environments, such as MS-Windows, provide an API so that programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. This makes it easier for users to learn new programs (Webopedia).

Data store — A permanent storehouse of data. The term is often used to lump the storage of all types of data structures (files, databases, text documents, etc.) into one generic category (PC Magazine Encyclopedia).

“DevOps” is an emerging set of principles, methods and practices for communication, collaboration and integration between software development (application/software engineering) and IT operations (systems administration/infrastructure) professionals. It has developed in response to the emerging understanding of the interdependence and importance of both the development and operations disciplines in meeting an organization’s goal of rapidly producing software products and services. (en.wikipedia.org/wiki/DevOps)

Hybrid cloud – The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds). (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

IaaS – Stands for “Infrastructure as a Service.” The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

NoSQL data store – In computing, NoSQL (sometimes expanded to “not only SQL”) is a broad class of database management systems that differ from the classic model of the relational database management system (RDBMS) in some significant ways. These data stores may not require fixed table schemas, usually avoid join operations, and typically scale horizontally (Wikipedia).

OAuth – OAuth is an authentication protocol that allows users to approve an application to act on their behalf without sharing credentials like their username and password (OAuth Protocol).

PaaS – Stands for “Platform as a Service.” The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

Private cloud – The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises. (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

5 This capability does not necessarily preclude the use of compatible programming languages, libraries, services, and tools from other sources.
Public cloud – The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider. (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

RESTful – Stands for “Representative State Transfer.” It’s a style of software architecture for distributed hypermedia systems such as the World Wide Web. REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. HTTP, for example, has a very rich vocabulary in terms of verbs (or “methods”), URLs, Internet media types, request and response codes, etc. REST uses these existing features of the HTTP protocol, and thus allows existing layered proxy and gateway components to perform additional functions on the network, such as HTTP caching and security enforcement. Conforming to these REST restraints is referred to as “RESTful” (Wikipedia).

SaaS – Stands for “Software as a Service.” The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based e-mail), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

---

6 A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer.