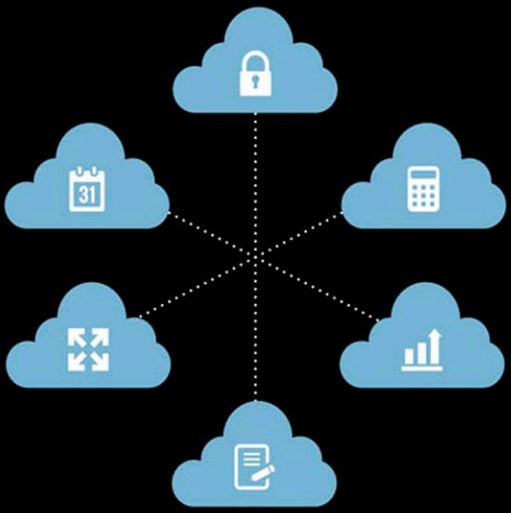
CLOUD COMPUTING

NAYAN B. RUPARELIA

REVISED AND UPDATED EDITION



THE MIT PRESS ESSENTIAL KNOWLEDGE SERIES

CLOUD COMPUTING

The MIT Press Essential Knowledge Series

A complete list of books in this series can be found online at
https://mitpress.mit.edu/books/series/mit-press-essential-knowledge-series.

CLOUD COMPUTING

REVISED AND UPDATED EDITION

NAYAN RUPARELIA

The MIT Press | Cambridge, Massachusetts | London, England

© 2023 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Chaparral Pro by New Best-set Typesetters Ltd.

Library of Congress Cataloging-in-Publication Data

Names: Ruparelia, Nayan, author.
Title: Cloud computing / Nayan B. Ruparelia.
Description: Revised and updated edition. | Cambridge, Massachusetts : The MIT Press, [2023]. | Series: The mit press essential knowledge series | Includes bibliographical references and index. | Summary: "Why cloud computing represents a paradigm shift for business, and how business users can best take advantage of cloud services"— Provided by publisher.
Identifiers: LCCN 2022045880 (print) | LCCN 2022045881 (ebook) | ISBN 9780262546478 (paperback) | ISBN 9780262376228 (epub) | ISBN 9780262376211 (pdf)
Subjects: LCSH: Cloud computing.
Classification: LCC QA76.585 .R87 2023 (print) | LCC QA76.585 (ebook) | DDC 004.67/82—dc23/eng/20220927
LC record available at https://lccn.loc.gov/2022045880
LC ebook record available at https://lccn.loc.gov/2022045881

10 9 8 7 6 5 4 3 2 1

CONTENTS

SERIES FOREWORD

The MIT Press Essential Knowledge series offers accessible, concise, beautifully produced pocket-size books on topics of current interest. Written by leading thinkers, the books in this series deliver expert overviews of subjects that range from the cultural and the historical to the scientific and the technical.

In today's era of instant information gratification, we have ready access to opinions, rationalizations, and superficial descriptions. Much harder to come by is the foundational knowledge that informs a principled understanding of the world. Essential Knowledge books fill that need. Synthesizing specialized subject matter for nonspecialists and engaging critical topics through fundamentals, each of these compact volumes offers readers a point of access to complex ideas.

PREFACE

This is a concepts book. It does not tell you which buttons to click on a cloud provider's portal in order to configure your cloud resources or how to use certain features that a particular cloud provider offers. Instead, it will equip you with the necessary understanding so that you will be able to judge what cloud resources you ought to be configuring and the rationale for doing so. Such an understanding is vital because, without that core foundation, you will not be able to understand the capabilities and usefulness of cloud computing.

Cloud computing has become a mainstay since the first edition of this book was written some seven years ago. With its greater adoption and mind share, cloud computing has made significant strides since then. As it is now a mainstream technology, it behooves you to know what cloud computing is, what its benefits are, what pitfalls you will need to overcome when transitioning to the cloud, the appropriate security measures you will need to adopt, and how you will use the cloud, whether it is a public cloud, such as Amazon's AWS (Amazon Web Services), or a private one that you specify, implement, and use. The purpose of this book, therefore, is to cut through the hype and show how to take advantage of cloud computing.

Until the last two chapters, I refrain from considering commercial cloud services or offerings in this book for three main reasons: (1) if one is not careful, the book could easily become an advertisement for various cloud service providers; (2) some vendors are here today and gone tomorrow, in keeping with the nature of the fast-paced technology industry; and (3) the principal aim of this book is to provide the concepts that will equip you to better make your own decisions about such offerings in the first place.

You could be an investor who wants to learn more about the cloud-based technologies employed by the businesses you invest in, an entrepreneur who wants to use cloud computing to ramp up your startup in an agile manner, a lawyer or judge working on a case that relates to cloud computing, a technologist who wants to use cloud computing in a new product or service that you are defining, a business student who wants to understand the paradigm shift that cloud computing represents to businesses globally, or a layperson who is curious about the subject—regardless, this book is for you. This book will help you understand cloud computing from a user's standpoint: when to use it and when not to, how to select a cloud service and integrate it with other cloud services or traditional IT, and best practices when using cloud computing.

This book is written primarily for a nonspecialist, although a technical specialist should benefit from reading it to understand the concepts and broader impact of cloud

computing. Thus no prior knowledge of cloud computing or any of its related technologies is required in reading this book. I advise strongly, however, that you read the first chapter of the book as a prerequisite so that we may have a shared, common vocabulary and understanding of cloud computing. This will prove useful when you read the succeeding chapters. You may read any of the chapters of the book in any order after the first chapter. A glossary is provided at the end of the book for some of the main terms and acronyms used in cloud computing and related technologies. I wish you all the best on your cloud computing journey. May it be an exciting one!

1

INTRODUCTION

Consider a typical workday that you spend using your computer. How much of your computer's resources do you actually use during peak usage times? The average for most users is about 10 percent of the processor, less than 60 percent of memory, and 20 percent of network bandwidth. (That is at peak usage; the normal usage levels during working hours are considerably less, on average.) Nevertheless, you paid for 100 percent of the resources up front when you bought your computer. Your networking costs are not any different in nature because most internet service providers (ISPs) have a lock-in period that commits you to their customer base for at least a year. Now suppose that your workplace has hundreds or even thousands of computers that are being used at a nominal rate: would it not be good to pool the unused computing resources of all

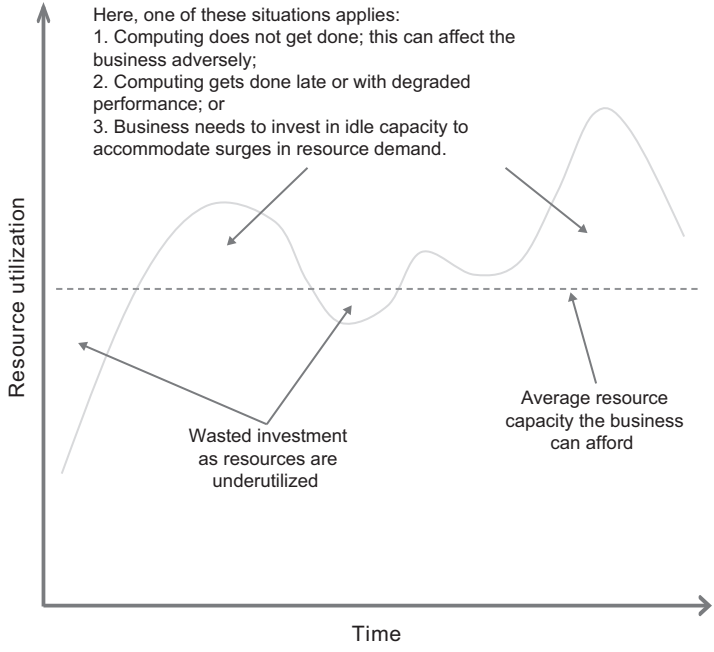


Figure 1 Investment problem solved by cloud computing.

the company's computers and put them to good use? That way your company would get the biggest bang for its buck.

Let us now transpose the same thinking to the data center, where numerous servers—web servers, application servers, database servers—are being used in a similar manner at minimal usage rates. These servers too could have their hardware resources pooled and shared across

servers so that you could utilize them more efficiently; otherwise, as in figure 1, you will need to invest up-front in computing resources that may be used only occasionally. The extra pooled resources that are unused by you could then be used by others in your company through your company's network. Alternatively, if your company appoints a third party to provide resource pooling, then you would access your computing resources over the internet.

But what if your company or department paid only for the computing resources that it uses? Then your company would not have to invest up-front, as a capital expenditure, in purchasing the computing but simply sign up with a service provider on a pay-as-you-use billing model. This effectively means that your company changes from a capital expenditure (CapEx) model to an operating expenditure (OpEx) model for meeting its computing needs. This is where cloud computing comes in.

A Definition of Cloud Computing

Although information technology (IT) has become ubiquitous at home and at work today, the industry is still in its infancy in comparison to, say, the automotive or telecommunications industries, which have existed for over a century. And cloud computing, one of the latest IT innovations, is still in its formative stage. It seems inevitable

that during the formative stage of a technology, much of the attention given to it borders on hype. Consequently, everyone from the technologist to the salesperson is keen to jump on the bandwagon by labeling anything with the remotest resemblance (often exaggerated or extrapolated) to cloud computing as being part of the cloud computing domain. This creates obfuscation and results in several definitions of cloud computing. The best definition is that provided by the National Institute of Science and Technology (NIST), a technology agency that is part of the US Department of Commerce and works with industry to develop and apply technology, measurements, and standards. The NIST definition of cloud computing is:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.[1]

Although the NIST definition needs updating—the service models, for example, are now five, not three—it is the most passable definition at present. In essence, this

book follows the NIST's definition to describe, one by one, the essential characteristics, deployment models, and service models of cloud computing.

Before delving into the characteristics of cloud computing, let us first delve into virtualization and cloud services because these form the basis of cloud computing in two distinct manners: virtualization from a technical perspective and cloud services from a conceptual perspective.

Virtualization

Cloud computing relies on virtualization technology. There are two basic types of virtualization: server virtualization and application virtualization. Application virtualization delivers an application that is hosted on a single machine (denoted as the server) and made available to many users. The application can be situated in the cloud on high-grade virtual machines (VMs); because several users access it, its costs are shared by those users. This makes the application cheaper to deliver to the end user. The end user does not need to have high-grade hardware to run the application; an inexpensive machine, such as a low-end workstation or a thin client machine, will suffice.[2] And if the data used by the virtual application are stored in the cloud, the user is not tethered to any one device or location to use the application or access related data. Typically in such cases, the

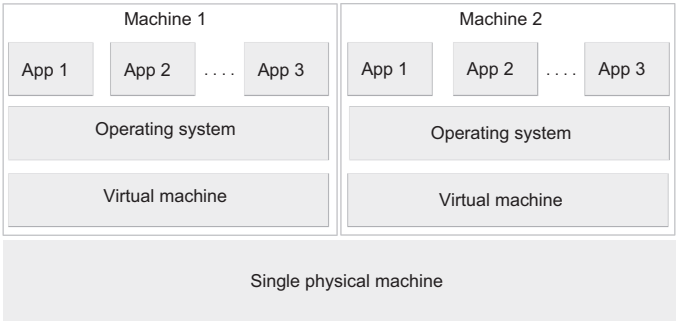


Figure 2 Virtual machines hosted on a physical machine.

virtual application is consumed through a mobile app or an internet browser by the end user.

Server virtualization uses common physical hardware (networks, storage, or computing machines) to host VMs, as figure 2 illustrates. A physical host machine could have any number of VMs running on it so that one set of hardware is used to run different machines.

VMs can be installed with their own operating system and their own different set of applications; the operating systems or applications do not need to be the same across the VMs. However, at the infrastructure level, it is possible to have virtual storage (multiple storage volumes on a single physical storage device) and virtual networks in addition to VMs.

Server virtualization has a major cost benefit: it allows you to consolidate a large number of physical machines

into fewer physical machines that host the VMs. This increase in computing efficiency results in lower space, maintenance, cooling, and electricity costs, besides the obvious reduction in the procurement costs of the machines. An additional benefit is that fewer physical machines and lower electricity consumption translate to environmental friendliness.

When the VMs are pooled together such that they may be instantiated—that is, activated and switched on—instantaneously in such a manner that they can join or leave the pool, you will be able to scale your resources to meet any change in demand, whether that change is an increase or a decrease. This instantaneous change in the number of VMs within a pool is known as elasticity and can be achieved in a cost-efficient manner owing to server virtualization.

Now, what is the difference between virtualization and cloud computing? Let us recall from the NIST definition of cloud computing these characteristics: on-demand self-service, rapid elasticity, and measured service provision. None of these features is provided as a matter of course by virtualization. Virtualization acts as an enabling technology to facilitate these features, but many additional enablers are required, such as reporting, billing, demand management, and various other business processes and tools.

To truly deploy a cloud, you need to consider how to employ virtualization and standardize your service

To truly deploy a cloud, you need to consider how to employ virtualization and standardize your service offerings.

offerings, make them available through simple portals, track usage and cost information, measure their availability, orchestrate them to meet demand, provide a security framework, provide instantaneous reporting, and have a billing or charging mechanism based on usage. Another way of looking at deployment is to understand that virtualization per se is not a service. It can be used, in conjunction with other tools and processes, to create an Infrastructure-as-a-Service offering.

Containerization

A container represents a partitioning of a server's resources such that it can run an application and thereby provide a service. The container is a run-time application that is built and run from a container image. This provides a cookie-cutter approach since you can build the same containers during run time from a single image. The images are stored in a registry that acts as a repository of all types of container images. You can have public image repositories, such as Docker Hub, or private ones that you create and make available only to developers in your own organization.

You could use a VM instead of a container to provide the same service. However, the container is lightweight compared to a VM and so you can deploy it from a cold start much faster than a virtual machine. This is because the container runs on the server within the server's

environment—albeit within a jail—whereas the VM has its own environment that is created from a partition of the physical server's memory, disk space, and central processing unit (CPU) resources, as figure 3 shows.

This means that a container can run on a VM as well as on a physical one. Indeed, since cloud computing itself is predicated on VMs, almost all containers that run in public clouds have historically run on a VM that itself is hosted on a physical machine. (For performance reasons, public cloud providers favor physical machines to host containers these days.) As figure 4 shows, VMs running on a physical server are used to host containers that in turn run applications; the single-purpose applications and containers, when bundled, are the microservices that cloud service providers enable you to create.

Chapter 4 delves deeper into containers and containerization.

Cloud Services

Let us examine what constitutes a service—specifically a cloud service—by employing the analogy of an accountancy firm. Suppose you want your accounts managed, and you appoint an accountancy firm. Let us postulate your main criteria for selecting the firm:

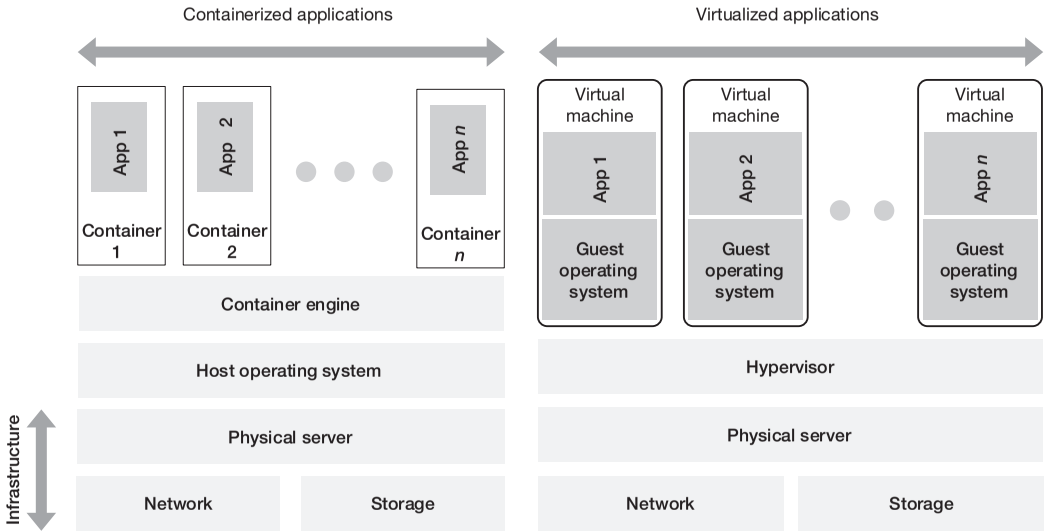


Figure 3 Difference between containers and virtual machines.

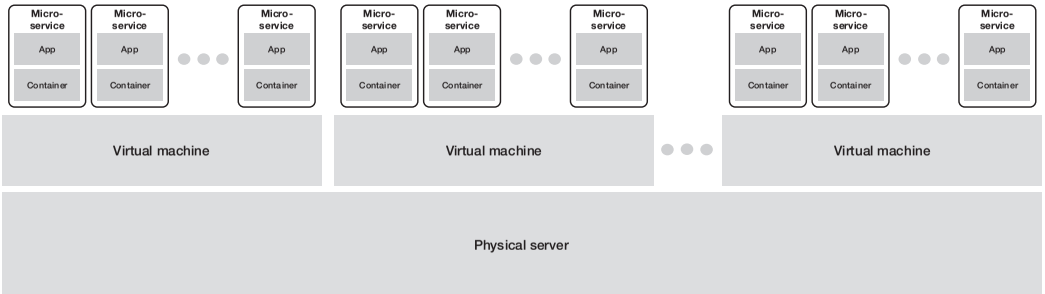


Figure 4 Containers hosted on virtual machines.

1. integrity and reputation of the firm (you want your accounts to be accurate, and you do not want your accounts released to the world!),

2. promptness in preparing the accounts and lowering your tax bill (the benefits that you will receive), and

3. fees the firm will charge (the cost of realizing the benefits).

You are not likely to care about the number of employees the firm employs in preparing your accounts, the software used, or the computers the software is installed on. Rather, your interest is in the firm's service and its benefits. These service benefits form a contract, a bond between you and the accountancy firm in a written or unwritten format. Such a contract is called a service-level agreement (SLA).

In IT, a service is a collection of IT systems, components, and resources that work together to provide value to users. An important element of this is that, for parties to the contract to measure and agree on the value received, two parameters are usually used to assess a service: cost and the SLA. The SLA is essentially a contract between the service consumer and the service supplier in terms of how quickly the service will be delivered (when), its quality (what), and scope (where and how much). Notice that these parameters represent the benefit that accrues to the service consumer. If the service consumer happens to be

internal to the company, such as a marketing department, then the internal agreement between the supplier and the consumer departments of that company is called an operational level agreement (OLA). A cloud service therefore is the implementation of a business process—provided through a set of related functional components and resources—that provides business value to its consumers.

Continuing this analogy, let's suppose that the accountancy firm wishes to ensure that it meets the SLA conditions agreed on with you. It could put in place various metrics internally that it could use to monitor its performance while creating the accounts. For example, the metrics could be that the audit needs to take three days, or the cash book needs to be reconciled within a week. The firm may agree to share these metrics with you, although usually they are used internally as objectives to ensure that the firm meets the overall SLA. These objectives, or metrics, are referred to as service-level objectives (SLOs). From an IT standpoint, SLOs are specific, measurable characteristics of the SLA such as uptime, throughput, available resource capacity, response time, and delivery time.

Service Models: Levels of Abstraction

Now let us look at IT from the accountancy firm's perspective. The IT department at the firm has that firm's

A cloud service therefore is the implementation of a business process—provided through a set of related functional components and resources—that provides business value to its consumers.

accountants as its customers. These accountants have a choice in the way they interact with their IT department:

1. They could get down to the nitty gritty and specify the hardware and software in terms of the type and version of software to use, the operating system that hosts the software, the hardware's memory, storage space, and so on and so forth.

2. They could specify the software they wish to use and let the IT department figure out what hardware to use.

3. They could simply agree on the type of input data they would like to have computed and the format of the resulting data set and leave it to the IT department to use whatever software and hardware it wanted to employ to compute the data.

The first choice represents a level of abstraction at the infrastructure layer. In cloud computing, it is known as Infrastructure-as-a-Service (IaaS). Common examples of IaaS are when you store data, files, or pictures in the cloud (this uses the storage infrastructure) or use the cloud to transfer files. A higher level of abstraction is when the IT department provides a platform, complete with hardware and operating system, and the accountants specify the software to use, as in the second choice above. This is Platform-as-a-Service (PaaS).

If the IT department were required to decide the right software and computing platform to use on behalf of the accountants, as in the third choice, such that the accountants only need to care about the accuracy and timeliness of the data returned to them, that level of abstraction would be Software-as-a-Service (SaaS). These three levels of abstraction—IaaS, PaaS, and SaaS—are the service models referred to in the NIST definition.

What if our accountancy firm decided to outsource the entire auditing function to another firm so that it could concentrate on advising you on tax matters? Our accountancy firm would agree to an SLA and the cost with the other firm, which would then audit the accounts accordingly. This amounts to outsourcing an entire business function, or process, to another firm.

That other firm could just as well be replaced by a cloud service. Thus the cloud service, when providing a business function, is providing a Business Process-as-a-Service (BPaaS).

Suppose that our accountancy firm wished to obtain the latest tax regulations. It could then commission an information service. This would be akin to Information-as-a-Service (INaaS). Since the tax codes and regulations are updated regularly as the law changes, this is a service that relies not only on the storage of data, which would then be IaaS, but also on the manipulation of that data to provide meaningful information. Hence INaaS is distinct from IaaS.

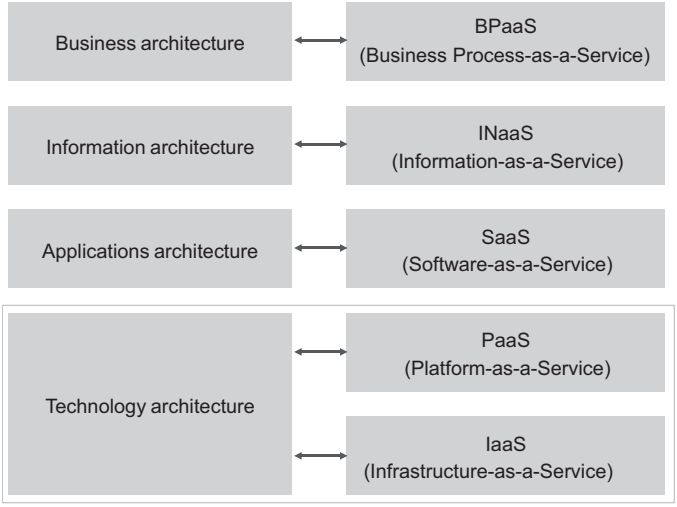


Figure 5 Enterprise architecture stack and cloud service models.

The NIST model therefore needs to be updated with these two service abstractions, INaaS and BPaaS, as figure 5 illustrates. IT enterprise architecture recognizes four architecture domains, as shown in the left column of figure 5. These are technology architecture, applications architecture, information or data architecture, and business architecture. Technology architecture encompasses IT infrastructure, middleware, and operating systems; applications architecture concerns software applications, their interactions, and their relationships with business processes; data architecture defines the data assets and

their management; business architecture translates the business strategy to an IT strategy, relevant governance framework, and definition of business processes. Each of these domains in the enterprise architecture stack maps to and aligns with the cloud service models shown in the right column of figure 5.

Cloud Deployment Models

The NIST definition contains four distinct deployment models: public, private, community, and hybrid clouds.

A public cloud provides services to anyone with internet access. Such a service may be provided by computing resources located anywhere in the world. This type of cloud has the disadvantage of data integrity for some companies for regulatory reasons. For example, companies based in the United States are not allowed to store consumer data in other countries. Financial institutions especially need to comply with strict regulations of this nature. As a result, some companies tend to favor private clouds. Another reason for favoring a private cloud over a public one is cost; the cost of having your own private cloud becomes lower once your annual cloud computing spend approaches $1 million.

A private cloud is one that provides services to a single entity, either a government organization or a business enterprise, such that cloud services are provided to that entity

from its own private network. Usually only large enterprises can afford to have private clouds. Small or medium-size enterprises have the option of using a community cloud.

A community cloud provides a middle ground between a private cloud and a public one. Various entities, ranging from individuals to enterprises, that have a common interest can pool their resources to create a hybrid cloud. Such clouds take various shapes: a banking cloud in Switzerland serving the cantonal banks (Switzerland is divided into administrative units called cantons), a paper industry cloud in the Nordic countries, or a health cloud for the health industry in the United States. However, there could also be community clouds for various interest groups, for instance for chess players or numismatists.

A hybrid cloud is essentially a conglomeration of the other types of clouds. Its use is necessary mostly when a cloud service needs to use computing resources from other clouds because its own resources are being utilized at full capacity. Such a concept is known as cloud bursting because the service bursts out of its cloud to utilize resources from other clouds to meet its SLA.

Five Characteristics of Cloud Computing

The NIST definition lists five characteristics of cloud computing, given in the first column of table 1: ubiquitous

Table 1 Characteristics of cloud computing

Characteristic	Description	Parameter
Broad network access	Consume services from anywhere.	Where
On-demand self-service	Consume services when you want.	When
Resource pooling and virtualization	Pool the infrastructure, virtual platforms, and applications.	How
Rapid elasticity	Share pooled resources to enable horizontal scalability.	How
Measured service	Pay for the service you consume as you consume it.	How much

access, on-demand availability based on the consumer's self-service, pooling of resources, rapid elasticity, and measured service usage.

Ubiquitous access through a network is important because you are not constrained by your location in using the cloud service; a concomitant concern is that, as the potential user base increases, so does the risk of security being compromised. For this reason, network access may be limited to a private network or a community of users. The former translates to a private cloud and the latter to a community cloud model.

On-demand self-service, or on-demand availability, makes it possible for you to use the service whenever you want. Availability has two characteristics here:

1. The service will be available to you even when you are not using it so that it will be ready for you to use when you request it (thus the uptime of the cloud computing services must approach 100 percent).

2. The service remains available while you are using it (thus your user experience should not be impaired even if there is an upsurge in the number of users).

The latter aspect means that, regardless of the varying demands you place on the service, it should remain available to you. For example, you could have an IaaS cloud service on which you host your website, and its usage levels will vary according to the hits the site receives. The usage will in turn depend on several factors, such as the time of day, whether it is a weekend, or if you have a marketing campaign running. As a result, the load placed on the cloud computing infrastructure will also vary, and the infrastructure will need to scale out when greater demand is placed on it. Similarly, it would scale in when there was less demand so that the infrastructure resources could be used elsewhere. This capacity for scaling in and out is known as elasticity.

Horizontal scalability occurs when greater numbers of the same type of resources—say, computing platforms—are used to meet demand. Vertical scalability occurs when the performance of those resources is improved by upgrading them, for example by increasing the amount of memory. Elasticity occurs when horizontal scalability is used to

scale out when demand is high and then scale in when it is low. To implement this, computing resources are pooled. The resources normally tend to be virtualized because you can use software to pool and scale them automatically.

Just as virtualizing the hardware allows you to pool and share the resources in an elastic manner, you can have virtual applications that can be shared even though a single instance of the software runs on the pooled VMs. This technology, however, is still in its formative stage as the biggest constraints are commercial issues such as the licensing arrangements and billing model. When multiple users use the same virtual resources in the cloud, such as the software, storage, or VMs, those resources have multiple tenants. The pooling of resources to provide a shared, common service to each user of the cloud service is known as multitenancy.

A major disadvantage of current public cloud services is a lack of transparency in terms of the resources consumed and the costs incurred.[3] Yet these are distinct characteristics of cloud computing insofar as the consumer ought to know what computing resources are being consumed as and when they are consumed, and the instantaneous concomitant costs of the consumption. (Of course, this factor becomes less relevant if the charging model is based on a "consume as much as you want" pay monthly basis.) That is why it is important for a cloud service to measure the consumption of the service and make that metric transparent to the user in real time.

2

A HISTORICAL PERSPECTIVE

Now that we have a foundational idea of what cloud computing is, let us consider its history. In accordance with the adage "standing on the shoulders of giants," cloud computing uses numerous technologies and paradigms that were developed earlier. It uses the client-server paradigm, whereby a local client uses the computation performed by a server remotely, but it is directly predicated on internetworking (to enable ubiquitous access) and virtualization (to enable scalability via elasticity). So let us assess cloud computing's history by first delving into the history of its two key components.

Internetworking

Before the internet existed, engineers were busy contriving various ways to connect computers in local area networks.

As a result, several protocols surfaced in the 1970s and 1980s leading to what was known as the protocol wars. The differences between the various networking solutions ranged from physical (what connectors to use, wired or wireless, etc.) and logical connectivity (the topology to use: bus, star, tree, ring, mesh networks) to protocols (packet-, message-, or data-based protocols, connected or connectionless protocols, bandwidth to use, etc.). As the number of solutions and implementations proliferated, it became apparent that these disparate networks needed another network, an internetwork, to connect them. Thus the internetwork (commonly known today as the internet) was created around 1989 by a partnership of computer and telecommunications companies that incorporated the TCP/IP protocol into various operating systems. Earlier, in May 1974, the Institute of Electrical and Electronics Engineers (IEEE) had published a paper titled "A Protocol for Packet Network Intercommunication" that described an internetworking protocol for sharing resources using packet switching among network nodes. This protocol had been developed by the Defense Advanced Research Projects Agency (DARPA) of the US Department of Defense (DOD) in collaboration with various universities as part of its ARPAnet. Version 4 of the internetworking protocol, denoted IPv4, was released in 1981 by the DOD, which made it a standard for all its military computer networking in March 1982.

To ensure that the whole gamut of networking operations was addressed—including how data should be packetized, addressed, transmitted, routed, and received—so that no other variance could be used to whittle the specification, the internet protocol was designed to specify end-to-end data communication. The internet protocol suite (often referred to as TCP/IP) is organized into four abstraction layers: the link layer (which provides the communication methods for data within a single network segment, or link), the internet layer (which specifies internetworking between independent networks), the transport layer (which handles host-to-host—i.e., computer-to-computer—communication), and the application layer (which handles how the data are processed by applications). The last two layers are referred to as TCP and the first two layers as IP. The IP layers have the task of delivering packets from the source host to the destination host using only the IP addresses in the packet headers. To accomplish this, the IP layers contain packet structures that encapsulate the data to be delivered, as well as the labels for the datagram containing the data's source and destination information. The IP protocol is complemented by the TCP protocol, which is a connection-oriented service.

Among the first corporations to adopt TCP/IP were IBM, DEC (formerly Digital Equipment Corporation), and AT&T. Beginning in 1984, IBM started releasing TCP/IP in various systems such as MVS, VM, and OS/2. At the same

time, TCP/IP stacks began to be offered for MS-DOS and PC-DOS by several smaller companies. Thus the internet as we know it today became commercially available in the mid-1980s.

Virtualization

In the 1960s, during the heyday of mainframe computers, IBM embarked on solving a problem: how to make computing cheaper on a per user basis. As mainframes were very expensive, having efficient per user costs based on time sharing of computing resources seemed like a viable option. (Time sharing allows multiple users to use a single computer at the same time.) This led IBM to pioneer virtual machines in 1964 at the IBM Cambridge Scientific Center in Cambridge, Massachusetts, that created the control program (CP), which was a hypervisor that ran multiple virtual machines (VMs), with each VM having its own separate hardware stack and a lightweight version of the operating system. This culminated in IBM releasing the first commercially available VM on August 2, 1972. It ran on the S/370 operating system on a mainframe computer and was called VM/370.

As with the internet, DARPA played a pivotal role in virtualization. It funded the FLASH research project at Stanford University on virtualization as described in "Disco: Running Commodity Operating Systems on

Scalable Multiprocessors," a paper published in the *ACM Transactions on Computer Systems* in 1997; the authors of the paper were Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. They described a prototype hypervisor—software that sits between the hardware and the operating system—which they named Disco. Disco used the TCP/IP protocol to couple operating systems running on different machines, and so allowed them to scale horizontally. In 1998, three of the paper's authors co-founded VMWare, a company that provided the very first commercially available VMs that ran on Intel microprocessors, referred to as x86 machines. These were stand-alone VMs running on workstations. In 2002, VMWare released the first commercially available hypervisor, the ESX Server 1.5, which allowed users to consolidate physical devices by creating a greater number of VMs on them.

The next major milestone was to have a free, open-source hypervisor that made virtualization cost-effective. In 1998, the same year that VMWare came on the scene, a demonstration of Simics was presented at the USENIX 98 conference. This was followed by Xen, an open-source project allowing multiple operating systems to be run on the same hardware concurrently. It was created as a research project at Cambridge University and released on October 2, 2003. Xen is currently part of the Linux Foundation and has Amazon AWS (Amazon Web Service) and Citrix as its project members, among others.

To end our discussion of the historical aspects of virtualization, let us not forget containers. They came a little later when, with the release of FreeBSD 4.0 in 2000, jails were introduced. A jail is a partition of various system resources that share the same operating system kernel and so imposes limited overhead on the hardware. This paved the way for containers, which were introduced in Linux on August 6, 2008.

Cloud Computing

Cloud computing's history is checkered. It has evolved gradually, based on the formative technologies that we have considered in this chapter. Multiple users, for instance, could use a single mainframe in the 1950s by using dumb terminals. The need to perform computing, which was an expensive resource then, on a time-shared basis meant that virtualization technologies were pioneered in the 1960s. By 1999, the internet was well established when Salesforce developed its cloud-based SaaS customer relationship management software. Around this time, Amazon was becoming well known as a business disruptor using technology. It was approached by various retail companies, such as Target and Marks & Spencer, to build online shopping sites for them that would sit on top of Amazon's e-commerce engine. This was followed

Today, three major cloud providers have the lion's share of the public cloud computing market: Amazon, Microsoft, and Google.

by Amazon creating application programming interfaces (APIs) and tools to interface with its Amazon.com website catalog in 2004, and Amazon internally became a service company that used decoupled APIs to access services. However, in 2003, the concept of what AWS could be was formulated at an executive retreat. Indeed, it was one of the first businesses that considered the use of 10 percent of its computational capacity as a financial problem that needed to be solved.

On March 14, 2006, AWS was launched as a public cloud by making storage available as S3 (Simple Storage Service), which was followed by the launch of Amazon SQS (Simple Queue Service) and Amazon EC2 (Elastic Compute Cloud) in August 2006. Others, such as Oracle, IBM, Microsoft, and Google, followed with their own offerings in later years.

Today, three major cloud providers have the lion's share of the public cloud computing market: Amazon, Microsoft, and Google. Although Amazon's AWS has around 90 percent of the market share, Microsoft has the potential to press ahead (provided it revises its pricing model and creates lightweight Windows servers) as it has a few sticky technologies, such as Microsoft Office, Microsoft Active Directory (for security), and Exchange (email server), that create a compulsive ecosystem for businesses.

3

TYPES OF CLOUD COMPUTING

The first chapter touched on two key characteristics of cloud services: service abstraction levels (IaaS, PaaS, SaaS, INaaS, and BPaaS) and deployment models (public, private, hybrid, and community clouds). This chapter delves deeper into both by comparing their component characteristics, culminating in some key paradigms of cloud computing based on cloud relationships.

Let's recall the definition of cloud computing from the first chapter. Cloud computing has five key properties:

1. broad network access,
2. on-demand self-service,
3. resource pooling or shared services,
4. rapid elasticity, and
5. measured service.

Every single one of these properties needs to be present for a service to qualify as a cloud service. It is very important to understand this because otherwise the service is just not a cloud service, despite the claims made by would-be cloud service providers. Consequently, all the deployment models and abstraction levels that we consider in this chapter have in common these five properties of cloud computing.

There are a few novel concepts that I have developed while designing and creating cloud services that we shall consider. One such concept is based on object-oriented design (OOD) principles in terms of object relationships. Another concept is that of a cloud cell; this can be thought of as a service component performing a single task, such as storing data, implementing a single business function, providing a database service, or serving web pages.[1] (Regardless of whether the cloud service is hosted on a virtual machine or a container, we shall use the generic term "cloud cell" instead of "microservice," which is generally applied to containers.) You can create a cloud application that uses a number of cloud cells to create a cloud service. This concept means that you can reuse a number of cells and, through various combinations, create a variety of cloud services. We can extend this approach further by employing cloud patterns. These are distinct use cases based on a combination of cloud cells and their relationships. The chapter concludes with a discussion of cloud patterns, or cloud service patterns.

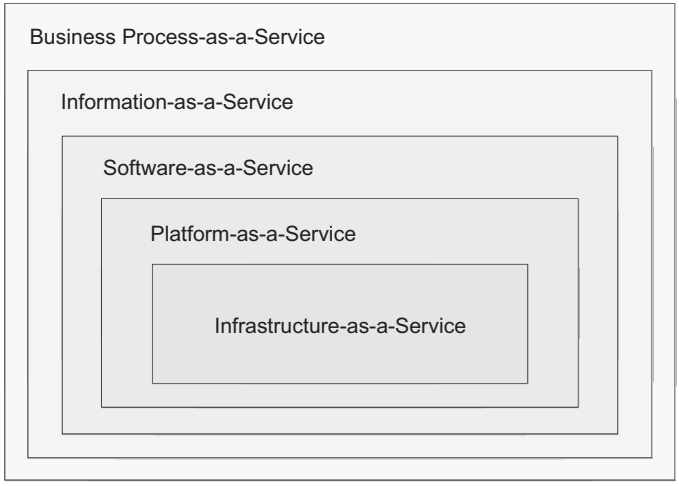


Figure 6 Levels of abstraction.

Abstraction Levels

Service models can be viewed in terms of increased levels of abstraction. As you move up the stack to the top, you have the highest form of abstraction at the Business-Process-as-a-Service (BPaaS) level, as illustrated in figure 6.

Notice that each higher level of abstraction incorporates the levels below it. Software-as-a-Service (SaaS) incorporates software that provides a business function and at the same time provides a Platform-as-a-Service (PaaS). And Platform-as-a-Service provides an operating

environment in addition to Infrastructure-as-a-Service (IaaS). Table 2 describes the five abstraction levels of cloud computing in terms of the service offerings.

Notice in the table the distinctions between the abstraction levels. It is easy to mistake one service for another, especially when marketing departments of different cloud service providers stretch the definitions. For example, suppose you are provided with IaaS. The service provider will pre-install the operating system for you because it does not wish you to have access to the infrastructure for security reasons. The service offered is still IaaS, not PaaS, even though it has an operating system installed. For it to be PaaS, the service provider would additionally need to install software that your application would need for it to run. This additional software might be software libraries that perform set tasks, a framework such as .Net that includes a standard set of libraries, or even an application stack such as LAMP (Linux-Apache-MySQL-PHP or Perl) that includes a web server (Apache), a database (MySQL), and a programming language with its libraries (PHP or Perl) hosted on a Linux server. Once you use the IaaS or the PaaS service to write your application, you can host it in the cloud and offer it as SaaS—Software-as-a-Service. That way your customers do not have to worry about installing the application, the operating system it runs on, or any of your application's software dependencies. More crucially, they do not have to worry about keeping the

Table 2 Service offerings for different abstraction levels

Abstraction Level	Service Offering
Infrastructure-as-a-Service (IaaS)	Provides hardware infrastructure such as servers, storage, and the like on a utility basis.
Platform-as-a-Service (PaaS)	As for IaaS, but also includes the operating system and any other core applications that make up the operating environment to enable users to install and run software. Pricing is generally on a utility basis.
Software-as-a-Service (SaaS)	As for PaaS, but also includes hosted applications that fulfill a function. The function could be a business, social, or personal function. You simply use the application or applications that you need, when you need them, and avoid the cost of installing and maintaining the application and its supporting hardware infrastructure. Pricing is on a per use basis.
Information-as-a-Service (INaaS)	Provides information that an individual or corporation needs and that is relevant to their business, business process, or a task. Pricing is usually on a consumption, per use basis.
Business Process-as-a-Service (BPaaS)	Fulfills a business function or replaces a business process in an organization. Typically combines business process outsourcing (BPO) with Software-as-a-Service (SaaS). Pricing is generally on a per use basis.

software current because you will install the latest version with your SaaS offering.

Deployment Models

Recall from chapter 1 the four deployment models of cloud computing: public, private, community, and hybrid. Each deployment model additionally has an abstraction level that describes it. For example, a public cloud having an abstraction level of SaaS would be described as an *SaaS public cloud*. Likewise, a *PaaS private cloud* would be a cloud that has a private deployment model and a PaaS abstraction level, and so on.

Public Clouds

A public cloud, as its name suggests, is available to the public at large. In this regard, the public can be a consumer or an organization that wants to use cloud services. Public cloud services are almost always consumed via the internet rather than via a private or restricted network. The public cloud is the deployment model that most people are familiar with. The public cloud has applicability across the various abstraction levels. As such, you will find public clouds that provide infrastructure, platform, software, information, or business processes as a service. The public cloud model is almost always available only with a monthly

charge for use that is based on subscription or utility (pay per usage) price models. Examples of a public cloud service include Google Cloud Platform, Google Docs, Microsoft Office 365, Amazon's AWS (Amazon Web Service), DocuSign, Dropbox, and Microsoft's Azure, among others.

Private Clouds

A private cloud has as its scope an organization, business unit, or even one person. A multinational corporation may have its own private cloud that delivers services over its wide area network (WAN). A WAN can be thought of as a company-wide internet that restricts outsiders through the use of security devices such as firewalls. A local area network (LAN) is similar to a WAN except that it is much smaller in scope: it normally is restricted to a particular site, such as a home or a business location. A person could have her own private cloud to consume services over a LAN. For example, a home might have its own cloud that (1) connects a streaming server to a video set-top box so that videos can be watched, recorded, or played back from anywhere in the house; (2) provides a backup server to store files centrally; or (3) comes with a synchronization service that synchronizes data across devices (laptops, mobile phones, tablets, etc.) using a wireless LAN; this would be akin to having your own personal Dropbox.

A private cloud is therefore one that delivers services over a LAN or a WAN and restricts the consumption of

those services to a select group of users. In limited circumstances, private cloud services may be delivered via the internet, but with restrictions so that only private entities can gain access to those services. Generally, private clouds require some form of capital expenditure to set up, and they may also have an operating expenditure component.

Community Clouds

The community cloud is a broader version of a private cloud. It supports a community that has common interests or shared concerns such as security requirements, a common regulatory environment, business models, or hobbies. A community cloud may even have a geographic region as its scope, for example an EU community cloud or a North American cloud. It can have trade as its scope, for example an ASEAN or a BRIC community cloud. The trade concept is quite interesting because it can be extended to any number of industries or business groups: a paper industry cloud, a publishing cloud, a banking regulation cloud, a health industry cloud that may be specific to a country (e.g., a US health community cloud) or to the industry vertical globally (e.g., a worldwide health community cloud), and then have as its participants regulatory bodies, health providers, practitioners, and consumers, or any combination of these.

The community cloud shares the same cloud infrastructure if it is an IaaS community cloud, the same cloud

software if it is an SaaS community cloud, or the same cloud business processes if it is a BPaaS community cloud. Unlike the services available through a private cloud, community cloud services are usually delivered over the internet and have an operating expenditure price model.

Hybrid Clouds

A hybrid cloud is an encapsulation (see the section on encapsulation below) of two or more cloud deployment models (private, community, or public) that has its own unique characteristics. The hybrid cloud can be made up of a single deployment model, as in *Hybrid cloud 1* of figure 7, which is a hybrid cloud comprised of three private clouds; it can equally well be composed of public clouds instead. The deployment model does not need to be the same. You can have a hybrid cloud consisting of different models, as is the case for *Hybrid cloud 2*, which has a private and a public cloud. Then again, hybrid clouds can have community clouds as in the case *Hybrid cloud 3*. There can even be a hybrid cloud within a hybrid cloud, as in *Hybrid cloud 4*, so that you could have a replica of the other component clouds within your inner hybrid cloud for purposes of business continuity or load balancing.

Can the component clouds of a hybrid cloud be of any abstraction level? For instance, can you have a private cloud that is IaaS and a public cloud that is PaaS within a hybrid cloud? Absolutely! Let us consider an example by

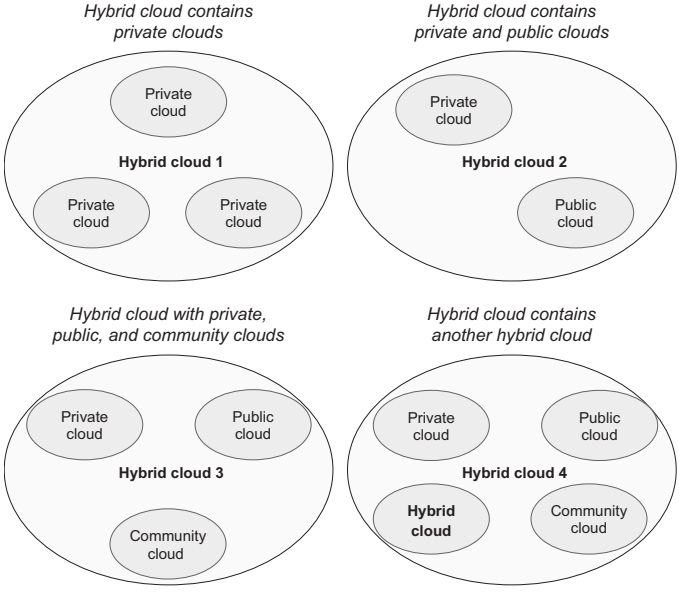


Figure 7 Varieties of hybrid clouds.

revisiting our accountancy firm. Suppose that our firm wants to make use of public cloud services for customer relationship management, mail, and word processing using commonly available commercial public cloud services but at the same time has some business applications such as inventory-tracking software that are available in its own private cloud. Our firm could create a hybrid cloud such that some of the data could be shared between the public and private clouds, for example supplier addresses. This

would allow the public cloud's word-processing service to send letters to suppliers and at the same time allow the private cloud's inventory tracker to assess which supplier ought to be contacted in order to replenish stock. What are the advantages of such a hybrid cloud? The private cloud provides data privacy, greater performance, and transparent service-level objectives; the public cloud provides standard services that grant it flexibility and cost efficiency, as our accountancy firm does not have to reinvent the wheel by having to create its own word-processing services.

Types of Cloud

Clouds can take the form of any deployment model. You can deploy your personal cloud as a private cloud or as a public one, for example. Dropbox is an example of a public cloud deployment model that can be used as a personal cloud for files. In this section we consider the personal cloud and the cloud of things as two distinct types of cloud.

Personal Cloud

A personal cloud is defined by its scope rather than by whether it is available on a shared basis. Recall that this cloud's scope is a person or a single entity and that it can be a private, public, or hybrid cloud. Examples of public personal clouds are iCloud, Google Drive, and Dropbox.

Dropbox is an example of a public cloud deployment model that can be used as a personal cloud for files.

An example of a private personal cloud is a network-attached storage device that backs up your data; Apple's Airport Time Capsule, when connected to more than one device, is an implementation of this concept.

Cloud of Things

A cloud of things has inanimate objects, or things, as its scope; that is, it is a cloud that works with things instead of people or organizations. For example, you can have a cloud of public lighting sources—such as street or car park lighting—operated on a pay per use basis so that the lighting is available only to those who pay for it. The lights therefore turn on when someone is in the vicinity and turn off when they leave; the amount charged would depend on the length of time they are in the vicinity, and hence the payment would be based on their use of the lighting. This may mean that the lights do not remain turned on all night and waste money. It may also mean that only those residents who use the lights pay for them rather than everyone in the community through taxes or any such collective charging schemes. The pay per use charging mechanism could be provided through the agency of chip-and-pin or near-field communication technology that is available with most credit and debit cards. The lights connect to their cloud via general packet radio service (GPRS) and transmit information pertaining to such things as usage and bulb replacement needs.[2] Other applications of a

cloud for things would be for cars, houses, health monitoring equipment, household appliances, and offices.

Cloud Cells

A cloud cell is a cloud service that provides a distinct, fundamental function or service; it acts as a unit so that its service can be reused by other cloud services or cloud cells. You can think of a cloud service as encapsulating cloud cells. Microservices are an example of a cloud cell in its native form. Microservices are short-lived services that use containers to provide a single service. However, you could also have a cloud cell that is based on a long-lived service using a virtual machine instead of a container.

Examples of cloud cells are a database cell that provides the services of a database server, a web server cell that hosts internet sites, or an email server cell. In other words, the same type of cloud service does not need to be created for every cloud that needs it because you share the cell. You invest once in creating a cloud service—as a distinct cloud cell—and then leverage it so that other cloud services can use it. (Of course, there are a lot of technicalities involved—how the cloud cell exposes its services to other cells, how it advertises its services to those cells, whether a service catalog is maintained by a controlling cell, and what common data models and formats ought

to be used to pass information between the cells—but we won't delve into these details since our purpose is to understand the main concepts.)

This reusability provides cost efficiencies. Thus, having specific cloud cells that perform distinct functions and are reused by other cloud services or cells can increase productivity and thereby profits. This is the opposite of Gossen's first law of economics, referred to as the law of diminishing marginal utility, according to which the more of something you use, the less each additional unit of it is worth to you than the one before it; the cost efficiencies of reusability can be thought of as providing an increasing marginal utility with usage. Likewise, reusing cloud cells increases agility in calling up a particular cloud function because you will not have to create it from scratch. The increased agility translates to a shorter time to market.

Figure 8 shows a cloud service—called a mother cloud cell—that consists of a number of cloud cells. The controller cell is optional, depending on the use. Its main purpose is to act as an orchestrator of services that the other cloud cells provide. Using a web service as an example, you could have the following cloud cells encapsulated: a web server cell, a database cell, a storage cell, and a SaaS cell that contains the business logic for the web service. A special form of cloud cell, used mainly within the IaaS and PaaS space, is the cloud gear. Cloud gears are specialist cloud cells that individually provide applications such as antivirus

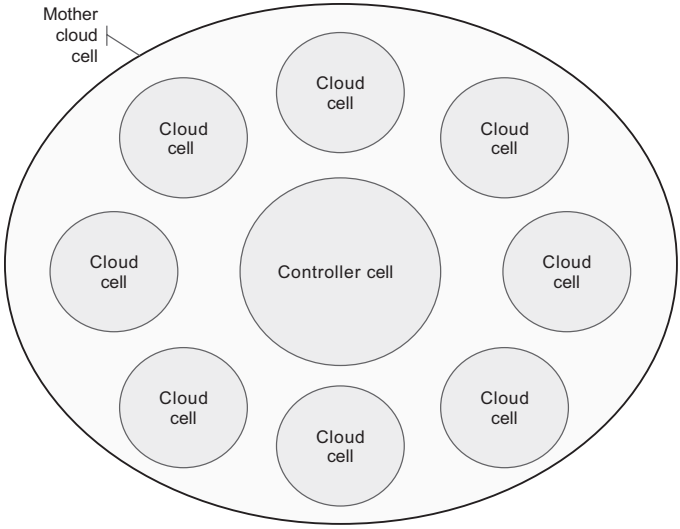


Figure 8 Cloud cells within a cloud service.

protection, hard disk encryption, public file sharing, and backup.

Cloud Cell Patterns

The web service cloud we just considered follows a pattern. Like all web services, it has a database, some storage, business logic, and a web server. Those components, in essence, describe a pattern for a web service. Similarly,

you can have numerous use cases: email service, inventory management service, order-processing service, and audio streaming service are just a few examples. Each of these can be described as a distinct cloud service comprising cloud cells that provide distinct functions. Thus cloud patterns are implemented by cells, and they provide a template for a given cloud service or its use case.

Patterns are not new. In programming, they come to us from the field of object-oriented software and were first popularized by the "Gang of Four"—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, authors of the 1994 book *Design Patterns*.[3] They described software patterns as objects that can be reused to create a solution to a problem. They posited that a pattern has four essential elements:

1. *Pattern name*

   An identity that creates a common vocabulary to invoke the pattern.

2. *Problem statement*

   A description as to when and under what circumstances to apply the pattern.

3. *Solution*

   A description of elements that make up the pattern (in our case, the names of the cloud cells), their

relationships (we discuss these in detail in the following section), and interfaces.

4. *Consequences*

   Trade-offs and the impact of using the pattern (in our case, the price and value model that would apply to the pattern).

Table 3 describes our web service example as a pattern using the Gang of Four's framework.

The format of table 3 can be used to describe any number of cloud patterns that fulfill a particular use case. When the pattern is implemented, the components of the pattern become distinct cloud cells.

Cloud Relationships

A common theme you will find with hybrid clouds is that the component clouds will need to share or integrate data to be able to port the cloud-based applications or their services from one component to another. Actually, this need to share information is not specific to hybrid clouds. A private or public cloud also has as its components other clouds or cloud cells that provide given tasks or services. And for this to happen, there needs to be defined relationships between the clouds or cloud cells, and microservices can be thought of as cloud cells in this regard.

Table 3 Cloud pattern for a web service

Name	Web Server Cloud Service
Problem	Host a customer-facing website that uses business logic to respond to users in real time
Solution	*Components*: 1. Web server cell: Contains Python, Apache HTTP server, and Node.js 2. Database cell: Contains PostgreSQL database 3. Storage cell: Provides storage for the other cells 4. Service bus cell: Provides connectivity with back-end systems 5. Firewall cell: A virtual firewall; implements a local demilitarized zone *Relationships*: Encapsulation—composition of above five cloud cells *Interfaces*: 1. Internal: Firewall Webserver (Database, storage and service bus cells) 2. External: Users (via the HTTP server), administrators, and back-end database (via the service bus)
Consequences	*Price model*: Consumption-based price model *Value models*: User demand flexibility and location flexibility

These relationships are examined as encapsulation, composition, and federation.

Encapsulation

Encapsulation occurs when an object contains or consists of another object. In encapsulation, one object is said to contain another. Encapsulation minimizes your work because you can apply a cookie-cutter approach with it. For

example, a forest of eucalyptus trees could be described as a forest that encapsulates eucalyptus trees. From an engineering viewpoint, you need only describe one tree and then describe the forest as an aggregate of that single type of tree.

Encapsulation is of two types: composition and aggregation. Our eucalyptus forest is an example of aggregation. Composition occurs when distinct objects contribute to create an overall object, and not just one repeated object type as in the eucalyptus tree forest example. Suppose you have to describe a car: it consists of four wheels, a steering mechanism, a hood, an engine, and so on. The car encapsulates these parts that, together, as a composite, make it a car. The car is a composition of its parts, whereas the eucalyptus forest is an aggregation of eucalyptus trees. From a cloud computing perspective, one cloud can encapsulate other clouds or cloud cells. Hence, for our purposes as related to cloud computing, we define encapsulation as being the composition or aggregation of cloud services.

Aggregation

Aggregation is applicable to any cloud abstraction type. Figure 9 shows an example of a SaaS cloud that is made up of other SaaS components that can be described in similar terms with regard to the functionality, operating expenditure charges, or the service-level agreements (SLAs). So, what use does aggregation have in practice? Let

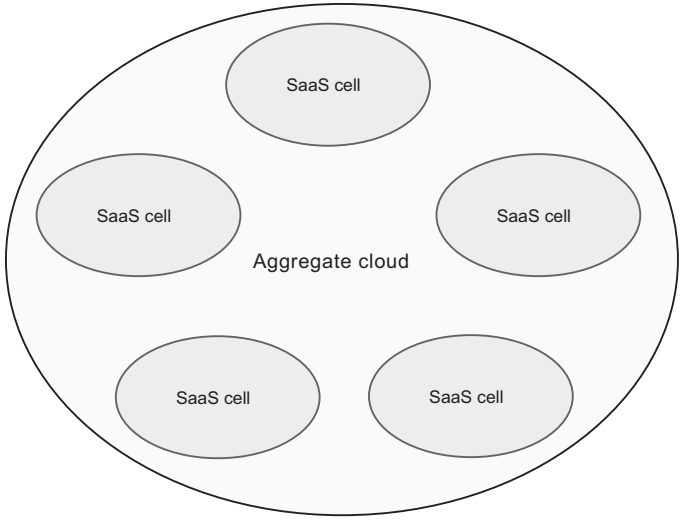


Figure 9 An aggregate SaaS cloud.

us revisit our accountancy firm analogy. Suppose our accountancy firm has an SLA with its IT department to provide accounts reconciliation through a SaaS cloud service. Suppose further that the accountancy firm has a surge in new clients—perhaps as a result of better marketing or a merger with another firm. The IT department now has increased demands placed on it that it cannot meet using the same SLAs. It could therefore create a replica of the reconciliation cloud so that it can cater to the increased demand. The reconciliation SaaS cloud will then be two cloud

services (or cloud cells), each cell a replica of the other. In this way, demand for the reconciliation service can be distributed across the two cells. The effort expended in creating a replica cell is very small compared to that needed to create a cell from first principles since an image of one cell can be used as a template for creating another. The obvious gains from this cookie-cutter approach, which is an example of the use of aggregation, are in agility and cost efficiency.

Let us consider another approach: suppose that instead of creating a replica of the SaaS cloud, the IT department identifies an external company that can provide the same type of reconciliation as its own cell, so it negotiates an SLA with that company for its cloud function to be the same as its own. This way the IT department will have a SaaS cloud that encapsulates its own reconciliation cell together with the other company's cloud cell to meet the increased demand. The IT department could go even further and treat the other company's cloud cell as if it were its own so that any extra demand that its own cloud cell could not meet would be sent to the other company's cell. This is known as "cloud bursting," whereby one cloud bursts to another so that it can be elastic in meeting demand needs. Yet another use case for aggregation would be to have a cloud cell located in a different data center than the main center. The replica cell in the different data center could

then be used for disaster recovery or business continuity. Both the primary and the secondary cells in the two data centers would be part of one SaaS cloud service that met business continuity SLAs. Last, IT could create a community cloud service that aggregated cells so that some cells that met its requirements could be reused or even shared with other community clouds, provided that the cells were common to both community clouds.

In summary, aggregation has many use cases. Examples include (1) to increase elasticity to meet user demand, (2) to enable cloud bursting across different clouds or service providers, (3) to provide business continuity, and (4) to enable reusability, or the sharing of cloud services that are common to different clouds. Incidentally, the example here of a community cloud is applicable to other types of clouds too.

Composition

When a cloud is composed of one or more distinct cloud services, composition is being used. In figure 10, a composite cloud is shown that encapsulates five cells. A key benefit of composition is reusability. For instance, a storage cell can be defined and used for one composite cloud and the same cell, or a replica created from its image, can be used in another composite cloud. This reusability is predicated on standardization, so a standard storage cell

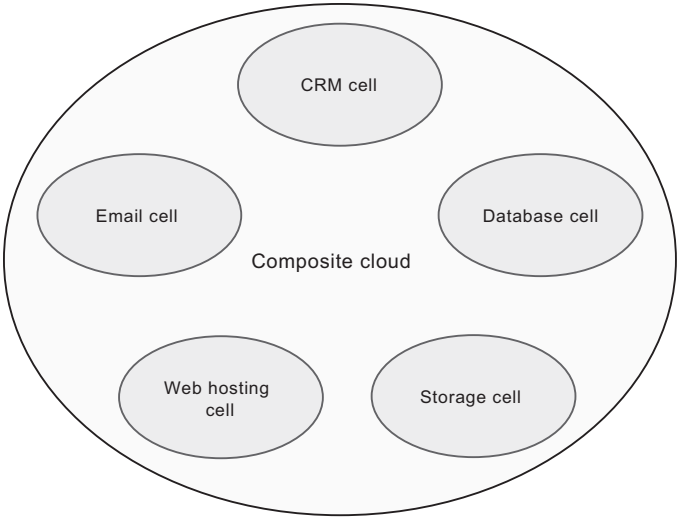


Figure 10 Cloud composition.

definition with its own set of requirements, interfaces, functions, and management characteristics needs to be created. By management characteristics we mean:

1. updates and patches to the cloud cell,
2. upgrades to the cell's hardware,
3. upgrades to the cell's operating system and core applications that form part of the platform,
4. upgrades to the applications hosted on the cell, and

5. changes in the roadmap that define when upgrades and updates are to occur in future.

Any change to one characteristic of the standard cell will change that characteristic on all its replica cells. This makes managing changes much easier, although it does require extra testing because any change applied to a cell may be fine in one composite cloud but not in another. So all the composite clouds that encapsulate the replica cells derived from a given standard cell will need to be tested to commit changes to the standard cell.

Standardization, reusability, and manageability are key benefits of composition. Other benefits include agility in instantiating and provisioning clouds. That is to say, as you define a new cloud service, you need only decide what building blocks—in the form of cloud cells—it should contain. And presto, you have a new cloud service ready in minutes! Moreover, with the higher level of abstraction provided by encapsulation, details such as how a cloud cell is built are not important. In creating the cloud cell, all you need to know is what a cloud cell does; you do not need to know the details of how it is built and what it contains. Tools such as Terraform, CloudFormation, Ansible, and Cloud Foundry allow you to automate the building and deployment of cloud services, and so using templates for cloud cells with those tools allows you to automate the process.

Federation

Federation is a special type of composition. Cloud services provided by disparate clouds—usually from different cloud service providers—can be federated to create a composite cloud service. You could think of it as a "mash-up" of cloud services. The component cells of the federated cloud can be a mixture of your own cloud cells and a third-party's cloud cells. You can even have a federated cloud that is composed entirely of cells from third-party cloud service providers. This means that other than negotiating your SLAs and the prices, you do not have to go through the trouble of creating your own cloud or its cells. This approach assumes, of course, that the right cloud services are available from service providers. Apart from the benefit of rapidly creating and using your cloud, federation has another benefit, from a service provider's or broker's perspective. A cloud service broker can assemble a cloud that federates cloud services from a number of cloud service providers; can negotiate contracts, SLAs, and prices with them; and can dispatch the federated cloud service as an assemblage to you. As a cloud service consumer, you then would not have to negotiate terms and prices with all the other providers because you would simply deal with the cloud service broker. Another benefit of federation is that you can ramp up your services, either by adding extra capacity to existing services to meet increased demand or by augmenting your service offering with extra services.

You can do this because you have instant access to a global marketplace of cloud services. As a result, if you have a customer that suddenly needs resources or services that you do not have available as part of your current cloud service, you can simply purchase the service from the marketplace and add that service to your cloud service catalog as a cloud service broker or service provider. Yet another benefit is that you can use federation to ensure that one cell within your cloud will fork out work to another cell to balance the workload or to ensure failure safety. (Failure safety provides a guarantee that when one cell takes over work from another, the transaction will not fail, thus ensuring that a resilient service is provided.) Please note that federation is based on a major assumption: one cloud can integrate and interoperate with another. This "plug-and-play" feature, as afforded by interoperability, is discussed in more depth in the chapter on transitioning to the cloud.

4

CLOUD NATIVE FOUNDATIONS

Cloud computing is based on host virtualization. Host virtualization technology takes two forms: virtual machines and containers. We discussed the former in detail in the introduction to the book. Here we consider the latter. Yet a more important reason for this chapter is that containers form an integral part of cloud native technologies. A formal definition of cloud native technologies is given by the Cloud Native Computing Foundation (CNCF):

> Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

> These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.[1]

To elucidate the above definition: cloud native technology is predicated on containers providing services in the cloud using immutable infrastructure,[2] and those services are invoked using application programming interfaces (APIs). In addition, the services have telemetry to make them observable and automation to ensure service changes occur easily and in a timely and seamless manner.

Cloud native components thus are comprised of containers (that provide distinct services as microservices), their management, their orchestration, and their observability. We consider these components in this chapter as a precursor to further discussion in later chapters.

Containers: A Historical Perspective

Container technology was born in 1979 when version 7 of the Unix operating system provided the concept of a jail in which an application could run in. This jail was enabled with the use of the chroot (change root) system, which restricted an application's access to a specific directory

Cloud native technology is predicated on containers providing services in the cloud using immutable infrastructure.

such that this directory was denoted as the application's root directory.[3] The jail system was refined in 2001 with the partitioning of resources such as memory and central processing units (CPUs) that would be available specifically to an application in the jail. In 2008, Linux Containers (LXC) used control groups—known as cgroups—to group processes in a manner that would prevent any one container from taking up all the resources of the machine; this meant that one container could not starve other containers on the same machine of the use of resources. LXC also used namespaces, which allowed a process to have a distinct set of users and the running process to have root privileges inside the container, but not outside its jail. LXC was the enabling technology for containers. It is still available today and is actively supported and being improved. Docker created a user-friendly API that was a wrapper over LXC, and so Docker containers were born in 2013. Today, Docker containers form the backbone of many microservices, although alternatives such as LXC and Linux's podman (pod manager) are also popular.

Why Containers?

Although virtual machines can be used as cloud native components instead of containers—and they sometimes are—they are less common because they take longer to

deploy and so are less suited to short-lived or on-demand microservices. Web applications and services usually need to use on-demand microservices to scale quickly to meet spurts in user demand. Hence containers are favored, although for some back-end services, you could use virtual machine–based microservices. Some components of a web service, however, such as databases, require immutable hardware to maintain state. Immutable hardware, or infrastructure, does not change. It is the antithesis of a container, which is often used as a temporary provider of computing power. So virtual machines, or managed services that the cloud service provider sells (such as RDS on AWS, Cloud SQL on GCP, or Azure SQL), which may be hosted on virtual or physical machines, are usually used for databases instead of containers.

One of the key advantages of using a container is that it provides consistency; thus your software will run on any platform on a container and each deployment of a container will be the same as another as long as they are built and run from the same image. Yet another advantage is that it saves space as well as management time when compared to a physical or virtual machine. This is because it can be a short-lived machine that is often run to execute a piece of code that serves a single purpose as a microservice. Because containers are lightweight, a number of them may be needed to fulfill a large demand for a given service requiring ease of scaling in and out. Although this agility is a

benefit of using containers, too many of them introduce a management and observability overhead. Thus managing and observing thousands or tens of thousands of containers will prove to be a difficult task, and we discuss this topic later. But orchestrating the containers so that they run on schedule when needed without any human intervention is quite easy using various tools such as the open-source container orchestration system Kubernetes; this level of automation saves time and effort. Let us assess some of the other benefits of using containers in a cloud native environment: statelessness, loose coupling, and scalability.

Statelessness

Short-lived containers do not maintain a memory of what they did or for whom as this allows you to scale your application using as many of them as needed to meet demand, and so are denoted as stateless. This is an advantage from a scalability perspective because the containers are then dispensable and so can be created at will to meet user demand without having to transfer state from one container to another. Should some form of memory need to be maintained during the time of a transaction when various containers run and die after having provided their services or during the provision of their services, then a cache or a database is used to maintain state. Such a transaction is considered to be stateful; one without state—known as stateless—has no memory of what one service does.

Loose Coupling

One of the principles of software systems design is "separation of concerns." Broadly, this means that you separate your software into components such that each component has a separate, specific concern or capability. Let a component then be a container that provides a specific service, and as such is a microservice. There are two ways you can think of coupling between microservices: at design level and at run time.

At design level, a loosely coupled service knows as little as possible about any other services it interacts with. Thus a change that is made to a microservice does not necessitate a change in another. This type of loose coupling is achieved because the services collaborate with each other through a service bus at a physical level and through a defined interface or protocol at the logical level.[4] Consequently, the service receives instructions or shares data in a defined and standard manner: each service can be thought of as a black box providing a specific function through a common, standard interface. A tightly coupled system, on the other hand, will require changes to all services that interact with a given service if any changes are made to that service. So a design that uses loosely coupled services cuts down on maintenance and refactoring time.

At run-time level, loose coupling relates to the dependency of services when interacting with each other. For example, suppose service A needs to talk to service B: if

service A sends an instruction or data to service B and expects to receive a response, then it depends on service B. However, if service B were down or busy and not responding immediately, should service A suspend all other tasks and just wait until service B responds or should it continue with other tasks and deal with service B's response when it arrives? Obviously, the latter is more efficient as service A is loosely coupled to B. The former case, in which service A blocks all other requests or tasks while awaiting B's response, is an example of tight coupling during run time. The loosely coupled system is said to be event-driven on the basis of events occurring for service A to respond to. Loose coupling, in an event-driven architecture, is therefore a very efficient way of using uptime and computational resources. Thus containers, being lightweight and deployable speedily when needed, are suited to an event-driven design when they are loosely coupled.

Scalability

Since loose coupling and statelessness allow you to spawn as many containers as you need to perform a given function to meet user demand, you can scale your application quite well using containers. You would then choose to scale each microservice within your application on the basis of the computing resources it is consuming; when a resource limit is reached, you spawn a container and its concomitant microservice to provide greater capacity.

Because scalability will be independent of any particular microservice, you will be able to scale efficiently by provisioning the resources in a more targeted manner without affecting the configuration or setup of other microservices. Not only does this provide horizontal scalability to meet the increased resource needs, it also obviates capacity planning.

Challenges

So far in this section we have considered several advantages of using containers and have touched on various reasons to use them. However, using containers does pose some challenges. These are listed below.

Integration: Integrating with legacy applications can pose a problem as you would need to update the legacy applications to fit with the microservice architecture and refactor their connectivity to use API-based integration.

Observability: Monitoring and reporting on a large number of containers can be challenging. You will need to design and create the metrics, tooling, and dashboards to observe them effectively.

Networking: Connecting containers as a service chain requires appropriate networking and IP address management within the subnets. This can become challenging where large numbers of containers are involved.

Storage: Since containers provide microservices as and when needed, any information inside a container will be lost when that container is killed or shut down. Retaining container-based information requires the adoption and use of container storage tools.

Troubleshooting: Since containers are deployed dynamically to different hosts, troubleshooting can be a challenge as you can easily lose track of which container is on what host and what it is doing. Thus container monitoring tools are needed to complement the host monitoring tools.

Costs: The added functionality, ease of use, and convenience of containers come at a price: the same, if not more, physical resources are needed to implement a like-for-like design with a container-based solution as with a physical one, and this represents an additional cost to you.

To address these challenges, several tools have been created to manage, orchestrate, and observe containers. Kubernetes is one such tool; we discuss it in the next section.

Managing Containers with Kubernetes

Working with a handful of containers is easy, but working with hundreds or thousands of them becomes a major

challenge. You therefore need a container management tool such as Kubernetes to create, schedule, kill, and observe your containers in order to realize their promise of scalability. Kubernetes started life as Borg, which was Google's internal cluster management tool.[5] In 2015, Google released its documentation on Borg, which was being used by Google's engineering teams internally. Thereafter Google donated Borg's specifications and code to the CNCF, which then released Borg into the public domain as open-source code in the form of Kubernetes.

The key premise of such a container management tool is automation, and its main benefits are listed below.

Abstraction: Container management includes tools for storage, security, scheduling, and monitoring to reduce the time and complexity when working with containers.

Ease of use: Administration of the containers becomes simplified and easier through the use of container management tools.

Automation: Container management automates a number of processes ranging from load balancing to orchestration. This means that you can scale your solution automatically and conveniently. Thus automated scalability is yet another advantage of such a tool.

Continuous health monitoring: Container management provides automatic health checking of applications

hosted on the containers as well as the containers themselves in terms of resource utilization and failure.

Kubernetes schedules workloads across many hosts by orchestrating the running of containers. To do this, it allows you to define a desired state of operation in terms of the workload and it then reconciles the current state to your desired state by either spawning or killing containers, as required. And it provides a command-line interface (CLI) for you to interact with the Kubernetes engine. In Kubernetes parlance, a pod is a container, and a collection of pods resides on a node. A node is a machine, either physical or virtual, that runs pods; the number of pods and their creation are automatically handled by the Kubernetes scheduler. Please bear in mind that the Kubernetes scheduler creates new application pods without considering the current actual utilization of nodes, and this is a shortcoming at present. In any case, Kubernetes is good at scaling horizontally to handle large increases in traffic and workloads, and it does this by adding more pods and nodes from the available computational resources. However, it lacks the ability to scale vertically, and one way of overcoming this is to install and use Kubernetes on virtual machines in the cloud that are part of an autoscaling group so that additional machines can be made available to it when needed.

Cost Optimization

From a real-world standpoint, cloud computing costs can very easily spiral out of control, and cloud native computing can be even worse. Other than creating your own private cloud—and this can be a good choice if your cloud computing costs approach $1 million per annum—you have very few options at your disposal to contain costs. Obvious approaches are to use tooling that optimizes or reduces costs and to adopt processes that discipline your use of cloud computing resources. In this section we discuss approaches that have a large impact on cost reduction in a cloud native environment.

When implementing a microservice, the number of containers you use will define the costs you will incur. If you need to scale in order to meet user demand, your costs will grow proportionately as more containers are spawned. The number of containers that need to be spawned will depend on the resources used per container, such as memory and CPU cores, as well as on how resource-hungry your microservices are. If you use a programming language that executes a piece of code faster than other languages, then the microservice that has been implemented with the faster code will need fewer resources than one that is slow to execute. So, one way to contain costs is to use a programming language that will allow you to execute your

From a real-world standpoint, cloud computing costs can very easily spiral out of control, and cloud native computing can be even worse.

code faster and to use programming constructs such as multithreading to implement your microservice.

Another factor is networking. If your microservices use or need greater network bandwidth, then your costs will increase. Therefore, you should try to use batch processes for transferring data whenever possible. In general, when you have data arriving at intervals, whether regular or irregular, you may want to cache them so that you may then perform the bulk transfer of the cached data as a batch. This will enable you to create batch transfers of data that do not exist in a bulk form to begin with. Of course, this may not always be possible, especially with an event-driven architecture. In such cases, perform batch data transfers of static or reference data only since the dynamic data will be part of the events in your event-driven architecture.

What's Next?

To recap, cloud computing forms the foundation of cloud native applications as it is used to host containers that provide microservices, and these are accessed through declarative APIs. Additionally, tools such as Kubernetes are used to manage, orchestrate, and observe the microservices and containers to provide self-healing, scalable services. We continue this topic in the next chapter, where we delve deeper into microservices and their design patterns.

MICROSERVICES AND THEIR DESIGN PATTERNS

Although no formal definition of a microservice exists, something that is quite close comes to us from James Lewis and Martin Fowler, who wrote a seminal blog piece on microservices in which they listed nine constituent characteristics of a microservice.[1] We examine these characteristics briefly in this chapter and then discuss microservice design patterns.

Characteristics of Microservices

The following nine characteristics define microservices and their use:

1. **Componentization via Services** A monolithic architecture has a single instance in which all its

functionality is implemented, and it scales by replicating that instance on multiple servers. A microservices architecture, on the other hand, decomposes the application's functionality into separate service elements that are implemented such that each service element has its own specific instance; thus a composition of service instances together makes up the application functionality. Such an application scales by distributing its services across servers.

2. Organized around Business Capabilities When implementing a large application, the general approach in the past has been to break the implementation up along technical capabilities. So, for instance, there would be teams of front-end developers, back-end ones, database administrators, middleware specialists, and so on, all of whom worked within their own teams. With microservices, the application's implementation is assorted along business capabilities and functions instead, so that specialists from each of the technical capabilities work together as a team within each business function.

3. Products, Not Projects A project-based approach tends to demarcate and assign development, testing, delivery, and support to different teams across an application's life cycle. A product-based approach, on the other hand, is inspired by Amazon's approach of

"you build it, you run it," whereby a single team has ownership of the product across its entire life cycle. This ties in well with the organization of the product's implementation along business capabilities, as discussed in the previous item.

4. Smart End-Points and Dumb Pipes For communication between processes or services, use dumb pipes instead of smart pipes. Dumb pipes just do the messaging, such as publish-subscribe or queuing, whereas smart pipes have a lot of built-in intelligence, such as message translation or formatting and rules-based message routing. Thus a lightweight message bus, often termed an event bus, is preferred over an enterprise service bus (ESB).[2] This helps the microservices remain decoupled from each other and own their own domain logic. And the end-points are smart using REST protocols that favor statelessness so the microservices are replaceable. Replaceability serves two purposes: first, it allows easy maintainability and refactoring as one need only replace a microservice instead of the entire codebase, and second, it supports scalability as one can replicate those microservices in proportion to demand.

5. Decentralized Governance Instead of having centralized decision-making in terms of the programming language to use, the libraries to use in

your code, or the tools that help you build and test your code, responsibility is devolved to the team that works at the domain level. This means that a microservice can be different internally from another that provides a different service to it, but both share the same contractual boundaries in their communications internally between them, as well as externally, with other applications and systems. Thus, the "you build it, you run it" paradigm is empowered as the team that builds it not only runs it but also takes ownership and control over it.

6. Decentralized Data Management Usually, data are kept in a central repository that serves all the business domains and at the same time provides transactional consistency of the data. The general outcome of such a centralized approach is that the data repository mutates into a data warehouse and any transactional commits, especially large ones, can take an inordinate amount of time. Also, such a centralized approach does not favor the domain-driven approach because a central view of the metadata is imposed on all the domains; for example, the accounting department may have a different definition of who a customer is from the support department. To realize a domain-driven approach more easily, data management needs to be devolved to each of the domains and the application or service that supports it. But there is a disadvantage to this when updating data:

transactional consistency becomes much more difficult to maintain.[3] As a result, eventual consistency is favored, with the understanding that businesses operate in such a manner in any case.[4] Thus microservices prefer letting each service manage its own database, an approach known as "polyglot persistence."

7. Infrastructure Automation To ensure quality, you need to test. And you need to test at various stages such as when developing, integrating, and releasing your application because catching errors at an early stage of a product's life cycle is less expensive to fix than catching the errors at a later stage. So continuous integration and continuous delivery (popularly called the CI/CD pipeline) is espoused. And to build-test-integrate-test and deliver in a continuous manner, you need to automate the entire pipeline. So automated building, testing, and deployment are characteristics of a microservice architecture.

8. Design for Failure Because an application is built using different microservices, and the application scales by spawning multiple instances of those microservices, the killing of a microservice (in order to scale down) or the failure of a microservice should not affect the functionality of the application. Thus it should be designed with failure in mind. Real-time monitoring of the application is therefore emphasized by the microservice paradigm. This includes both architectural

elements (e.g., how many requests per second a database is getting) and business ones (e.g., how many orders per minute are received). In addition, resiliency is emphasized; for example, Netflix uses Chaos Monkey to terminate live instances of microservice containers or their virtual machines in the production environment.[5] The idea is to incentivize engineers to build resilient services.

9. Evolutionary Design To continuously improve, you need to have the capability to make changes continuously and in an evolutionary manner. If by design you have an architecture that embraces such change, then it is geared toward evolutionary change. Having components as microservices means that you can release at a speedier rate and more easily because you can make changes at a more granular level; that is, you can change at the microservice level rather than at the application or system level. Thus you need only change and deploy those microservices that you modify. Another use case is the ability to add and remove functionality when needed. For example, a company running a marketing campaign would have new microservices deployed during the campaign and then removed from the application afterward. One form of thinking suggests that you bundle services that change more often in the same microservice, but this approach should be adopted only if it supports the domain-driven design approach.

Let us denote a cloud-based application that incorporates the above nine characteristics to be one that follows the microservice pattern. In the next section, we discuss some design patterns related to such a microservice pattern, starting with the database-per-service pattern.

Microservice Design Patterns

A design pattern solves a design problem and provides a reusable solution to speed up the design process. A microservice design pattern is a design pattern created specifically for a microservice-based architecture.

Let us consider a selection of the patterns shown in figure 11. To describe the patterns in a standardized and consistent manner, let us adopt the following structure:

1. Context (Under what circumstances would we use the pattern?)
2. Problem (What is the problem statement that the pattern addresses?)
3. Solution (How does the pattern solve the problem?)
4. Benefits (What are the pros of using the pattern?)
5. Drawbacks (What are the cons of using the pattern, and are there any alternative solutions?)

A design pattern solves a design problem and provides a reusable solution to speed up the design process.

Let us commence the discussion with the database-per-service pattern using the above template.

Database-per-Service Pattern

Context An application is made up of microservices, and some of them need to persist data. How do you ensure that any two microservices will have seen the same data in a consistent manner?

Problem Do you use one large database and create a lock on a table or record when its data are updated, or do you ensure that each service has its own database?

Solution Keep each microservice's persistent data private to itself and accessible only through its API, gRPC, or event bus. You can use a lightweight database, for example SQLite, for each microservice, a NoSQL database server instance or collection, or a private schema or table for that microservice if using a relational database.

Benefits Doing so ensures microservices are loosely coupled since any changes made to one microservice's database do not affect other microservices. Also, each microservice can have the type of database that is suited to its needs. For example, a service that does text searches could use Elasticsearch, whereas one that works with a social graph could use Neo4j.

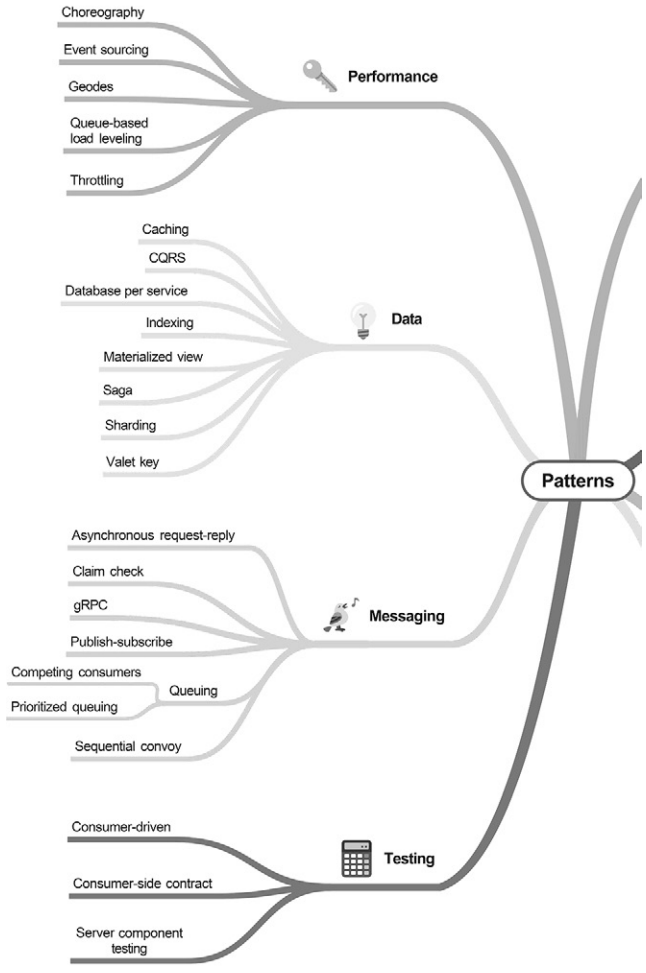
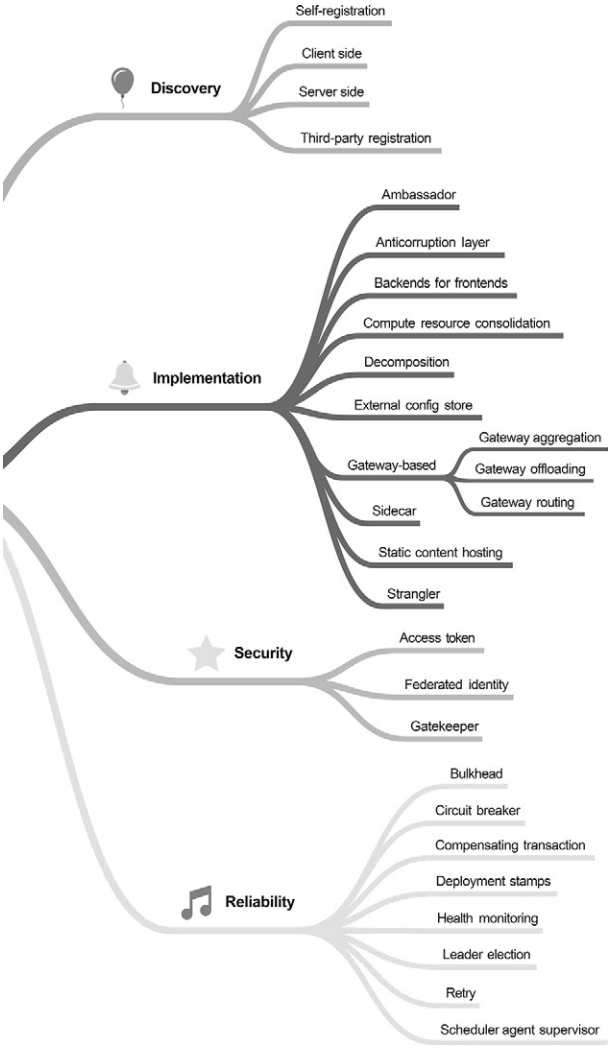


Figure 11 Overview of microservice patterns.


Discovery
Self-registration
Client side
Server side
Third-party registration
Implementation
Ambassador
Anticorruption layer
Backends for frontends
Compute resource consolidation
Decomposition
External config store
Gateway-based
Gateway aggregation
Gateway offloading
Gateway routing
Sidecar
Static content hosting
Strangler
Security
Access token
Federated identity
Gatekeeper
Reliability
Bulkhead
Circuit breaker
Compensating transaction
Deployment stamps
Health monitoring
Leader election
Retry
Scheduler agent supervisor

Drawbacks Transactional consistency that spans multiple microservices is difficult to implement. Another drawback is that skills to support multiple types of databases would be needed, and yet another drawback is that queries that require join constructs across multiple databases become much more difficult. However, there are various patterns that address these issues. For example, the saga pattern may be used to implement transactions that span services. For queries that span services, you could consider the API composition pattern or the command query responsibility segregation (CQRS) pattern; these are discussed below in sequence.

Saga Pattern

Context You have implemented the database-per-service pattern and would like to have transactional consistency as some of your business transactions use several microservices, each with its own private database.

Problem How do you implement transactions that span multiple microservices? You do not want to use a two-phase commit pattern because it is a blocking process, and you want your microservices to be nonblocking and thus decoupled at run time.

Solution Use a saga of local transactions; break up the business transaction into local, microservice transactions.

Then each microservice provides a status of its local transaction. The business transaction is completed when all local transactions have completed successfully. If a single local transaction were to fail, then all the other related local transactions would be rolled back using a set of compensating transactions.

Benefits Transactional consistency across multiple microservices is achieved.

Drawbacks Greater complexity in programming results as a failed saga would need to be rolled back, and also, each local transaction would need to provide a status response.

API Composition Pattern

Context You have implemented the database-per-service pattern and want to implement queries that join data from multiple microservices.

Problem How do you implement queries that span multiple microservices in a microservice architecture?

Solution Implement a query using a special microservice, commonly called an API composer, which queries the microservices owning the data and performs an in-memory join of the results. The API composer remains free of the business logic, which remains with the microservices

implementing that business function for its particular domain. Used in conjunction with the API gateway pattern, each service composition can be defined as an aggregate of services behind the API gateway, which effectively acts as a façade. In this way you can hide legacy systems and replace them at some later date, and can provide a consistent design of the overall API to clients.

Benefits Implementing the API composer microservice simplifies the querying of databases in a microservice architecture. Because an in-memory cache is used, query joins are performant.

Drawbacks Should the resulting datasets of a query be large, then performance is affected since an in-memory cache is used. The CQRS pattern can be an alternative solution.

CQRS Pattern

Context You have implemented the database-per-service pattern or the event sourcing pattern and you want to perform queries that join data from multiple microservices.

Problem The CQRS pattern (command and query responsibility segregation) has the same problem statement as the API composition pattern, which is an alternative solution: how to implement a query that retrieves data from multiple microservices.

Solution Define a view database, or schema, that straddles the private databases of the microservices and provides an aggregated view of the data domain. The view database should be a read-only replica of the other databases in aggregate, and is kept up to date by subscribing to domain events published by the microservice owning the data.

Benefits This pattern has five main benefits:

- It supports multiple denormalized views that are scalable and performant.
- It improves separation of concerns between microservices.
- It enables an event-driven architecture to be implemented and is especially useful when event sourcing is used. Thus this pattern is often used with event sourcing.
- Queries are simplified.
- Security is improved since the pattern allows only read-only queries and the writes are performed locally by the relevant domain microservice.

Drawbacks This pattern may not be relevant if the domain or the business logic is simple. It also introduces a further lag to the eventual consistency of the application.

Publish-Subscribe Pattern

Context You want to enable your microservices to communicate with each other asynchronously, in an event-driven manner, and without coupling the receivers to the senders.

Problem In a distributed, event-driven architecture, how should the microservices communicate or send status events on the event bus without any coupling between the senders and receivers?

Solution Use a publish-subscribe (also referred to as pub-sub) paradigm for communication. In this method a publisher publishes messages to a message bus using a topic that serves as the subject of the message. Interested subscribers then subscribe to the published topic to receive messages from the publisher on that topic. A message broker, also known as a pub-sub broker, then routes messages from publishers to subscribers in an asynchronous manner using message queues (for decoupling) and topics (as subject groupings to subscribe to).

Benefits This method provides run-time decoupling, improves scalability, and enables seamless communication between microservices. Each microservice can have its own publication topic. Another benefit is that it enables the creation of an event bus and event sourcing.

Drawbacks It is not amenable for use in an application that has just a handful of microservices. Instead, use a message queue in such a situation. Another alternative could be to use the gRPC pattern, but the concept is slightly different: with gRPC you execute code on another microservice, whereas with pub-sub and queue-based messaging you send a message to another microservice and it decides which code to execute, so you need to think of the boundaries for code execution, especially in a domain-driven design.

Event Sourcing

Context An application implements the database-per-service pattern and the saga pattern. Its microservices need to update their private databases and send messages or events to report the status of the update.

Problem How should a microservice reliably (i.e., preserve atomicity between the update and its corresponding status message) update its private database and send status messages? (Because of its blocking nature, two-phase commit is not considered.)

Solution Use an event store to store events, such as a microservice updating its database, and then publishing that event to other microservices that subscribe to the event store's topic. Each microservice would then have its own event topic for publishing on the event bus and the event

store would aggregate all such events, which then would be sourced to interested microservices that subscribe to the relevant topic on the event bus. Thus, the event store becomes a central store of events for all the microservices.

Benefits Event sourcing has the following benefits:

1. It implements an event-driven architecture that publishes events reliably whenever any state changes occur.

2. It provides a reliable audit log of the changes made in a business domain.

3. It enables temporal queries to be implemented so that the state of a microservice is known at any time.

Drawbacks The event store requires many queries to reconstruct the state of a particular microservice. Thus the application needs to use the CQRS pattern to implement queries.

API Gateway Pattern

Context You have a database-per-service implementation where the data needed are distributed across various microservices that implement domain (i.e., business function) logic. These domain microservices may have different code implementations and interfaces but you want to

present a single, consistent view to external client applications that communicate to your application using a RESTful API.

Problem How can a unified front end be presented to external clients that talk to your application's microservices, using a RESTful API?

Solution Implement an API gateway that acts as the single-entry point for all clients. The API gateway handles requests in one of two ways: (1) some requests are simply proxied and routed to the appropriate internal microservice, and (2) other requests are fanned out to multiple microservices. An API gateway might also perform protocol translation. The API gateway can also provide a RESTful API to external client applications even though the microservices internally use a combination of protocols. Where appropriate, the API gateway can translate between the RESTful external API and the internal gRPC-based APIs. The API gateway may also incorporate some extra functionality:

- Authentication: verifying the identity of the client application making the request.
- Authorization: verifying that the client is authorized to perform that particular operation.

- Rate limiting: limiting the number of requests per second to or from a client.
- Caching: caching responses to reduce the number of requests made to the microservices.
- Metrics collection: collecting metrics on API usage for billing or marketing purposes.
- Logging: logging requests to the API gateway in the event an audit trial is needed.

Benefits The main benefits are as follows.

- Use of an API gateway insulates client applications from implementation details.
- It provides an optimal and unified interface to each client.
- It enables client applications to retrieve data from multiple microservices with a single round trip. Fewer requests mean reduced overhead, and so user experience is improved.
- It is essential for mobile applications.
- It translates external communication protocols to internal ones.

Drawbacks The main drawback is that there is an extra middleman between the client and the service, and this introduces a slight lag or latency into the request-response transaction. However, this is small compared to the savings achievable with a reduced number of round trips when multiple services are involved. Another drawback is that it adds an extra layer of complexity to the solution.

Circuit Breaker Pattern

Context A microservice invokes another one but the called service is busy handling other requests or simply fails to respond. How do we stop one microservice cascading its faults or errors to those that invoke it?

Problem How do you prevent a failure in the network or in a microservice from cascading to other microservices?

Solution When invoking a microservice, use a proxy that mimics an electrical circuit breaker. When the number of failed attempts reaches a given threshold, the circuit breaker proxy trips and informs the calling microservice. It then implements a time-out period during which any further attempts to call that microservice will fail immediately. Once the time-out expires, the circuit breaker permits a limited number of requests to pass through on a test basis. Should these requests succeed, normal operation

resumes; should the failures persist, the time-out period is cycled to the next iteration. The circuit breaker, as a proxy, keeps a count of all connection attempts and retries as part of its functionality.

Benefits This pattern enables microservices to handle the failure of other microservices that they invoke.

Drawbacks The trade-off between false positives and greater latency is difficult to make when selecting optimum time-out values.

Sidecar Pattern

Context You want each microservice to monitor its own health and to report it on the event bus should issues be detected.

Problem Since your microservices should implement code pertinent to a business domain or one of its functions, you do not want to burden it with peripheral tasks such as health checking, logging, event publishing and other such tasks that ought to be common to all microservices.

Solution Create a supporting function that connects to the main microservice function to perform mundane tasks. Often, the sidecar function is implemented as a separate microservice in its own container and is referred

to as a sidecar or sidekick container. The sidecar can access the same resources as its connected microservice and so should monitor system resources used by both itself and its domain microservice.

Benefits A sidecar is independent from the connected microservice and so you can use an instance per each microservice. Because of its proximity to its domain microservice, there is minimal latency when communicating between them.

Drawbacks You should consider whether a simple library could replace the functionality you wish to implement with a sidecar. Also consider the interprocess communication mechanism used by the sidecar; generally, gRPC or the event bus should be preferred.

CLOUD COMPUTING

A Paradigm Shift?

In this chapter we step back and ask some pertinent questions: Why cloud computing? What does it give you? What is so special about cloud computing? And how will it affect you, your work, and our society? Just as Microsoft Windows became ubiquitous at home and at work, and changed our lives, cloud computing represents a paradigm shift. This is because cloud computing is an enabling technology that bypasses many functions provided by your computer, the software installed on it, and your workplace's IT and finance departments.

In this chapter we consider cloud computing's paradigm shift from three different viewpoints: (1) how it could affect you socially and personally, (2) how it will affect you in your work, and (3) how it will affect businesses.

This chapter might be construed as offering a brief look at those viewpoints, whereas chapter 10, on transitioning

to the cloud, will provide you with practical tools and frameworks to realize these paradigm shifts.

Social Paradigm Shift

How you spend your leisure time and how you live, both at a personal level and as part of society, are summed up in the term "social life." How social life could be affected by cloud computing is the subject of this section. To examine that paradigm shift in your social life, let us consider the three types of clouds that would be the primary change agents: the societal or community cloud, the personal cloud, and the cloud of things.

Societal Clouds

A societal cloud is one that serves a group of people that have something in common. As examples, that common element could be along geographic lines (e.g., township, state, national, or international boundaries), hobbies (e.g., philately, numismatics), languages, or interests (e.g., trade unions, scouting, sports). Your membership in a societal cloud is defined by the common element you possess.

An international societal cloud could be defined for NATO, UNO, EU, and other such bodies. The citizens of countries belonging to the international body would then

be members of that cloud. Common benefits or issues could be considered by the cloud members, which could host, for instance, discussion boards, instant messaging, storage of shared documents, and video conferencing. All this could be done in a secure environment.

Similarly, a societal cloud could exist at a national level for health care, training, politics, farming, education, and the like. The data collected could be depersonalized and aggregated to provide trend analyses. For example, in health care, the information collected with regard to a particular disease could be analyzed in terms of its concentration in particular areas, age groups, or social or income brackets, in real time and in an automated manner. The information could be made available freely to anyone researching the prevalence of that disease in the aggregate. That information could be used to link multiple disease states and their effect on one other. Moreover, the range and dispersion velocity of infections could be gathered from such a medical cloud to predict the spread of a disease across a country or region. This information could then be used to prepare the distribution of immunization drugs or the deployment of medical resources.

A community cloud is one that provides a service to those who have a common interest. The common interest could be an avocation such as farming, weather forecasting, or participation in trade bodies, banking, law, or

publishing. In a sense, with the internet and various websites that cater to common interests, we already have internet-based communities, known as social media. Converting these to societal clouds is more a case of using cloud elasticity and an appropriate price model. So the societal cloud is less of a paradigm shift from an individual's perspective. However, having a societal cloud that is a community cloud comprising other clouds (a societal cloud of clouds) enables member services to be defined in a unique and individual manner. Thus the societal cloud, at a high level, acts as a service broker for members belonging to a societal cloud and can tailor service delivery to suit an individual's background and interests.

Personal Clouds

A personal cloud is one that belongs to you for your use. You may already have come across such clouds in the form of Apple iCloud, Google Drive, or Microsoft OneDrive. These clouds allow you to store files such as documents, eBooks, pictures, and music so that you can access those files from any device and any location. However, you should have a large choice in the future with regard to various use cases for a personal cloud.[1] In general, we can consider a personal cloud in terms of the following three use case categories: leisure and well-being, finance, and shopping. Some of the examples cited may seem futuristic, but they demonstrate the range of possible uses for a personal cloud.

Personal Cloud for Leisure and Well-Being

The current plethora of storage clouds such as iCloud fall in this category. In the future, as storage becomes cheaper with economies of scale, you should see video-streaming personal clouds that retain your collection of movies or video clips. This would be tantamount to having your own personal YouTube service.

Other personal clouds, such as a health wallet, are already available; they store information on the health providers you visited over a period of time, the results of your health screenings, and the medical costs associated with the health checks. Various other devices, such as your weight scale, pedometer, or blood pressure monitor, could be hooked up directly to your personal cloud to provide you with immediate alerts should your readings breach an ideal threshold. (While all those health devices hooked up might evoke an idea of an internet, or cloud, of things, we classify such a cloud as a personal cloud rather than a cloud of things because such devices monitor you or are related to you and you alone.) A health service provider could depersonalize and aggregate everyone's health data to analyze the best fitness and health plans for you. Alternatively, the analytics could be sold to a health insurance company, which would then be able to predict the health care costs associated with someone in similar circumstances to you.

Another example of a personal cloud for leisure comes from motoring. A car could capture your driving profile

As storage becomes cheaper with economies of scale, you should see video-streaming personal clouds that retain your collection of movies or video clips. This would be tantamount to having your own personal YouTube service.

and send it to your personal cloud. The information could be, for example, the average speed at which you drive, the locations where you drive, your general driving style (aggressive or conservative), and the number of accidents you have. Some cars already capture this type of information, but it is stored onboard rather than made available to you in your own personal cloud. An automobile insurance company could then use this information to tailor its insurance offering to your driving profile. Yet another example is having smart streetlights in an area that is not well developed. The streetlights could sense your presence through proximity sensors and provide lighting to you on the basis of whether you have paid the local taxes or the road tax; those motorists who have not paid these taxes would not have the roads lit up for them unless they paid instantaneously, using technologies such as near-field communications, to have the lighting turned on. Of course, this assumes that a pact is not formed between the compliers and noncompliers to journey together in a group to foil the lighting scheme.

Personal Cloud for Finance

A personal cloud that receives your bank statements and credit card transactions could provide you with a balance sheet and a budget on the fly. Then, at the end of the financial year, when your personal finance cloud obtains income tax regulatory information from the government's INaaS

cloud, this cloud could create an income tax statement for you. As some governments' tax authorities allow the remittance of electronic tax returns, your cloud could file the tax statement with the relevant government department upon your approval. Thus the chore of creating a tax return would be automated for most individuals and they would not need an accountant. Another use for a personal finance cloud would be to provide you with an integrated view of all your investments across various pension funds, IRA schemes, and brokerage accounts. This would enable you to assess at a glance what your investments' performance is over a given period at a moment's notice.

With various countries exploring the adoption of digital currencies, your personal finance cloud could double as a wallet or be linked to your wallets that contain digital currencies and tokens. These wallets could even represent credit and debit cards so that you could use them for various purposes. For instance, wallet 1 could be for dining out, wallet 2 for home expenses, and wallet 3 for clothing. Or you could have the wallets affiliated with various retail outlets: wallet 1 for eBay, wallet 2 for Amazon, and so on. You might sacrifice some privacy as a result of adopting these technologies, but there is a trade-off between privacy and convenience. The right balance is somewhat difficult to decide on, but it needs to be considered by all of us.

Personal Cloud for Shopping

Your personal shopping cloud could store your shopping preferences based on your shopping history across all stores. It would then analyze your buying patterns and alert you to what you need to buy in a timely manner by using predictive analytics. It could even scan discounts or offers at various stores, physical or electronic, to provide you with a purchasing choice. Further, it could manage your electronic wallet so that you could pay for the goods quickly and easily. A lot of work is currently being done by various companies on electronic payment protocols, and some of these payment protocols could be integrated with your personal cloud for shopping or with your electronic wallet.

Cloud of Things

A cloud of things is a cloud service that helps in the management or use of a thing (a nonliving entity) by one or more living entities. (Those 'things' are connected via the internet of things.) For example, you could have a cloud for your house. It could receive information from several sensors related to security, the presence of smoke, proximity, light, and other installed devices, and also automatically control other things such as opening and closing curtains, fire alarms, lighting, and heating for you and other residents of that house. Moreover, depending

on the room, each occupant of that room could have a personalized profile in terms of when curtains would be drawn or lighting turned on. Likewise, for your work environment, you could have a facilities cloud. Such a cloud would be an example of a BPaaS because of the physical processes it would manage automatically to benefit you, such as the drawing of curtains and monitoring of lighting. Another such example would be a meeting room cloud that kept a logbook of the room's availability so that you could book the meeting room for a given period provided it was available. The cloud could further inform various parties, such as security or catering, of its occupancy to enhance or ease the use of the meeting room. The meeting room cloud itself could belong to an aggregate cloud comprising meeting room clouds, and they could then act in concert so that if a meeting room was unavailable at a certain time, you would receive a choice of suitable available rooms. This way you would have a selection of meeting rooms that fulfilled your criteria in terms of availability, room size, or location, for instance. The meeting room aggregate cloud could in turn belong to the facility cloud, which itself would be a composite cloud. Cloud relationships (please refer to chapter 2) can play a prominent role in the cloud of things because you could have various relationships—such as encapsulation, federation, composition, and aggregation—between clouds of things to create other clouds of things.

Work Paradigm Shift

Two major trends are currently taking place in the workplace:

- Workstations are being replaced by zero or thin clients.
- Ubiquitous computing, allowing the use of any device for work.

Workstations (laptops and desktops) are being replaced by machines that do not have applications installed on them. Such machines are known as zero clients if they have the operating system embedded on the silicon chips or thin clients if they require an operating system on disk. If a workstation has no applications installed, you would need to use cloud-based applications to perform your work on the thin or zero client workstations. The clouds hosting those applications can have varied deployment models: they can be private, public, community, or personal clouds, for instance. Generally, for productivity-related applications such as Microsoft's Office Suite or email, you would use a public cloud service, whereas for your own bespoke applications, you would use a private cloud service. But the applications do not necessarily have to be cloud-based; they can be hosted on servers in the data center using traditional physical or virtual computing. As long as

the application allows you access using a web browser, you should be able to use it regardless of the underlying technology used for hosting it. The benefit of a zero or thin client computing environment is that your company's IT department does not have to manage all those applications installed on a large number and variety of workstations. Instead, it would manage just one application in the cloud or provide access to an application provided by a third party's cloud service. Another benefit is that because the workstations do not contain any local disk or data storage mechanism but instead use a cloud-based data store, the work information is stored in a central and, hopefully, more secure data storage cloud. This means that if the workstation were to be lost or stolen, the company's data would not be compromised. Indeed, in such a case, the zero or thin client workstation would be less expensive to replace as most of the application hosting, storage, and computing takes place elsewhere, in the cloud.

Ubiquitous computing took hold in universities that needed to cater to the plethora of computing devices that students brought to the campus. Providing access to university-provided applications and information on various students' devices meant that a university's IT department needed to have a secure way of allowing access to the university's resources on devices that the IT department did not manage or have any control over. As the technology developed, it became known as bring your own device

(BYOD), which is currently being adopted by businesses to deliver IT to their employees. But ubiquitous computing is much more than BYOD. Ubiquitous computing means that you can compute and access corporate information from anywhere, not only from the campus or the workplace, by using any device and at any time. The introduction of such a computing framework means that the IT department increasingly becomes a cloud service broker that maintains a service catalog of allowable cloud computing applications for an employee to use for work purposes. Those cloud-based applications then can be used anywhere, at any time and on any device. Any data that need to be used or stored locally on the device are stored in a secure area within the device, known as a sandbox. The sandbox is created when you become an employee and deleted when you leave the company. It stores the information so that only allowable applications can access it; further, the data are generally encrypted in the sandbox.

Organizational and Business Paradigm Shift

Most businesses have an IT department that looks after central applications such as mail servers and web servers. The more business-related IT is performed locally by the business units or groups that deal directly with delivering a product or service to the customer. So you have a hub-and-spoke model wherein the centralized IT acts as a hub and the various business groups work autonomously on

the periphery as spokes. If the central IT function is unaware of applications used by the various units or groups in the business, then those applications and their computers tend to be classified as shadow IT. If something were to go wrong with such an application or with the computer hosting it, there would be a problem in terms of supportability. A strong central IT department would refuse to support shadow IT whereas a weak one would support it at the cost of extra effort and money expended on learning about the system. In any case, shadow IT represents a potential security breach as well as an additional expense because of its nonstandard nature. For companies, shadow IT therefore represents a business risk.

As more applications become available via cloud computing, business units are likely to increase their reliance on shadow IT because the spending threshold will be lower, as figure 12 shows. Because of the greater level of abstraction provided by cloud computing, the skill set required to manage and support the shadow IT is less, and this helps reduce shadow IT costs. Thus almost all the IT used in a company is bound to become cloud-based. The central IT department, to survive, will need to evolve and become a cloud service broker. Doing so will ensure that the business departments are enfranchised to work in a semi-autonomous centralized structure as far as IT is concerned. This will further ensure that shadow IT becomes mainstream IT and so will no longer be classified as

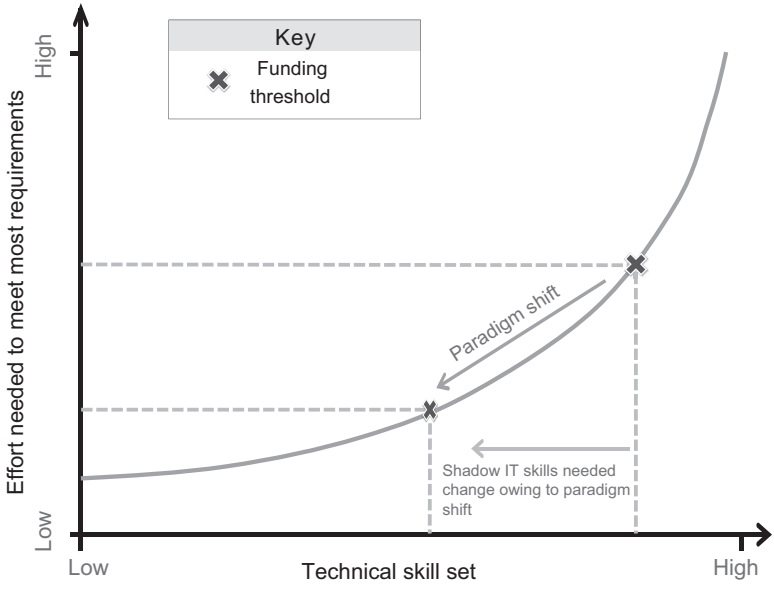


Figure 12 Finding the threshold for shadow IT.

shadow IT. As a cloud service broker, the central IT department would become a specialist branch of the purchasing department since most of the IT and computational resources would be bought as services or used on a pay-as-you-go basis. The increased commoditization of IT as a result of cloud computing and related technologies will make this more possible as the technical skill set requirement to use and purchase IT diminishes, as shown in figure 12. The corporate IT department's function will evolve

to maintaining cloud service contracts with cloud service providers, whose services are listed and described in a cloud service catalog that the IT department maintains.

How would the IT department then measure the value of a cloud service? How would it compare various pricing schemes from different cloud service providers to select the services to be made available through its cloud service catalog? We consider these questions in the next chapter when we discuss price and value models.

7

PRICE MODELS

Every undertaking or service has a cost and a benefit associated with it. Consuming cloud services is no different. This chapter considers the cost element of your using a cloud service and the price you pay for it after assessing various pricing regimes, known as price models. (Price models are also referred to as pricing models.) To offset the price you pay for the cloud service, you need to realize a commensurate benefit from it. To articulate those benefits, please refer to chapter 10, "Transitioning to the Cloud."

Understanding price models will enable you to compare various cloud computing services in an objective manner. Although most public cloud providers will not provide you with a choice of price model, it is important to know about various pricing mechanisms, especially if you are commissioning a private or a hybrid cloud.

The chapter ends with a discussion of the various financial metrics that you could use to evaluate cloud services from a financial perspective.

Price Models

Price models provide a means of establishing the price that you pay to receive the value of a product or service. A cloud service provider computes the costs of provisioning and operating a cloud service using a cost model. The cost model is then converted to a price model. The type of price model selected will depend on the cloud service provider's business model, marketing strategy, and revenue expectations.

Every price model starts life as a cost model, which is a financial model that the cloud service provider creates to find out how much money to outlay on a particular cloud service in creating it, operating it, and then refreshing it with newer technologies after three years. Three years is the usual life span of technology before it becomes outdated, and five years is generally the absolute maximum the cloud provider has before replacing the technology.

The cost model includes such factors as inflation, exchange rate variations (if applicable), depreciation, electricity costs (these can be significant because of the power and cooling required by the servers), floor space costs,

Three years is the usual life span of technology before it becomes outdated, and five years is generally the absolute maximum the cloud provider has before replacing the technology.

software license costs, labor costs, and capital costs to buy and operate the servers. Margin and a factor for risk are added to the sum of all the costs to arrive at a price. The price has two components: nonrecurring and recurring elements. The nonrecurring element of the price is converted to a recurring element by amortizing the net present value into a series of recurring cash flows. (The mathematics for doing so is considered toward the end of this chapter under the heading of "Net Present Value.") These cash flows are then added to the recurring element to arrive at a monthly price point for providing the service to you. This recurring price point is expressed as the price model, and this price is used to sell and market the cloud service.

There are a variety of price models in existence. Broadly, let us categorize them as utility-, service-, performance-, and marketing-oriented models. Although cloud computing today mostly uses utility- and service-based price models, a wide range of models are considered because financial and business innovation is bound to catch up with technical innovation, enabling some of the less-used models to enter the cloud computing domain in the future. You may even choose to use or specify your own model in case you need to commission your own private, community, or hybrid cloud, after learning about and evaluating all your options concerning the various price models discussed below.

Utility-Based Price Models

Utility models are metered price models whereby usage of the service is monitored, and the user pays accordingly. Originating in the price plans that utility companies have adopted, they are characterized by regular payments, often monthly, to the cloud service provider. Three utility price models are discussed here: consumption-, transaction-, and subscription-based price models.

Consumption-Based Price Models The consumption-based price model is a commonly used model for IaaS and PaaS. You pay for the computing resources that you use: for example, the amount of storage (in megabytes or gigabytes), computing or processing power (in terms of CPU cycles or number of processor cores used), and memory (in megabytes or gigabytes). An average consumption rate of these resources is computed over a day, week, or month, and you pay for the average utilization. This is a rather crude model that does not scale well for SaaS, INaaS, or BPaaS, since for these resources you want to be charged in a meaningful way when it comes to how your business operates. For instance, for an INaaS service that provides you with the latest tax rules, you really ought not to care how many CPU cores or how much memory is used in delivering that information to you. But for the cloud service provider, there could be other factors that contribute to the cost of providing the service, such as application licenses, data

gathering, and maintenance costs. So for SaaS, INaaS, or BPaaS, other price models are more appropriate.

Transaction-Based Price Models Transaction-based pricing uses transactions instead of computing resources as the basis for pricing. The transactions can be business-related, such as invoices processed for BPaaS, data-related for INaaS, or application-related for SaaS. You can also have transaction-based pricing with IaaS and PaaS, for example, by using the bandwidth as a proxy that indicates utilization of computing resources; in this way a consumption-based price model can be converted to a transaction-based model by assessing the bandwidth used in each transaction.

The cost of a transaction is calculated by dividing the cost of providing a cloud service by the estimated transaction volume over a given period. This is then the unit transaction price. This price model is suitable for use under the following circumstances:

- Transaction volumes are known and predictable.
- Your business process can be defined clearly and can be measured in discrete units to represent a transaction.
- The transaction volume is tied to your cost drivers.

From the cloud service provider's perspective, when business processes are standardized and driven by trans-

actions, using this price model is appropriate. Transaction-based pricing is most suitable for INaaS and BPaaS abstraction levels and is equally suited to all cloud deployment models.

Subscription-Based Price Models Like the all-you-can-eat model, the subscription price model relates to a price paid regularly, usually monthly, to use a service. For example, when you subscribe to a magazine, you pay a regular fee regardless of whether you read all of it, some of it, or none of it. With the onset of web-based magazines and news portals, the content is refreshed quite often, so the content is not a fixed amount as it is with a paper magazine. Paying a subscription for such a service approaches the all-you-can-eat model as your capacity to consume becomes less than the amount of new content being produced. Sometimes there is a contractual period over which you are bound to pay the subscription. In cloud computing, for instance, you may have a monthly fee for computing resources that are allocated to you, and you would pay the monthly amount regardless of whether you used those allocated resources. Also, you may have a notice period requiring you to inform the supplier—for example, three months beforehand—if you decide to stop using the service. Subscription pricing can be used well for all cloud deployment models and abstraction levels.

Service-Based Price Models

Service-based pricing uses the benefit delivered to you, such as the SLA realized, risk transfer, or money saved, as the criterion for defining the price you pay for the cloud service. Broadly, the fixed price model is a risk transference model whereas the other two models discussed—volume-based and tiered—largely provide money and service benefits to you as a cloud user.

Fixed Price Models The price that you pay for service in the fixed price model is fixed on a yearly, quarterly, or monthly basis. The fixed price is usually made up of two components, recurring and nonrecurring prices. The latter is a one-off amount that you pay at the outset, followed by recurring payments at regular intervals. The fixed price model is generally chosen when you have a clearly defined scope that is aligned with your short-term goals. Although this is used to transfer your risks related to delivery, people, and quality, you still own the risk of the service's scope by deciding how much of the service to use and to what extent. The risk transferral occurs through the SLAs that you define and agree on with the cloud service provider. Fixed pricing can be used well for all cloud deployment models and abstraction levels.

Volume-Based Price Models Volume can relate to, for example, the number of users, amount of storage space,

speed of transactions (denoted as number of transactions per minute or hour), amount of bandwidth, or processing power utilized. Any of these parameters or a combination of them can be used as the basis for deciding the price you pay for the cloud service. Because volume varies over time, business cycle, or events such as a marketing drive, the price can change. It is therefore imperative to define, calculate, and measure it. For instance, the price for a thin client computing service where your employees use cloud services on a volume-based pricing model could be calculated on the basis of average users, peak users, allocated users, or concurrent users per day, or a combination of these. Similar considerations would apply to other parameters, should they be used in the volume pricing instead. Although volume pricing is most often used in IaaS and PaaS, it is just as suitable for the other abstraction levels.

Tiered Price Models The tiered price model uses a tiered form of pricing that can be based on SLAs, volume, or amount spent. It is similar to the tiers that airlines have for their membership levels as determined by the amount you spend on travel with the airline. With cloud computing a similar form of tiered pricing may apply, with greater discounts available provided you spend a certain amount each year. Alternatively, you could have tiers based on the SLAs such that the more stringent the SLAs, the more you

pay. For example, there could be three SLA tiers and three corresponding price tiers, with each SLA tier providing greater benefits to you. Or you could have the tiers based on volumetrics such as the number of users being served. That is, suppose your business requests a cloud service provider to provide storage to your employees such that they can store documents and access them from anywhere and from any computer. The storage provider could have three tiers for its pricing: for serving less than a hundred users, the price might be $5 per user per month. If your company needs storage for between a hundred and a thousand users, then the price might be $4 per month per user, and for more than a thousand users, the price might be $3 per month per user. These bands, or tiers, that define the prices for you on the basis of volume represent a tiered price model. And the basis for creating the tiers can be volume, SLAs, or the amount spent. Tiered pricing can be used for all cloud deployment models and abstraction levels.

Performance-Based Price Models

Performance models are benchmark-based models that rely on key metrics, or benchmarks, to decide the price paid. Most performance models originate from employee remuneration- or outsourcing-related price strategies but can be applied to cloud computing, especially to a private or hybrid cloud service. Sometimes these models are used

to align your business goals with those of your service provider's goals to create a true partnership.

Performance price models exhibit some common traits:

- They require a clearly defined output or metric that can be measured easily.
- The metric is often aligned with a business process or outcome that has a demonstrable relationship to its impact.

We consider below the outcome-based, business-linked, and gain-share price models as representative performance models.

Outcome-Based Price Models If your department wants to use cloud computing because it wants to reduce time to market, then you may want to negotiate a "bonus" payment to the cloud service provider that is linked to that outcome. Most outcomes use metrics that relate to cloud computing's value proposition, as expressed by the value models that we consider in the next section. There is a difference in psychology between the outcome-based model and some performance-related price models. With the former, you provide a bonus if an outcome is achieved, and with the latter, you penalize the

provider if an SLA or benefit is not realized. Outcome-based models are often used with other models, usually fixed price models, to create a value culture based on rewards.

Business-Linked Price Models Whereas outcome-based models use metrics that measure the value of cloud computing, business-linked models measure the contribution that cloud computing makes to the key performance indicators (KPIs) that affect your business model. One of the challenges is linking the business outcome to the contribution made by cloud computing. Figure 13 shows a possible mapping between the objectives for using cloud computing and the related business outcomes as expressed by business KPIs.

Gain-Share Price Models The gain-share model has its roots in employees' remuneration schemes. The idea is that as the organization gains, it shares some of those gains with its employees. A typical gain-sharing organization measures its own performance and shares the profits with all its employees using a predetermined formula. The organization's actual performance is compared to its historical average (known as its standard or baseline performance) to determine the amount of the gain. In a cloud computing context, instead of having penalties should certain SLAs not be met, you reward the service

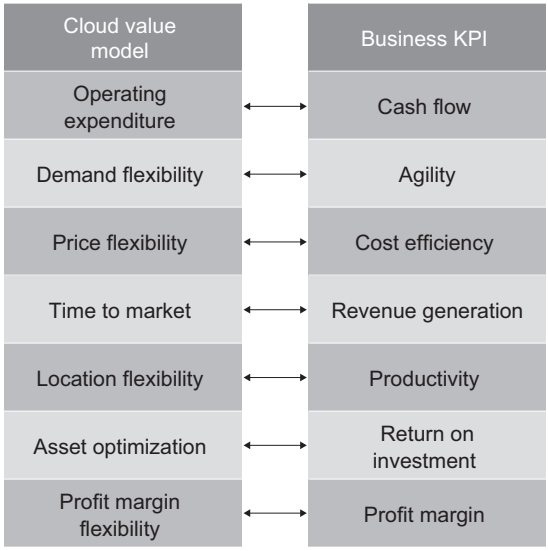


Figure 13 Mapping cloud computing objectives to business and financial KPIs.

provider by sharing your profits if the SLAs are exceeded. It is a different approach psychologically. However, you can combine the gain-share model with a penalty-based performance model to create a hybrid performance model.

Marketing-Based Price Models

Certain price models are driven by marketing rather than performance. The key driver behind such models is to attract as much custom as possible and then to monetize it

to create a profit. We discuss two such marketing led price models in this section.

Freemium Price Models There are two types of freemium. In one type, you try before you buy a more enhanced service; in the other type you get a free service, but the advertisements provided to you make up for the service's price. This model is especially suited to SaaS because many software companies such as LinkedIn and Dropbox use it well. They offer a free version of their product that has limited functionality but they also provide the option to pay for a premium service with extra features. The idea is to offer enough value to users in the free version to attract and retain them, and more value in the enhanced version to ensure that the users convert and maximize the service provider's revenue.

Razor-and-Blades Price Models This pricing model relies on two components, a base component and a reusable component. The reusable component is utilized by the base component to deliver a service. It is akin to selling you razors cheaply, or even giving them away for free, and then making up for it from the prices of the consumable blades. Printers are another example. They are sold cheaply, but the price is made up from the printer ink supplies.

In cloud computing, a device or an app that uses a cloud service may be given away, but the price may be

A device or an app that uses a cloud service may be given away, but the price may be made up from the data that are stored, analyzed, and presented by the cloud service.

made up from the data that are stored, analyzed, and presented by the cloud service. For example, you could have a blood pressure monitor that sends data automatically to a cloud service. The cloud service would then store and analyze the data, which it would use to alert you if a certain blood pressure level were traversed. The sensor could be provided for free or at a reduced price, whereas you would pay for the use of the cloud service that made the sensor information meaningful to you. Another example of this is Amazon's Kindle, which can act as a window to a virtual storefront from which you may purchase a wide variety of books. The Kindle device is sold at a discounted rate and is called a loss leader, but its value is made up from the increased sales revenue in the storefront that result from its use.

Hybrid Price Models

The utility-, service-, and performance-based price models discussed above are not mutually exclusive; they can be combined to produce hybrid price models. For instance, you could have a subscription-based system that utilizes a tiered approach. If the dollar spend per annum were to be at a certain level, then that level would decide the discount tier that would apply to you. Another approach would be to combine the risk transfer of the fixed model with the affordability of one of the utility price models to provide a

fixed monthly price to users such that they may consume as much of the service as they wish for that fixed monthly fee. This type of hybrid price model is quite common for many public cloud services such as Google docs and Microsoft 365. In fact, it is a good pricing model for large or long-term services, especially if they need to be perfected over time. The hybrid price model can be applied successfully to all the cloud abstraction levels and deployment models.

Financial Metrics

How do you translate the value that cloud computing can provide into a meaningful financial metric? There are various financial yardsticks that can be used to assess the value proposition of a cloud service. Let us consider four common financial metrics: payback method, net present value, return on investment, and time to market. There are other metrics that can be used as well, such as economic value added, return on assets, and return on equity. These latter metrics, however, are difficult to use when considering a single service, product, or project because they typically aggregate the computations at a corporate level and so rely on other factors such as the company's tax rate and its corporate KPIs. For this reason we will not consider them further.

Payback Method

The payback method measures the time needed to recoup your investment in a product or service. A service that has a shorter payback period is deemed to be better than one that has a longer period, as in the following example:

- Suppose you purchase a cloud service at $1,000 a month so that you can process invoices twice as fast as you did using your older system.
- Over a year, the cost for the service comes to $12,000.
- Suppose that the old system processed $10,000 worth of invoices a month and the newer system processes $20,000 worth of invoices over a month.
- The value obtained is the difference between the old system and new system, which is $10,000 per month because the new system is twice as fast.
- Per day, the value amounts to $333 worth of invoices, on the assumption that a month has thirty days.
- This means that the new cloud service will pay for itself after thirty-six (12,000 divided by 333) days.
- Thus the payback period is thirty-six days.

Usually, the payback method is better suited for capital expenditures because you can depreciate it over

several years. Hence the one year used in our example to arrive at the payback time for an investment of $12,000 would need to be changed to encompass the years over which you can depreciate capital items. For operating expenditures, you would probably have a lock-in period or a contractual period with your cloud service provider, and it is this that would substitute the one year used in our calculation.

One of the shortcomings of the payback method is that it does not consider the time value of money, which can have a substantial bearing on the investment calculation during periods of high interest rates or over long periods of time. To account for this shortcoming, net present value (NPV) calculations are usually used.

Return on Investment

Whereas the payback method considers the time to recoup the investment, return on investment (ROI) uses the percentage of the investment amount that will be recouped. ROI is widely used in the IT industry to assess capital investments. The formula for ROI is

ROI = (Gain from investment – Cost of investment) / (Cost of investment).

For our example of a new cloud service at $1,000 per month, the ROI would be computed as follows:

- The gain from the investment is $10,000 worth of extra invoices a month.
- The cost of the investment is $1,000 per month.
- Thus ROI = ($10,000 – $1,000)/$1,000 = 900%.

Unadjusted ROI, as calculated in our example above, assumes the present value for all gains and costs; thus the assumption is that all gains and costs are produced at the outset, which is not often the case. To adjust for this anomaly, usually an NPV calculation is performed that discounts the time value of money to get a more realistic value for ROI.

Net Present Value

When you have multiple cash flows occurring on a regular basis, those cash flows are called a stream of cash flows. When regular streams of cash flows are equal in value, the cash flows are known as annuities. Because of the effect of inflation and interest rates over time, a cash flow amount of $1,000 next month is worth more than the same amount paid to you fifty months hence. To assess the present value to you of a series of cash flows, we use NPV calculations. The NPV of an investment is the present value of all future benefits, such as cash flows, generated by the investment, net of initial costs, discounted over intervals of time. For example, you are to receive $1,000

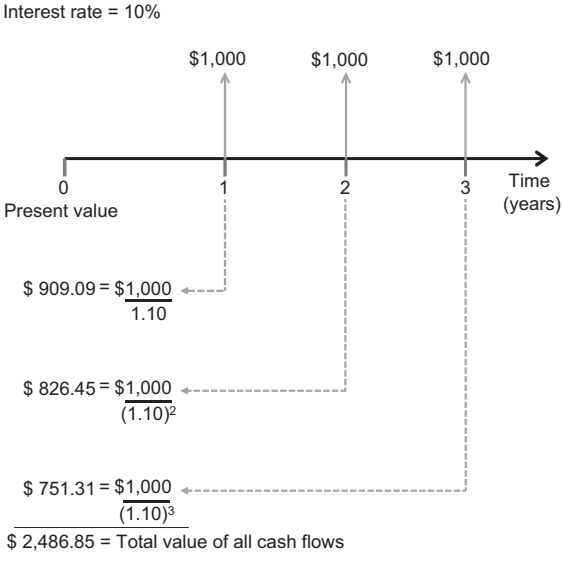


Figure 14 Calculating NPV using cash flows.

every year for three years and the interest rate is 10 percent. You would discount those cash flows with the interest rate shown in figure 14, and then add the discounted values to obtain an NPV of $2,486.85. If you had made an initial investment of $2,000 to obtain those cash flows, then you would subtract that initial investment from the discounted cash flows to obtain an NPV of $486.85, and this would then represent a profit of 24.3 percent for your investment of $2,000.

Although NPV analysis is frequently used to justify capital expenditures, you can use it to perform a reverse calculation to come up with recurring expenditures as represented by that capital expenditure. Thus NPV provides a means for converting CapEx to OpEx, and vice versa. The benefits of using NPV analysis are the relative precision of the results owing to the use of time value of money and the simplicity in the interpretation of its results: a positive NPV indicates a profitable investment. Another benefit of using NPV is that opportunity costs are accounted for implicitly because of the use of a discount rate. Thus, if a projected rate of return is less than your hurdle rate, or your desired rate of return, then you would not make the investment. And of those candidate projects that do pass your hurdle rate, the one with the highest rate of return should provide you with the optimum opportunity cost.

Returning to our invoice processing example, which had a cloud computing cost of $1,000 per month, let us suppose that the interest rate is 5 percent per annum, and we decide to make use of the service for at least three years. Those monthly outflows, as costs, can be represented as a capital expenditure using NPV analysis:

$r = 5\% \div 12 = 0.4167\%$ (converted to a monthly rate of interest)

$N = 36$ months

A = $1,000 (monthly amount)

FV = 0

$$\text{PV} = \sum_{k=0}^{N} \left(\frac{A}{(1+r)^k} \right) = \$\ 33{,}365.70.$$

The present value of using a cloud service by paying $1,000 per month for its use computes to $33,365 over three years. You then need to compare this amount with the amount that your IT department provides you with for creating your own computing platform. The lower value wins your investment time and money.

Time to Market

Estimating the time when revenues will be obtained is another financial yardstick that can be used. For example, let us assume that using traditional computing, it would take you a year to go to market with a new offering that you are developing. With cloud computing, however, it could take you three months instead. Thus, the time to market (TTM) will be nine months less. And if you expect to earn $20,000 per month with the new system, then the earlier TTM would represent additional inflows of $180,000. You can therefore express TTM not only in terms of time but also as its equivalent in monetary terms.

8

DATA

In this chapter we consider data security in its entirety. This account includes data integrity and privacy from an end-to-end perspective: from the user to the data center and back to the user through the network. We achieve such an end-to-end perspective by discussing the entire data journey that incorporates data encryption, certificates, and checksums. We also consider legal and compliance issues related to data and its use. We extend this discussion by considering data sovereignty and jurisdiction issues since these are of special concern to multinational cloud users. We conclude the chapter with a brief discussion of backing up and restoring data.

Data Integrity

Suppose you place a letter that contains some important information in a safe, which you then bury somewhere and do not let anyone know about it, and then throw away the key in the ocean. That, in information security (or IT security) terms, is obscurity, not security. In other words, the information in the safe is of no use to anyone, although it is extremely secure. On the other hand, if you were to place the letter in a safe document box and send it to the recipient so that only she could read the letter, then you would be sending the information in a secure manner. Now, if the safe were to be intercepted, then three scenarios may arise: (1) the interceptor has possession or control of the letter and can withhold it from you or the recipient; (2) the interceptor can read the letter and use its contents to further her own purpose (e.g., impersonate you by stealing your identity or hack into your computer systems); or (3) the interceptor can replace the letter with another document and send it to the recipient so that false information is received. Data integrity aims to secure the communication channel, or the data sent across it, to ensure that none of these three scenarios occurs.

For maximum data integrity, three elements are considered in securing the flow of data from one person (or user) to another. The end-points need to be secure; that

is, the sender and recipient of the data need to be authenticated so that you know they are the ones for whom the data are meant. The channel over which the information is sent needs to be secure so that other parties cannot easily eavesdrop and false data cannot be sent instead by others (known as a man-in-the-middle attack); and the data need to be encrypted so that the data cannot be freely read by a third party. (These three elements are not mutually exclusive, and you will often find a combination of the three elements used to ensure data integrity.) The next chapter, which considers security, covers the first of these elements, securing the end-points. In this chapter we consider how to ensure data integrity using encryption, how to use checksums to verify your data's integrity, and data loss prevention.

Encryption

To secure the network connection (we will refer to the connection as a channel through which data pass) that you use when transmitting data, you need to encrypt it. The data that are sent through that channel can be encrypted as well, although often this is negligently not done. Channels are encrypted by means of certificates. These allow the sender and receiver to communicate over a trusted channel. Such an encrypted, trusted channel is known as a secure socket layer (SSL) connection or a transport layer security (TLS) connection; TLS is the modern successor to

SSL. When two parties need to communicate over a secure channel, the following handshake protocol is used:

- The sending device uses public-private encryption keys to encrypt the channel.
- Best practice is to use valid certificates (these contain the public key) signed by a trusted certificate authority to jump-start the process.
- Both parties agree on the cryptographic protocols to be used.
- The parties negotiate a shared secret (the private key) to use.
- The receiving device uses the public-private encryption keys to decrypt the channel.

Once the handshake is completed between peers, both parties start to communicate in a secure fashion using the negotiated encryption algorithm and cryptographic keys. This secure channel protects them against any eavesdropping or man-in-the-middle attacks.

The concept for data encryption (i.e., securing the data that passes through the channel) is similar except that the data are encrypted by the cloud service and decrypted by your application or vice versa, depending on the direction of data flow. So the applications, instead of the web server or web browser, perform the data encryption.

Checksums

To establish the veracity of data you receive, certain algorithms are used to compute checksums on the data. The data and the checksum then are corroborated by the receiver.

Let us suppose, for example, that you send me a letter. Separately, you could also send me some information, such as "The letter has 251 words, of which there are 105 nouns, 81 adjectives and the rest are pronouns or verbs; it also contains 45 sentences and 5 paragraphs." This information describing the letter you sent me is akin to a checksum. A checksum is used to determine that the data sent are received as is. Various algorithms are used to compute checksums automatically, on the fly, so that the receiving party can ensure the accuracy of the information that has been sent. Examples of such algorithms include CRC-32, which is a 32-bit cyclic redundancy check that enables the detection and correction of errors. Another common algorithm is the message digest (MD5) algorithm.

Whereas checksums are used to compare and validate the data received, cryptographic hash functions are used to verify the data. A basic requirement of a cryptographic hash function is that it should be infeasible to find two distinct messages that have the same hash value. Such a function commonly acts as a one-way function that maps data to a fixed length. It is often used to compare large sets of data without the need to send the data over the

To establish the veracity of data you receive, certain algorithms are used to compute checksums on the data. The data and the checksum then are corroborated by the receiver.

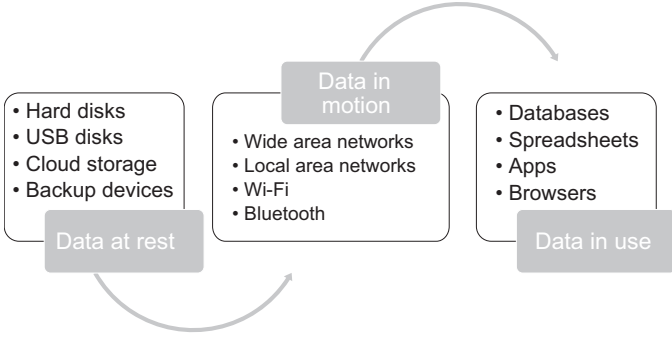


Figure 15 Three states of data.

network for comparison, and so to detect any tempering of the data when received. Examples of such functions are SHA-3 and BLAKE3.

Data Loss Prevention

Data loss prevention encompasses numerous functions: the discovery, identification, monitoring, management, and protection of data that are in use, in motion, or at rest, wherever the data are stored and used. Usually, data loss prevention is used for confidential or private data. Figure 15 shows the three states of data that define the scope for data loss prevention, together with common examples of places where data in each state can be found. These three states encompass the entire life cycle of the data: data creation, transmission, usage, storage, archiving, and final destruction. Thus, data at rest are inactive or semi-active

data that are stored in any digital form; data in motion are fluid data that are being transported from one endpoint to another; and data in use are active data that are in constant use.

As a user of cloud services, you should ensure that data loss is prevented during the entire life cycle of the data by employing tools such as encryption (to secure the data), checksums and cryptographic hash functions (to verify the data's integrity), tagging and monitoring (to know who has accessed what data), and effective data management (to ensure that stale or unused data are destroyed within time-bound limits.)

Data Privacy

We regard privacy as applicable to information that identifies a person. This is generally known as personally identifiable data (PID) or personally identifiable information (PII). The NIST defines PII as:

> *any information about an individual that includes:*
>
> *(1) any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records,*

(2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information.[1]

Most countries have data protection laws in place to ensure that such data do not fall into the hands of unauthorized parties. Most such laws include data retention directives, for example: "providers of services should erase or anonymize the traffic data processed when no longer needed." This is an example of a good directive that most cloud service providers ought to comply with. However, the same law has some directives that are just impractical to follow—especially those related to cookies—to the extent that no website today complies, or attempts to comply, with such directives. As a result, data protection laws have been discredited somewhat and have not been treated as seriously as is warranted by many service providers.

Data privacy is closely tied to data integrity. To ensure that your data remain private, you need to ensure that data retention and transmission are implemented with data integrity tools. Security is usually enforced with intrusion detection, intrusion prevention, and the use of firewalls, antivirus software, and antimalware tools. But if these defenses are breached, as they so often are, then your private data would be readily available if you did not encrypt it. You can encrypt data that have been stored on your hard drive, on disks, or in a cloud; this is known as

encrypting data at rest. Thus encrypted data that have been obtained as a result of a security breach would then not be accessible to an unauthorized user. In addition to data-at-rest encryption, you can encrypt data that you send to others via email, file-sharing products such as Dropbox, or to your own cloud storage. This encryption protects data during transit, and it ensures that any interception of the data while they are being transmitted from one place to another would prevent the data from being read by an interceptor.

Please bear in mind that encryption protects your data from mostly nongovernmental actors. Some governments can and do ensure that backdoors are in place at various strategic locations to ensure that they can access your data readily. Examples of strategic locations include the firmware of devices that (1) route the data (e.g., routers, VOIP servers, and modems), (2) store the data (e.g., hard disks), (3) provide data security (e.g., firewalls and intrusion detection tools), and (4) encryption algorithms.[2] These backdoors are usually implanted in the firmware of the devices so that no user or software is aware of their existence, let alone be able to guard against them. With the onset of the internet of things, the ubiquity of backdoors should increase markedly since the "things" that are connected to the internet could have such backdoors in their firmware. However, at present, only a select few governments (perhaps fewer than five) have this capability. One

Data privacy is closely tied to data integrity. To ensure that your data remain private, you need to ensure that data retention and transmission are implemented with data integrity tools.

way to ensure that your IoT device has a relatively small chance of being compromised by such backdoors is to insist that its firmware be open source. Of course, this relies on software developers and engineers to have noticed the existence of such backdoors should they have examined the source code of the firmware.

Data Jurisdiction and Sovereignty

Cloud computing, like the legal system, is a tool that ought to be used to help the good and protect them from the bad. Unfortunately, in real life, complexities arise, and things do not always work out that way. One of the reasons is that technology (cloud computing, being one of its latest innovations, is a good example of this) changes very fast in comparison to laws and the legal framework. This gives an opportunity for early adopters to interpret or bypass the spirit of the law if a void exists in the letter of the law. To some extent, this can be for the greater good as it does not stifle innovation. On the other hand, security breaches and exploitation of users result from such a situation. A balance therefore needs to be struck. A large part of the solution is to educate both users and the legal community of the information that can be collected, analyzed, and used as a result of the progress in technology. Perhaps greater cooperation and standardization across industry

bodies and governments are also needed. An example is MiFID (Markets in Financial Instruments Directive), which is an EU law that regulates investment services to increase competition and consumer protection. The main challenge with regulations, however, is jurisdiction. What regulation or law should a cloud service conform to if it is provided by a company established in country A, its users are principally in country B, its data are stored in country C, and the cloud service itself is hosted in country D? In such cases you should take care to understand the legal and regulatory jurisdiction of a cloud service before buying and using it. In particular, you need to be aware of who would see the data, what systems would handle it, and which applications would use it, then establish the legal jurisdiction of those people, infrastructure, and applications.

Tied closely to legal jurisdiction is the jurisdiction of ownership of the data. If several people or organizations have been involved in creating data, or if the data are consumed by several parties, then who owns the data? If the data are manipulated or enriched by intermediate systems or cloud services, who would own the data then? Would it be the original creator of the data, the intermediaries that augmented the data, or the end user? What if all those actors are in different countries, each with its own (different) data protection and ownership laws? These are some of the quandaries that arise where data sovereignty

is concerned, and they relate acutely to cloud computing because of its ubiquitous access. If you have the capacity or wherewithal to do so as a business user, you should ensure that you have a data ownership agreement with your cloud service provider that additionally considers the legal jurisdiction for redress in the event the agreement is breached.

Migrating Data to the Cloud

Data that you want to migrate to the cloud from an on-premises location will either be stored in a database or on a storage device such as a hard disk. To migrate data to cloud storage from on-premises storage, you essentially compress the data using a compression algorithm or a tool that implements the algorithm (such as gzip) and then transfer the compressed data to cloud storage using a secure copy (using the scp command) or by way of secure file transfer (using sftp). For database migration, there are two avenues: (1) back up data from the local database and restore to the cloud database, or (2) use an ETL (extract-transform-load) tool to extract the data from the local database and load them to the cloud database. The former is a straightforward and efficient option if the databases are similar; the latter gives you the opportunity to transform and manipulate the data before transferring them to the target database.

For migrating data from the cloud to local infrastructure, you would use essentially the same process described above but in reverse.

Backup and Recovery

In life as well as in IT, Murphy's law prevails.[3] To recover from disasters such as loss of data, it is advisable to keep backup copies. Data, in the broadest sense, can mean your personal data, the source code of apps, your financial data, and even your electronic books. We consider some of the nuances of backup copies by considering the differences between backing up and archiving. This is followed by a discussion of various types of backups, and we end by considering different backup strategies.

Whenever you implement or create a backup capability, always ensure you have a concomitant recovery capability in place. Without a recovery capability, all your backups will be useless. In this regard, you can use metrics that are known as the recovery time objective (RTO) and the recovery point objective (RPO). Both metrics are expressed in terms of time, as hours or minutes. RPO is the amount of data that you can lose without impairing your business capability; it is measured as the time from the critical point—such as a disaster or failure—to your most recent backup. RTO is the time that your system (in our

case, the system includes the cloud service) can be down without causing significant damage to your business; this includes the time spent restoring the system or service and its data.

The Difference between Backing Up and Archiving

Backing up is meant for the rapid recovery of day-to-day data, or operational data, that are in current use. Archiving is meant to store data that are not used regularly but that you wish to keep for regulatory or compliance reasons. For backups, speed of recovery is important as you would like to be up and running quickly should disaster strike, whereas for archiving, the capability to perform fast searches to locate information that is required is far more important. As you might surmise from these differences, backing up is performed regularly on data that change often. However, archival data do not change often, and your archiving schedules can be less frequent than backup ones. Hence your backup sets will have longevity measured in weeks or months while your archive sets will have retention periods measured in years and decades.

Types of Backups

We define a fileset (connotes what we are backing up) and then consider the three common types of backup schemes: full, differential, and incremental.

Filesets Usually, in storage file systems, a file hierarchy exists. This hierarchy consists of a series of directories that form a treelike structure. Each directory contains other directories, files, or other file-system objects such as links to files or directories. A fileset provides a means of partitioning the file system at a finer granularity so that you may select files for backup purposes and set read/write permissions individually. Hence your backup fileset is a set of files and directories that you have selected to back up.

Full Backups A full backup will back up your entire fileset every time you perform the backup. The advantages of a full backup are (1) all the files and directories are backed up to one backup set, which makes it easy for you to locate a particular file, and (2) files and directories are easily restored from a single backup set should you need to restore them. Its disadvantages are (1) it is more time-consuming than other backup schemes, and (2) full backups require more space when compared to other backup schemes.

Differential Backups A differential backup backs up only those files that have changed since the last full backup was performed. A full backup and its differential backup should therefore include all the files (changed and unchanged) in your fileset. The advantages of differential backups are (1) they require less space than incremental backups, and (2) backup times are generally faster when compared to

full and incremental backups. Their disadvantages are (1) restoring all your files may take considerably longer than with a full backup since you may need to restore both the last differential and full backup, and (2) restoring individual files or directories may take longer since you have to locate them on either the differential or full backup sets.

Incremental Backups An incremental backup provides a backup of files that have changed or are new since the last incremental backup. The first incremental backup performs a full backup as it backs up all the files in the fileset; subsequent incremental backups back up only those files that have changed since the previous backup. Its advantages are (1) backup time is faster compared to full backups, (2) it requires less storage space than other backup schemes, and (3) you can keep several versions of the same files on different backup sets. Its disadvantage is that in order to restore all the files, you must have all previous incremental backups available, and so it may take longer to restore a specific file or directory since you must search more than one backup set to find the latest version.

Backup Strategies

Backup strategies or rotation schemes originated when backup media were expensive and wore out owing to reuse when tapes were used to perform backups. These days it is far cheaper and easier to configure a disk array to store

backups and to monitor the disks for failure. In fact, this is the method that works well for cloud storage and backup service providers. However, you can utilize the concept governing rotation schemes by substituting tapes with cloud service providers so that you have a backup scheme that does not rely on a single cloud service provider.

The main purpose of a backup rotation scheme initially was to minimize the amount of storage media used for backup data. Such a scheme defines how and when a backup job is run and the period over which the data are retained. *For our purposes, however, the idea is to minimize the risk of dependency on a single location or cloud backup provider rather than to minimize backup media use.* The most common rotation schemes are FIFO (first in, first out), grandfather-father-son, and the tower of Hanoi. We consider the first two below in the context of cloud-based backups.

FIFO Backup Scheme When performed using a weekly rotation, for example, you would make a full backup every day using seven tapes, one fresh tape per day. Then, on the eighth day, you reuse the tape you used on the first day, and on the ninth day, you reuse the second day's tape, and so on. In a cloud context, let us assume that you appoint two different cloud backup providers, denoted provider 1 and provider 2. If you wanted to follow a full backup scheme, then you would back up to each provider

on alternate days. So, on the first day, you would use provider 1 for a full backup, and on the second day, you would use provider 2 for a full backup. However, for a differential backup, you would perform a full backup with provider 1 on the first day of a week, and then every day until the end of the week you would perform a differential backup. In the second week, you would use provider 2 in a similar fashion, thus alternating between provider 1 and provider 2 on a weekly basis.

Grandfather-Father-Son Backup Scheme The grandfather-father-son rotation scheme is a monthly-weekly-daily rotation scheme. With this scheme you would appoint three backup cloud providers such that the "grandfather" backup cloud kept your monthly backups, the "father" kept your weekly ones, and the "son" kept your daily backup sets.

SECURITY

Security is holistic. As such, our scope extends to the following six perspectives to provide a comprehensive, end-to-end understanding of implementing and assuring security:

1. The cloud service: We consider the shared responsibility model that is adopted by most public clouds.

2. Your connection to the cloud: This is where networking and network access to the resources (access credentials, applications, data, and business logic) are secured and managed.

3. The users using and administering the cloud service: Identity and access management (IAM) plays a major role in ensuring that the right users and applications are

granted access to the cloud resources with appropriate privileges.

4. The applications associated with the cloud service: The applications and their end-points, such as APIs, are considered in this regard.

5. The devices that connect to and use the cloud service: Host-based access management is quite important in ensuring that only applications or users access the cloud resources by using allowed devices.

6. The data that are used by the applications hosted in the cloud: Data integrity and encryption protocols are used so that only privileged users and applications have access to data and additionally to ensure that the data have not been compromised.

In addition to the technical resources in the cloud, security covers physical access to the computing devices, the divulgence of your passwords to others (either knowingly or unknowingly), and, of course, the enforcement of a security policy. Also, it is of the utmost importance that the environment be monitored for any security breaches and that appropriate action be taken. We cover all these topics in this chapter apart from data security, which was covered in the previous chapter.

You may want to refer to the appendix for a brief glossary of commonly used terms related to security.

Shared Responsibility Model

Because of its holistic nature, security is not just the cloud service provider's concern; it is yours too. Security, then, is a responsibility shared by you and the cloud service provider. The latter is normally responsible for security in the infrastructure up through the interface point between your application and its hosting environment in the cloud. You, however, are responsible for security with respect to interfacing with the cloud environment, and especially within the application that uses that cloud environment. Most public cloud providers use this shared responsibility model for security. Figure 16 shows how the model applies for the various abstraction levels.

Cloud-native applications generally come under the PaaS or SaaS category in figure 16, depending on the type of design and deployment used.

Let us consider, as an example, an application that you might want to install on a virtual machine (VM) at Amazon Web Services (AWS). Since it is an application that you are installing, you will be using AWS as infrastructure (i.e., EC2, AWS's reference to a VM) to host your application. A look at the IaaS column in figure 16 shows that the application's security is your responsibility but the physical host's responsibility will be with Amazon.

Often, the weakest link in any secure system is the human being. It is up to us, the users of technology, to ensure

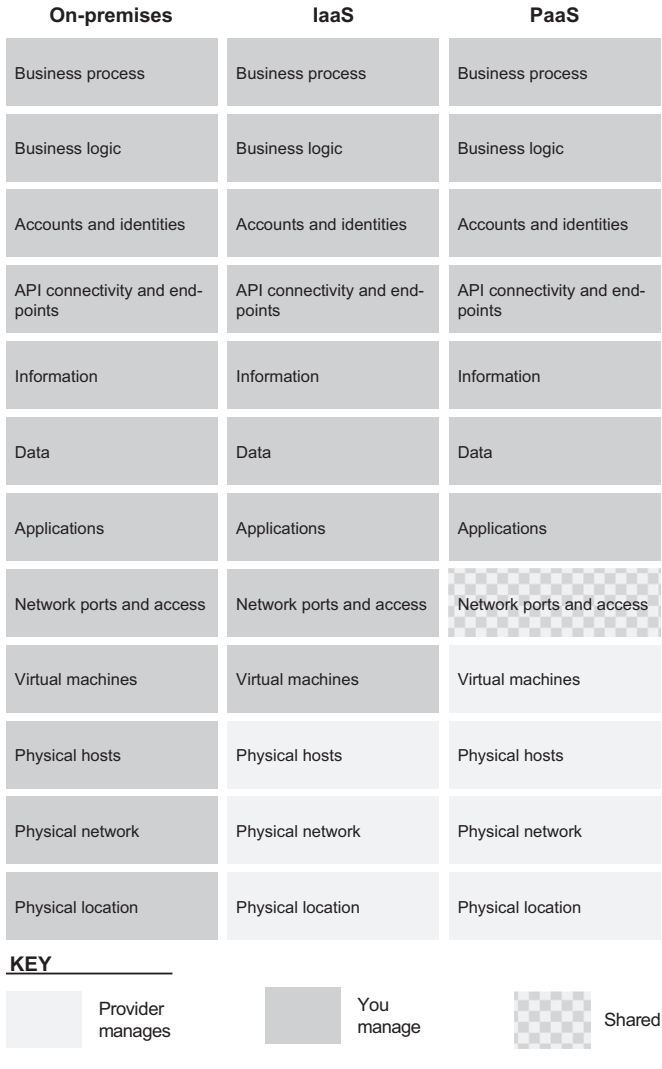


Figure 16 Shared responsibility model and abstraction levels.

SaaS	INaaS	BPaaS
Business process	Business process	Business process
Business logic	Business logic	Business logic
Accounts and identities	Accounts and identities	Accounts and identities
API connectivity and end-points	API connectivity and end-points	API connectivity and end-points
Information	Information	Information
Data	Data	Data
Applications	Applications	Applications
Network ports and access	Network ports and access	Network ports and access
Virtual machines	Virtual machines	Virtual machines
Physical hosts	Physical hosts	Physical hosts
Physical network	Physical network	Physical network
Physical location	Physical location	Physical location

Figure 16 (continued)

that our computing systems remain secure by taking certain measures:

1. guarding our passwords,

2. ensuring that antimalware (includes antivirus and antiphishing software, cookie management protocols, etc.) is in place and is updated regularly,

3. ensuring intrusion detection and prevention software on applications and router firewalls is running, and

4. generally, not opening emails or visiting websites that look suspicious to prevent phishing initiatives.

These measures should ensure that the devices that you use to access your cloud services remain secure and are not compromised. Further, you need to ensure that any data you transmit to your cloud are encrypted so that the data may not be read even if intercepted by a third party. There is very little technology can do if you, the user of technology, do not use it in a secure manner!

Key Security Components

This section attempts to convey the mindset you need to have to secure your cloud assets and considers the various

Often, the weakest link in any secure system is the human being. It is up to us, the users of technology, to ensure that our computing systems remain secure by taking certain measures.

components at your disposal for creating a security architecture that is secure and at the same usable. But first, a definition of security:

> The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.[1]

Notice that the definition uses the terms **c**onfidentiality, **i**ntegrity, and **a**vailability; these three are known as the CIA triad.

Confidentiality defines who can get what kind of information. For example, companies would be concerned about protecting their intellectual property, while individuals would tend to be more concerned about unauthorized access to their financial or medical records.

Integrity refers to the information being correct or consistent with its intended use. Any unauthorized modification of data, whether deliberate or accidental, is a breach of data integrity.

Availability refers to having access to data or information in a timely manner when it is required for its intended use. Availability has multiple aspects:

1. Will a service be available from any other location?
2. How readily will it be available when you want to use it?

3. Will there be certain times when it may not be available because of planned maintenance?

4. If the cloud platforms hosting the service were to go down, how long would it take to recover?

5. Once recovery is complete, will I have lost any data, and if so, how many hours' worth of data will it be?

The last two items are measured in metrics that are known as the recovery time objective (RTO) and the recovery point objective (RPO), respectively. Both metrics are expressed in terms of hours or minutes. Item 3 relates to downtime. It is usually expressed as a percentage that is calculated in the following manner.

% Availability = 100 * (Number of minutes of downtime over a period) / (Overall minutes in the period)

The definition of security above also raises several questions: Who decides what is authorized, or nonmalicious, use? How does information get categorized as requiring more or less secure use? Who or what uses the information, and why do they use it? What systems need to process the information or data? Where and how is the information stored and used by the systems? These are some of the questions that a security policy for the cloud would need to address. A security policy is very

important to have regardless of the amount of cloud computing you or your organization perform because it provides the foundation of the security implementation. The security policy should cover the various touchpoints discussed below.

Security Touchpoints

For our purposes, a cloud service user can be a person, a process, or an application, or a system such as a computing device. Support staff, when called to help resolve an issue, are also classified as a user. If we were to follow the data trail, the data would originate from a *user*, pass over a *network* to reach a *computing system* that hosts *software*, which processes the data and stores it in a *storage device*, such as a solid-state disk, and those data get backed up onto a *backup device* and finally get archived in an *archival system* after some time has elapsed. The elements that touch the data, given in italics above, are:

- user,
- computing system,
- software,
- storage device,

- backup device, and
- archival system.

It is important to note that this entire data journey needs to be secure from the perspective of the user. The user's need for privacy (i.e., confidentiality), data integrity, and availability delineates these security characteristics. We therefore look at security in terms of this data journey, with special emphasis on the zero-trust approach.

Zero-Trust Model

The goal of zero trust is to eliminate trust from the system; it assumes that the network has been breached and so verifies each request for information or access as though it had originated from an unsecure, untrusted, or open network.

The zero-trust model is fine when it comes to how we enforce security; but what happens to the data flow or user once that "always verify" line has been passed? We need to ensure that the data or user is sandboxed within a zone of trust. All those elements within the cloud computing environment that share the same security characteristics for a particular user can have a common security boundary. The computing elements that reside within that boundary trust each other.

Such boundaries are created by segmenting the network into subnetworks that share the same security attributes. This segmentation usually takes place at the network layer since the network is the mode of transport for information. Thus virtual networks are created that exist on a physical network and each virtual network has rules to ensure authorized access to its internal data in conformance with zero trust. To further enhance security for each virtual network, you could incorporate three security procedures: identification, authentication, and authorization.

Identification entails establishing the identity of the user. In this regard, the user may be a person, application, system, or "thing" that wishes to connect to the cloud or its security container.

Authentication entails establishing the veracity of the user's identity. Usually, authentication is based on what you have or what you know. An example of the latter is a password or a PIN (personal identification number), while an example of the former would be a smart card, fingerprint, or voice biometrics.[2] Authentication takes place once a user's identity has been established.

Authorization establishes the tasks that an authenticated user can perform. The rights given to an authenticated user depend on the role that the user has within the cloud computing environment or the security container. Authorization always takes place after a user has been authenticated.

In summary, identification asks "who are you," authentication asks "are you who you say you are," and authorization asks "what are you allowed to do within the cloud computing environment."

User Security: Identity and Access Management

Identity and access management (IAM) refers to the function of identifying a user and granting access to cloud resources based on the user's role. It is important to understand that from a cloud provider's perspective, IAM extends to the infrastructure in an IaaS environment in accordance with the shared responsibility model shown in figure 16. The general principle is to first delineate the relevant stakeholders (or principals, in AWS parlance) and assign roles to them in terms of what they will be doing and therefore be allowed to access in the cloud. You then assign permissions to those roles using policies. This, in general, is known as role-based access control, or RBAC, which allows granular access rights to cloud resources. You can group related users or roles into user groups and define group policies should you wish to implement a less granular approach or to simplify the policies for an entire team in your organization. With Azure, IAM is implemented using Active Directory where you define tasks for each task owner (or stakeholder), and this can be

optionally synchronized to your on-premise Active Directory instance.

Data Security

You need to classify, label, and encrypt data to provide data-driven protection; this ensures confidentiality because the relevant users receive data that have the appropriate labels. To ensure the integrity of the data, use checksums and cryptographic hash functions to verify the data and further warrant that they have not been tampered with, as discussed in the previous chapter.

Application Security

You should limit access to only the applications, services, and cloud resources required to perform the function; therefore, you would create security containers that relate to the function and limit access to that security container. The zero-trust approach is intended to ensure that you provide adaptive access to the applications without requiring human intervention during normal operating conditions.[3] To operate adaptive access, a decision-making engine is needed to apply the organization's security policy continuously based on the contextual information it receives.

When determining access privileges, an important contextual factor is the risk score of the device or end-point requiring use of the application. The risk score determines the health of the end-point or device using the application. Mobile devices, for instance, have risk scores based on information related to the device itself, its apps, any other content it is accessing, and the network it is using.

Network Security

The key components of a network security regime are:

1. firewalls,
2. intrusion detection and prevention,
3. network segmentation,
4. virtual private networks, and
5. certificates.

Firewalls are devices that analyze the network data to block malicious traffic. Every computing node connected to a network has an IP address. The firewall can block traffic originating from a range of IP addresses and it can also allow traffic on certain ports. Consider the analogy of a house: its location is akin to an IP address and its external

doors are akin to ports. Ports are numbered and relate to the network transport protocol (TCP or UDP) used.[4] For instance, websites use TCP on port 80 that is delivered over IP to serve web pages and computer clocks use UDP port 123 for time synchronization via NTP.[5] Thus a firewall blocks certain ports and allows traffic to or from others in accordance with the rules you configure the firewall with. You can have virtual firewalls that reside in the cloud and so provide the security to cloud applications and services.

As an example of a virtual firewall, AWS provides a Security Group that controls traffic to AWS resources and your application hosted on a VM. The virtual firewall is stateful, which means that outbound responses are allowed for allowed inbound traffic. Therefore, you would configure a security group for your web server VMs to allow web traffic by opening TCP/IP port 80, while a database server ought to be in a Security Group that blocks web traffic on port 80. (The port number 80 is well known for web servers, but internal web servers can have any port number assigned that is not in use.) This measure guards against attacks on your database from the outside world using web traffic. Thus the onus is on you as the installer of your application to ensure that it is securely installed in your private cloud environment. You can simplify and extend monitoring of this configuration by using the same tags for each resource and security group as well as the corresponding virtual network.

Firewalls are devices that analyze the network data to block malicious traffic.

Security Monitoring

Let us consider monitoring in terms of the data flow when accessing resources. Those security mechanisms and devices that appear during that data journey should be monitored. as discussed in this section.

The first defense mechanism that is employed to deter security breaches is a firewall. Individual firewalls should be placed at the boundary of every security container. The cloud itself, being a security container, will therefore have an outer firewall. Where you have containers within containers, which is not uncommon, then you would have firewall after firewall to traverse if you followed the data flow. The firewall will contain rules that tell it what traffic to allow through and what traffic to block. You can be alerted if someone tries to enter a security container and is blocked by the firewall. The alert can be a message, an email, or an entry in a log file that can be monitored. However, a security system will have several components such as firewalls, user authentication and identification protocols, intrusion detection and prevention ware, and user-based security such as antivirus and antimalware tools. Such a system, if properly configured, will generate logs that keep track of users, services provided to the users, and data. Human review of the log files is not possible since the logs grow to a large size very quickly. As such, software is used to aggregate the logs and then analyze the

server and firewall logs; this is the first step in detecting any suspicious activity.

The second line of defense is at the user authentication and identification stage. If several failed attempts are made to log in to a system or a service, then the monitoring tools ought to alert the cloud system administrator of this. If the tools are automated, then they may deny further logins on that account or to that service by the user concerned. Additionally, certain traffic can be deduced to be suspect because it is trying to use a service that is not allowed for a particular user or attempts are being made to retrieve data that would normally not be needed by the user. All such traffic can be analyzed and reported by software that is known as an intrusion detection system (IDS).

The third line of defense is on the client side. The client devices themselves need to be protected against malicious attacks so that user login details are not compromised while accessing cloud services. Usually this takes the form of antimalware (including antivirus and antispoofing systems, personal firewalls, and means to block tracking cookies[6]) that is installed on the client device that you use to access cloud services. Mostly, such end-user client devices are outside the scope of a cloud service provider to monitor or configure. It is on the end-user side, then, that the weak link in the security chain becomes evident. However, with thin or zero client computing, this weak

link becomes irrelevant, as the client device should have a centrally managed operating system.

SIEM, or security information and event management, provides a holistic view of your systems' security footprint, and so we consider it next.

SIEM

SIEM is a combination of two technologies, security information management (SIM) and security event management (SEM). It consists of a set of tools and services to provide a view of an organization's security footprint:

1. SIEM tools make visible the operation of information security systems in real time.

2. SIEM allows the aggregation and consolidation of event logs from numerous sources, such as infrastructure, application, and security tools.

3. By means of if-then or when-then rules, SIEM initiates the triggering of events and their notification. This is done by parsing and analyzing the consolidated log files. Rules are then applied to convert data events into meaningful security issues.

4. Dashboards are provided to allow users to visualize monitoring, events, and notifications, as well as any analytics (filtering, statistical, or aggregation).

Generally, a profile is created that defines the behavior of the cloud services under normal conditions and various predefined security events. Default rules, alerts, reports, and dashboards are provided that can be customized to meet your specific needs.

Some of the use cases of SIEM are as follows.

- To detect any unauthorized network connections to and from the cloud services.
- To search for insecure protocols to enable you to assess whether you should permit them.
- To inspect traffic flows in and out of the demilitarized zone, which manages connections between untrusted networks, such as the internet, and trusted ones in your cloud system.

10

TRANSITIONING TO THE CLOUD

Suppose you have decided that you want to use cloud computing. One of the yardsticks you will need to assess a cloud service is through service-level agreements (SLAs). If you were to look up SLAs and metrics on the internet or in the numerous books that have been written on cloud computing, you would frequently encounter a discussion of SLA metrics such as network capacity (bandwidth, latency, or throughput), storage device capacity, server capacity (number of CPUs, CPU clock frequency, size of RAM), instance starting time (time required to initialize a new instance of a virtual machine), horizontal storage scalability (the permissible storage capacity changes in response to increased workloads), horizontal server scalability (server capacity changes in response to increased workloads expressed as the number of virtual servers in a cloud's resource pool), and so forth, amounting to perhaps hundreds of such items. These SLAs are good if you need

to create and operate your own cloud. But why should you, as a buyer and user of cloud services, need to worry about them? After all, cloud computing is supposed to provide a layer of abstraction that should minimize your technology headache, not increase it. We therefore do not follow the well-trodden route that most books on cloud computing take but instead examine SLAs and the metrics that are relevant to you as a user of cloud services. Additionally, we provide a checklist that you can adapt to assess cloud services. This gives us a set of critical success factors. We follow up with a discussion on how to assess your own maturity as a cloud service user by considering a cloud maturity model from a user's perspective. That way you can assess your cloud adoption aligned to best practices, as well as track your progress over time as you use cloud computing. We follow this with a discussion on interoperability, or how your cloud services will interoperate with other cloud services, possibly from a different vendor, or interoperate with your on-premises services. We consider these key aspects of transitioning to the cloud, which are supported by previous chapters, especially data and security.

Critical Success Factors

When transitioning to the cloud or considering a new cloud service, you will need to develop a set of requirements to serve as a yardstick for comparing cloud services. Addition-

ally, the requirements will help you crystallize your SLAs so that you may assess whether a cloud service is appropriate for you. Remember, cloud computing may not fulfill all your IT or computational needs; there will be instances, such as if very high-speed computing or resilience is needed, when you will want to use traditional computing. The checklist below can help you define your requirements for cloud computing. You can also use the requirements to define the critical success factors for the cloud services that you purchase. I have placed the requirements into three general categories: functional, nonfunctional, and business-related requirements. They are generic enough for you to use them for all the deployment and service models of cloud computing.

1. Functional requirements
    a. Maturity
    b. Interoperability
    c. Feature set
    d. Usage model
2. Nonfunctional requirements
    a. Security
    b. Availability
    c. Resilience
    d. Network capacity

3. Business-related requirements
    a. Price and value
    b. Risks
    c. Business continuity
    d. Support model
    e. Reporting and billing

You can create a score matrix by scoring each of the requirements from, say, 0 to 3, with 0 denoting that a requirement is not met and 3 denoting that a requirement is met fully. Also, each requirement could be weighted to reflect what you consider to be most important. For instance, maturity might have a weighting of 0 if that requirement is not at all important to you. However, having the right feature set might have a weighting of 3 to denote its high importance or relevance to you. Multiplying the weighting by the score will then provide you with a weighted score card that you can use to compare cloud services.

Cloud Maturity Model

The matrix in figure 17 depicts a maturity model for cloud computing. It has five levels of maturity: performed, defined, managed, adapted, and optimized (the highest level

of maturity). The rows labeled "Focus" and "Success factors" describe the level of maturity in terms of its main characteristics and benefits, respectively. The other five rows consider the maturity characteristics of your cloud service in terms of (1) the people engaged to purchase, use, and manage the cloud services; (2) the processes that interact with the cloud services; (3) the financial and usage monitoring and reporting of the services; (4) the security, regulatory, functional, and financial oversight that is provided to ensure that the cloud services meet your needs; and (5) the financial management in place to ensure that the cloud services remain financially viable. The maturity levels are not mutually exclusive for these five characteristics; you might have a maturity level of 1 for people and a maturity level of 3 for processes, for example. However, the overall maturity of your cloud service would be denoted as that level at which most characteristics coalesce.

You can use the maturity model in two ways: first, to create a strategy to improve your use and commissioning of cloud services (this will enable you to transition from one maturity level to a higher one), and second, to understand the best practices of cloud computing. The last column, labeled "Optimized," describes the optimized maturity level and should be considered the ultimate goal.

We have considered the critical success factors and the maturity levels related to cloud computing. Let us now turn our attention to a more detailed approach by

	Maturity →	
	1. Performed	2. Defined
Focus	Functional capability Meets business needs on an ad hoc basis	Standardized capability Cost-efficient and secure
People	Little or no knowledge of cloud computing	Basic roles and responsibilities defined
Processes	No interoperability Cloud service reuse undefined	Data interoperability defined Best practice defined Service life cycle defined
Monitoring/ Reporting	Minimal monitoring	Metrics and KPIs defined
Governance	No clear ownership	Ownership defined Sponsorship from management
Financial Control	Credit card generally used	Billing and utilization statements tracked
Success Factors	New or consolidated business processes	Cost-effective due to standardization

Figure 17 Cloud maturity model.

considering interoperability and its concomitant factors when transitioning to the cloud.

Interoperability

For business continuity or commercial reasons, you will want to ensure that a cloud service can be replaced by a

Maturity ⟶

3. Managed	4. Adapted	5. Optimized
Effective capability Aligned to business needs	Flexible capability Responsive to business needs	Automated capability Business functions are automated
Learning and Development in place	Knowledge management in place promoting reusability	Automated implementation requiring little support
Data and process interoperability implemented	Cloud service reuse in place Business functions monitored	Business functions continuously improved Optimized processes
Metrics and KPIs reported	Metrics and KPIs tracked	Metrics continuously optimized to meet business needs
Governance process defined and followed Communication plan	Metrics and KPIs governed across business units	Federated governance with all business units participating
Penalties and chargebacks defined	Financial planning Cost management	Cost optimization Cloud computing considered a profit center
Agile and flexible resulting in faster time to market	**Measurable and repeatable outcomes**	**Continuously being improved**

Figure 17 (continued)

similar one from another provider. This calls for interoperability between services. Interoperability is the capability to use the same or similar cloud services offered by different cloud service providers.

The scope of interoperability applies not only to technical matters but also to such topics as the integration of billing, reporting, management, business processes, and, of course, data. Interoperability on all these dimensions

Interoperability is the capability to use the same or similar cloud services offered by different cloud service providers.

should be available regardless of the cloud delivery model used—private, public, hybrid, or community. The following discussion forms a checklist that you can adopt and extend to ensure that your cloud services have interoperability.

Governance and Auditing

If you are using an auditing process and policy for one cloud service, you should ensure that any other services you procure to interoperate with your current cloud service conform to that same auditing standard. Similarly, for governance, you need to ensure that the same governance board within your organization has oversight of all the cloud services, especially those that need to meet interoperability requirements.

Compliance

If you need to comply with industry-specific or country-wide regulations, you should determine whether the cloud services you use fall under their scope. If they do, then all the cloud services that need to interoperate have to comply with the same regulations. Generally, from a cloud services perspective, there tends to be a gradation between general regulations and industry-specific regulations as the cloud service becomes more specialist in nature. This is shown in figure 18, where general utility services such as email and office productivity have global scope and must abide by general international laws and regulations. When

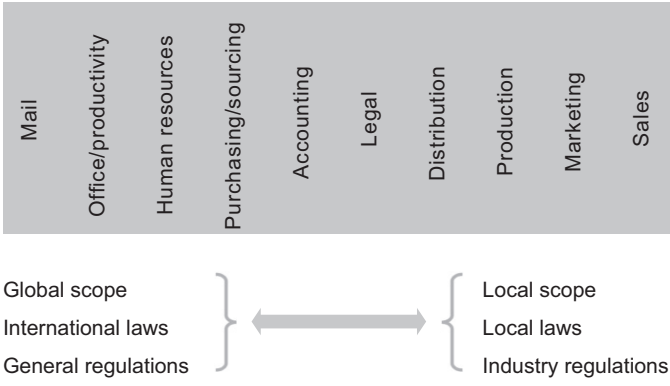


Figure 18 Scoping requirements for interoperability.

more specialist cloud services are considered, local laws and industry-specific regulations normally apply. This is a general observation that may not always apply to a specific cloud service, however.

Security and Data Integrity

For data integrity, the same encryption standards and schemes must be in place across the cloud services that need to interoperate. Otherwise, data flow will not occur seamlessly. For security, you should ensure that the same security procedures are in place so that there is no disparity in security levels between interoperable cloud services. Also, the same secret and password vaults need to be used to ensure commonality between services. (Generally, this

is achieved through single sign-on mechanisms.) Finally, the users of all the services should undergo a common security training program because it is best to standardize on a single security policy within an organization.

Data Integration

For seamless data integration between or with cloud services, two factors are important: a common data format and a common data model. Format refers to the way data are represented; a spreadsheet stores information in a different format from a text file, for instance. Even when you have the same format, there needs to be commonality about what the data relate to. This means having the same model for the metadata. For example, if you have two text files, both in the same format, but one file contains information about your inventory whereas the other one contains information about your salary, then the two files have dissimilar data models, but these data models should have common metadata: the address fields should be labeled the same, for instance. So you need to ensure that services that interoperate use the same data format and models.

Process Integration

Besides having the same data format and model for interoperability, you should ensure that the same tasks are carried out on those data by the processes within the cloud services. This is known as process integration. There are

two aspects to process integration. First, you want one process in a cloud service to receive data from another and commence a workflow that is an extension of the first process; second, equivalent processes in both interoperable clouds should be the same so that you may interchange them, almost in a plug-and-play manner. In any case, you need to ensure that processes use the same business process language (BPL) and protocol so that your cloud services are interoperable. To make things easy with data and process integration, an enterprise service bus (ESB) is usually used to ensure that many of the technical integration concerns are hidden at a lower layer of abstraction.

Business Continuity

There are two aspects to business continuity: disaster recovery and service availability. Service availability concerns the need to increase capacity to meet unexpected user demand. In such a case, you could use a second Availability Zone or region with your current cloud provider or use cloud bursting to a secondary cloud provider. This requires in-depth technical design that considers the right protocol and standards to ensure that control passes from one cloud service to another seamlessly. In addition, it is likely that data and process integration requirements will have to be met to enable this.

Disaster recovery refers to when you need to continue business operations as usual even if some unforeseen

event, or disaster, were to strike your cloud service. It entails employing a secondary cloud service using a separate Virtual Private Cloud and region, either from the same or from a different cloud provider, for failover to the secondary system. There are two main disaster recovery strategies, active-passive and active-active. We discuss these further below.

Disaster Recovery Strategies

In an active-active disaster recovery strategy, the same capability is replicated across two regions. Both primary and secondary regions provide services to customers, and you balance the load across them during normal operation. When disaster strikes one of the active regions, you still have its replica active in another region. Best practice is to use two cloud accounts and different regions so that you can have financial as well as technical and physical separation to enhance availability. Both RTO and RPO should be zero with this approach as failover will occur to an active region. Because you are balancing the load between your primary and secondary services in the active-active configuration, performance will be impaired should disaster strike, as one of the nodes will be unavailable. But at least business continuity will be maintained.

In the active-passive strategy, a secondary region is available that is not serving customers. It can be classified into three categories according to provisioning of cloud

resources: (1) the secondary region does not have any cloud resources provisioned, (2) those cloud resources related to data are provisioned, and (3) business-critical cloud resources are provisioned and running on standby. These three active-passive strategies are known as backup and restore, pilot light, and hot (or warm) standby, respectively.

With the backup and restore strategy, you create backups of your data and cloud resources (through infrastructure as code) and store the backups on both the primary and the secondary cloud regions. On failover, cloud resources are created in the secondary region from the backups. Because no services apart from storage for the backups are running, this is the least expensive option, but it has the highest RTO and RPO associated with it. These metrics, however, can be minimized if Infrastructure-as-Code is utilized whereby you run a program (referred to as a runbook or playbook for such tools as Terraform, Ansible, or AWS's CloudFormation) to automatically provision your cloud services, which then obviates the need to use backups.

With the pilot light strategy, the data are replicated across both the primary and secondary regions and so are live. However, the cloud services are created from a backup as part of the failover process, although basic infrastructure elements such as elastic load balancing can be live. This is a more expensive option than the backup

and restore option but should have an RPO approaching zero as the data stores and databases should be current, or nearly current.

In the warm standby strategy the secondary region maintains live data and handles service requests in a reduced capacity, but it must be scaled significantly to meet the production load should the primary region fail. This configuration approaches the active-active strategy except that it implements a minimum capacity during normal operation. It is the most expensive of the active-passive strategies although less expensive than the active-active strategy. Its RTO and RPO should approach zero.

Monitoring and Alerting

To some extent, cloud services are becoming self-healing in that the monitoring and remediation processes are automated, and this is especially true for cloud native services. However, you should ensure that any reports or alerts arising from the monitoring of your cloud services use a common standard and format for communication to your team. So, for example, you could use a common platform such as Slack for relevant team members to receive any such notifications. This way your team is able to respond to the alerts in a timely manner. In any case, the more you automate failure recovery through self-healing and observability, the more mature your cloud service will be and the less human involvement will be needed.

Additionally, you can be proactive by performing certain tests on a regular basis. These tests could be, for example, stress tests to ensure that you have the right capacity for serving your customers or penetration tests to assess whether any security gaps exist that might result in a security breach.

Billing and Reporting

The billing processes, formats, and reports (I am using these terms generically to denote statements and invoices) should all be similar between the services for you to assess the total cost of ownership in a meaningful and rapid way. You may also need to ensure that the people who work with billing have a common understanding of the processes and reports for you to have interoperability not only with the processes and reports but also with people.

Business Processes

Transitioning to the cloud gives you a good opportunity to redesign your business processes to make them more efficient. Processes connect different parts of a business, so improving them should improve your whole business. Cloud computing can be an excellent enabler in reengineering

a business because of the automated business processes that you can commission with a BPaaS cloud. Even if you do not go all the way to BPaaS, you can still derive greater efficiencies in your processes through the onboarding of INaaS and SaaS cloud services—and thereby effect business change.

The business process redesign (BPR) model (created by Hammer and Champy) shown in figure 19 provides a good procedure for redesigning your business processes.[1] The main idea in adopting the BPR model is that attempts to improve the efficiency of organizations can succeed only if the processes are also improved. (This also applies to IoT-related processes as the machine-to-machine interactions need to be similarly efficient.) Thus, when using the BPR model, one of the questions asked (during step 3 of figure 19) is whether a process or one of its workflows is necessary. Only those deemed necessary are kept, improved, and implemented. In step 5, you will design some of the processes as BPaaS in their entirety, some using one of the other cloud service models, and others outside the cloud. The criteria that you use to assess whether a service ought to have a cloud computing component are determined by your requirements and critical success factors, which we outlined at the start of this chapter.

Data are critical to business processes as well as when transitioning to the cloud, as per our discussion on

Transitioning to the cloud gives you a good opportunity to redesign your business processes to make them more efficient.

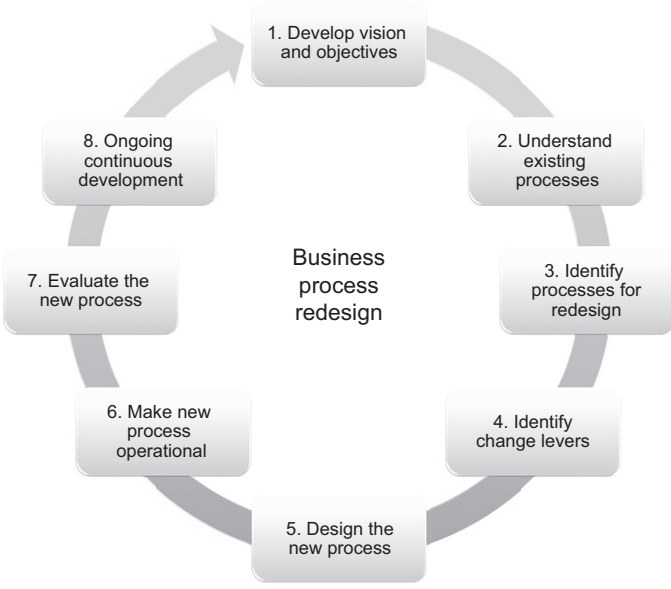


Figure 19 Business process redesign (BPR) model. *Source:* Based on Michael Hammer and James Champy, *Reengineering the Corporation: A Manifesto for Business Revolution* (HarperBusiness, 2006).

interoperability. Thus the discussion in chapter 8 on migrating data to the cloud should be consulted, as data migration will generally determine the success of transitioning to the cloud.

11

PUBLIC CLOUD EXAMPLES

To this point we have addressed the key concepts of cloud computing. In this chapter we consider some public cloud offerings. The principal ones are Amazon's Amazon Web Service (AWS), Microsoft's Azure, and the Google's Google Cloud Platform (GCP). We consider the concepts of these three public clouds briefly in this chapter and then list a select number of cloud services and compare them with those available on AWS, Azure, and GCP. This is followed by a discussion of reference architectures in the next chapter that uses a web application as an example.

Amazon's AWS Concepts

At a top level, AWS provides Regions, Availability Zones, and Virtual Private Clouds for its cloud resources to reside

in. A Region represents a geographic area where physical data centers are located.[1] Thus a Region has several physically separate and colocated data centers that provide fault tolerance and stability. The physical data centers in a single Region can combine to form logical data centers, also known as Availability Zones. No two Availability Zones share the same physical data center, and so each Availability Zone has its own separate cooling and power systems for the servers. A well-architected solution will use more than one Availability Zone because it offers failover capabilities for disaster recovery.

How would you group these Availability Zones to ensure they are connected in a manner that allows your applications to communicate across them? The answer is, through the mechanism of a Virtual Private Cloud (VPC). An AWS VPC is a logically segmented network and is associated with a specific region that allows you to span across that region's Availability Zones. But what if you want greater isolation so that your Availability Zone spans across regions as well? There are two solutions to this: (1) VPC peering across regions and (2) the AWS Transit Gateway. Both allow you to have redundancy of AWS resources across regions and thereby address business continuity requirements.

VPC peering is performed by a VPC peering connection between two VPCs that allows you to route traffic between your private (IPv4 or IPv6) network addresses. In addition to using VPC peering across regions, you can

also have the VPCs peer across AWS accounts, and that allows you to have financial isolation as well. When using a VPC peer connection, there is no single point of failure for communication or a bandwidth bottleneck because the connection is performed via existing routing tables rather than through any physical device or link. VPC peering across regions allows the VPC resources (e.g., Elastic Compute Cloud [EC2] instances, RDS databases, and lambda functions) to communicate with each other without requiring any gateways, VPN connections, or other network appliances, and the traffic remains in the private IP space. Traffic always remains on the global AWS backbone and does not traverse the public internet, which reduces security threats.

The AWS Transit Gateway is a central hub that allows you to connect your VPCs at multiple regions and additionally your on-premises servers. This reduces the complexity of having to maintain routing tables that are required with VPC peering and so is a better option when you have a large number of VPC peers. The Transit Gateway also allows you to connect VPCs across different AWS accounts so that you can have a well-architected implementation that caters to business continuity and disaster recovery requirements. Also, because of its central position as a hub, the AWS Transit Gateway's Network Manager provides a complete view of your entire network that can straddle across on-premises, AWS, and other cloud providers.

Google Cloud Platform Concepts

GCP physical resources in a data center are collectively known as a cluster because the plethora of resources such as servers, storage, and network devices (together with their concomitant building facility, power, and cooling infrastructure) forms a physical cluster for computational purposes. The clusters are mapped by GCP to a zone, which is a logical collection of resources in a region; a region can have three or more zones and is a specific geographic location where you can host your resources. The logical zone and physical cluster are then mapped by you for your project. This mapping between a logical zone and a physical cluster remains consistent within a project, although another project in the same region may have a completely different zone-to-cluster mapping. However, a project never has two zones mapped to the same physical cluster. You can align zone-to-cluster mappings between projects by using VPC networks. It is best practice to spread apps across projects using a shared VPC network for consistent zone-to-cluster mappings, and to use different zones in the same region—alternatively, zones in multiple regions—to ensure diversity by separating clusters between projects, as figure 20 shows.

In figure 20, you have two projects, project 1 and project 2, that use zonal resources from a region. These resources are within clusters, and you can define which

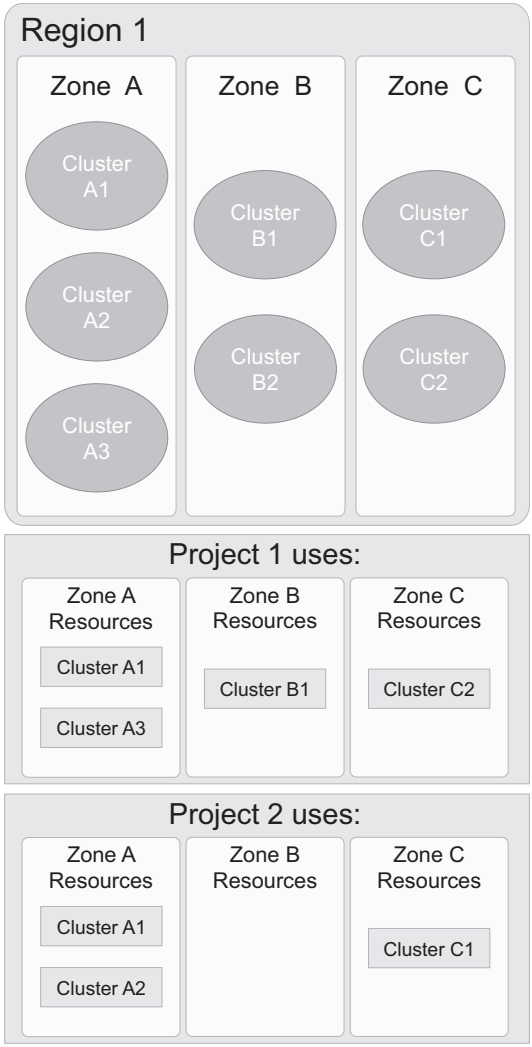


Figure 20 Zone to cluster aps for projects.

clusters to use within a project to achieve the required isolation or redundancy in your solution. This approach leverages the key capabilities of projects; as GCP does not require each project to have its own network, you can traverse zones across a region to obtain isolation or redundancy. A further advantage is that aspects such as shared services and security auditing tools are much easier to orchestrate since their access spans across the project. In AWS, on the other hand, you are restricted to using resources within a VPC and, should you want isolation, you will need to use extra resources—and consequently incur costs and increased complexity—such as separate accounts, VPC peering, or the Transit Gateway.

From a security perspective, GCP uses the concept of trust-level hierarchies. A hierarchy consists of organizations, folders, projects, and cloud services, as figure 21 illustrates. The resources in GCP you access are defined by the identity and access management (IAM) policies you set at each hierarchical level. Because resources inherit the policies of their parent resource, the effective IAM policy for a resource is the union of the policy for that resource and that inherited from its parent.

In figure 21, the organization level represents your company, and IAM roles granted at this level are inherited by all resources under that level. Similarly, the folders level represents departments or business domains in the company and may contain projects, other folders, or a

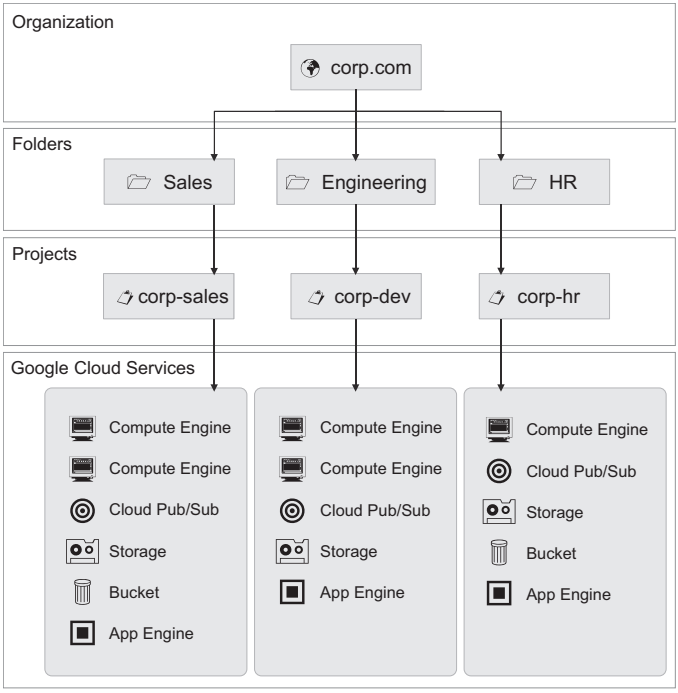


Figure 21 Resource hierarchy in GCP.

combination of projects and folders. IAM roles and access controls granted at the highest folder level are inherited by projects or other folders contained in that parent folder. The projects level represents a trust boundary within your company, and services within a project have an inherent level of trust by default. For example, a compute engine

instance can access cloud storage buckets within the same project. The cloud services level provides a more granular level of access control whereby you can provide access to specific resources to apps or users. One way to implement a zero trust security model is to ensure that very few, if any, access controls and roles are defined at the higher hierarchical levels and to define all your policies at the project or cloud resource levels. However, you would do this at the Google Group level rather than individually for users to ease the maintenance of your security policies. It is best practice to use the principle of least privilege for IAM roles, and to grant roles at the smallest scope or the lowest hierarchical level. Another best practice is to label your resources so that you can easily audit, group, and filter them.

Microsoft Azure Concepts

There is a hierarchy of resource usage, and consequently billing, that comes with Azure. At the top level is the Azure account that gets billed. The account has two requirements: billing capability (a credit card or bank information) and contact information (an email address). Within an account, you can have several subscriptions. A subscription is a logical container for Azure resources, and every subscription is tied to one, and only one, account, although an account can have multiple subscriptions. You would

generally have subscriptions for separate business activities (one for sales, one for accounting, one for HR, etc.) or capabilities (one for development, another for testing, a third for production). Every subscription can have a resource attached to it. A resource is a cloud computing resource that is managed by Azure. Virtual machines, storage, virtual networks, and functions are examples of resources. These can be grouped together in a resource group, which is a logical container for grouping related resources in a subscription and is commonly used to represent a collection of assets needed to support a workload, application, or specific function within a subscription. Each resource can exist in only one resource group. A management group, on the other hand, is a logical container that you use across subscriptions. You can therefore define a hierarchy comprising management groups, subscriptions, resource groups, and resources to manage access, security policies, and compliance through inheritance.

Azure relies on Active Directory (AD) for implementing IAM and role-based access controls. Azure AD is a cloud-based service that allows you to sign in and access Azure resources. When an Azure account is created (or even when you sign up for a Microsoft 365 subscription), an Azure tenant is created for you. An Azure tenant is a dedicated instance of Azure AD assigned to you (or your organization) and it represents a single organization. The hierarchy of Resources, Resource Groups, Management

Groups, and Subscriptions (quite similar to the resource hierarchy of Google's GCP shown in figure 21) are thus accessed using Azure AD and its security structures and capabilities.

There is a plethora of cloud computing resources available to you. Microsoft places them into ten categories for Azure:

1. Compute,
2. Networking,
3. Storage,
4. Mobile,
5. Databases,
6. Web,
7. IoT,
8. Big Data,
9. Artificial Intelligence, and
10. DevOps.

Most of these are abstracted services that are managed and hosted by Azure, but at the base you have the first three categories, on which most of the other services

and resources are built. For connected resources in a design to communicate with each other, a virtual network (VNet) is used to segregate the resources and separate them from other, unconnected ones. The VNet logically segments the network to realize this, in the same way that AWS uses VPC. It is best practice to have VNets defined separately for public IP address ranges, private ones, and management ones. A management VNet should be created to manage the cloud resources in Azure. Additionally, segment your application so that it has separate private VNets for each layer: one for the presentation layer, another for the business logic layer, and yet another for the data layer, for instance. Should you want resources on separate VNets to communicate with each other, you can connect the VNets as long as there is no overlap in the IP network range definitions. And this VNet connection is allowable even if the VNets are in different locations. You connect the VNets using VNet peering, in much the same way that VPC peering is used in AWS. In Azure, there are two types of peering: VNet peering for connecting VNets in the same Azure region and Global VNet peering for connecting VNets across multiple Azure regions. As with AWS and GCP, all network traffic using VNet peering remains within the Azure backbone without traversing the internet and so is less susceptible to security breaches.

To distribute workloads across multiple groups of computing resources based on the type of network traffic,

load balancing is used. Principally, load balancing optimizes resource usage, minimizes response times, and maximizes throughput. Also, when used across different accounts, subscriptions, regions, or data centers, it effectively implements failover by sharing the workload across redundant computing resources.

With Azure, load balancing is categorized in accordance with two dimensions: (1) global versus regional and (2) HTTP(S) versus non-HTTP(S).

Global load-balancing services distribute network traffic across regional backends, clouds, or hybrid on-premises services. They route end-user traffic to the closest available backend (where resources are located) and react to changes in service reliability or performance to maximize availability and performance. Regional load-balancing services distribute traffic within VNets across VMs, containers, or clusters within a region in a VNet.

At the application layer, HTTP(S) load balancers are used as they are primarily intended for web applications or other HTTP(S) end-points. Non-HTTP(S) load balancing is used for non-web workloads where HTTP or HTTPS protocols are not used.

Azure provides four load-balancing solutions: Front Door, Traffic Manager, Application Gateway, and Azure Load Balancer, as table 4 shows.

Azure Front Door is an application delivery network that provides global load balancing and site acceleration

Table 4 Azure load-balancing services

Load-Balancing Service	Scope	Traffic
Azure Front Door	Global	HTTP(S))
Application Gateway	Regional	HTTP(S)
Traffic Manager	Global	Non-HTTP(S)
Azure Load Balancer	Regional	Non-HTTP(S)

service for web applications to improve their latency or response times and their availability.

Application Gateway is a managed service for application delivery that uses HTTP/HTTPS information to distribute traffic across resources within a given data center. Application gateways have a single public IP address but can host up to twenty web sites, each with a pool of back-end resources. Application Gateway relies on HTTP host headers to differentiate between websites. When you enable SSL offload, the Application Gateway can also use server name indication to distinguish between websites.

Azure Front Door and Application Gateway combined provide the same functionality as GCP's HTTP(S) Load Balancer since the latter can be deployed either as a global load balancer or as a region-specific load balancer for HTTP(S) traffic.

Traffic Manager is a DNS-based traffic load balancer that enables you to distribute traffic to services across global Azure regions while providing high availability and

responsiveness. Because it is a DNS-based load balancer, it works only at the domain level and so cannot failover as quickly as Front Door.

Azure Load Balancer is a high-performance, ultra-low-latency load balancer for all UDP and TCP traffic. It is zone redundant and so ensures high availability across availability zones. It is similar to GCP's Network Load Balancer.

At the compute level, you have VMs (known as Compute Engines on GCP and EC2 instances on AWS) that you can purchase on a pay-as-you-go subscription basis. You can optionally install containers on your VMs since container-optimized VM images are available. From these you can spin your VM instances and start using containers, which can be LXC, Docker, or podman containers if using Linux machines or the latter two if using Windows. If you want Microsoft to manage these instances and containers for you, you can use Azure Container Instances (known as ECS on AWS and App Engine on GCP).

Public Cloud Resources

Although most resources of the three public cloud providers map to each other, though using different names, there can be slight differences in functionality, and it is always a good idea to consult their official documentation. Table 5 compares some of the cloud resources provided by AWS, Azure, and GCP.

Table 5 Comparison of selected public cloud services

#	Cloud Service	AWS Resource	Azure Resource	GCP Resource	Description
1	Virtual machine	EC2	Virtual Machine	Compute Engine	Server instances that allow users to deploy, manage, and maintain software in a secure manner.
2	Machine scaling	Auto Scaling Group	Scale Set	Managed instance groups; Autoscaler	Allows the number of virtual machines (VMs) to be increased or decreased in accordance with metrics and parameters that you define for CPU or RAM usage.
3	Containers	ECS	Container Instance	App Engine	A managed hosting platform for containers and microservices.
4	Container registry	Elastic Container Registry	Container Registry	Container/ Artifact Registry	A managed, private Docker registry service for containers and related artifacts.
5	Managed containers	Fargate	Container Instance	Cloud Run	Serverless container execution service.

Table 5 (continued)

#	Cloud Service	AWS Resource	Azure Resource	GCP Resource	Description
6	Microservices or serverless compute engines	Lambda Functions	Azure Functions	Cloud Functions	Execute code on systems in response to events or schedules without provisioning or managing machines. These are short-lived processes used for stateless architectures.
7	Serverless workflow	Step Functions	Logic apps	Composer	Provides serverless workflows that are used to orchestrate serverless compute services or microservices.
8	API publishing	API Gateway	API Management	Apigee	Service for publishing APIs to external and internal consumers.
9	Managed container service	Elastic Beanstalk	App Service	App Engine	Managed hosting platform for running programs and services for deploying web applications.
10	Containerized web apps	App Runner	Web App for containers	App Engine	Allow easy deployment and running Windows and Linux containers for web apps.

11	Kubernetes	Elastic Kubernetes Service (EKS)	Azure Kubernetes Service (AKS)	Google Kubernetes Engine (GKE)	Allow deployment and management of orchestrated containers using Kubernetes.
12	Application meshes	App Mesh	Service Fabric Mesh	Anthos Service Mesh	Fully managed service to deploy microservice-based applications without you having to manage virtual machines, storage, or networking.
13	Message queues	Simple Queue Service (SQS)	Queue Storage	Cloud Pub/Sub	Managed message queueing service for decoupling applications.
14	Event notification	Simple Notification Service (SNS)	Service Bus	Cloud Pub/Sub	Provides event notification of messages.
15	Publish-subscribe messaging	Amazon MQ	Service Bus	Cloud Pub/Sub	A set of cloud-based, message-oriented middleware technologies for reliable message queuing and durable publish-subscribe messaging.
16	Managed databases	RDS	Azure Database	Cloud SQL	Managed relational database service in which resiliency, scale, and maintenance are handled by the platform.
17	NoSQL database	DynamoDB	Cosmos DB	Cloud Firestore; Firebase Realtime	Managed multimodel database that is globally distributed and supports unstructured data.

Table 5 (continued)

#	Cloud Service	AWS Resource	Azure Resource	GCP Resource	Description
18	Data caching	Elasticache	Cache for Redis	Cloud Memory Store and Redis Enterprise Cloud	Distributed in-memory data cache typically used to store nontransactional data.
19	Object storage	S3 Bucket	Azure Blob Storage	Cloud Storage	Object storage service.
20	Block storage	Elastic Block Storage	Azure Managed Disks	Persistent Disk; Local SSD	Block storage optimized for I/O intensive operations used for VM storage.
21	Archival	S3 Glacier; Deep Archive	Storage Cool Tier	Cloud Storage (Archive)	Low-cost storage of data that are infrequently accessed and long-lived.
22	Backup	Backup	Backup	Cloud Storage (Coldline)	Used to back up and recover files from the cloud to provide off-site protection against data loss.
23	Data synchronization	DataSync	File Sync; Azure AzCopy	Data Stream	Online service for synchronizing or replicating data.

24	Cloud virtual networking	VPC	VNet	VPC	Isolated, private environment in the cloud with virtual networking that provides IP addresses, creation of subnets, configuration of route tables, and network gateways.
25	Web application firewall (WAF)	WAF	Application Gateway WAF	Cloud Armor	Provides centralized protection of web applications from common exploits and vulnerabilities.
26	Identity and access management (IAM)	IAM	Active Directory	Cloud Identity	Allows and denies access to resources through role-based access controls and policies.
27	Content delivery network (CDN)	CloudFront	Azure CDN	Cloud CDN	Distributed network of servers used to deliver web content to users at low latency.
28	Dynamic name service (DNS)	Route 53	DNS	Cloud DNS	DNS record management.
29	DNS Routing	Route 53	Traffic Manager	Cloud DNS	Resolves host names to their IP addresses and routes traffic to appropriate hosts or services.
30	Network load balancing	NLB	Load Balancer	NLB, GLB	Balances internet traffic at layer 4 (TCP or UDP) at a cross-regional or global level.

Table 5 (continued)

#	Cloud Service	AWS Resource	Azure Resource	GCP Resource	Description
31	Application load balancing	App LB	Application Gateway	Global Load Balancing	Load balancing at ISO layer 7 for applications that also supports SSL traffic.
32	Cloud resource monitoring	CloudTrail	Activity Log	Cloud Audit Logs	For monitoring and auditing cloud resources such as storage and VMs
33	Cloud monitoring	CloudWatch	Azure Insights	Cloud Monitoring	Dashboards to assess performance, uptime, and overall health of cloud-based applications.
34	Cloud automation	Cloud Formation	Azure Automation	Cloud Deployment Manager	Automates IT tasks related to cloud resources that include process automation, configuration management, update management, and deployment.
35	Cost optimization	Cost Explorer	Cost Management	Cost Management	Optimization of costs while maximizing cloud resources utilization.

12

REFERENCE ARCHITECTURES

A reference architecture is a predefined architectural pattern or collection of patterns that has proven useful in solving a particular business or technical problem. It encompasses (1) a common vocabulary, (2) reusable designs, and (3) best practices that can be used as a template for you to get started. In this chapter we begin with a discussion of the core architectural principles that you should bear in mind when creating a cloud-based design. This discussion is followed by a tour of how a cloud-based web-application evolves to a serverless one using cloud native principles.

Architecture Principles

You should architect cloud solutions with maintainability, availability, scalability, and security in mind. The principles

outlined in this section provide a guide to creating well-architected solutions.

In summary, the guiding principles we consider are separation of concerns, abstraction, automation, statelessness, defense in depth, and bounded contexts. The scope of these principles can vary from software design to overall solution design that includes cloud fabric design.

Separation of Concerns

According to the principle of separation of concerns, a particular solution should be broken up into building blocks such that each building block has a single concern. At a high level, this means that you break up your solution into layers or tiers: the presentation layer, the business logic layer, the data access layer, the data layer, and the storage layer. From a more granular perspective, it means that you break up your solution into services whereby each service has a singular concern; the solution then becomes a composite of different services.

Bounded Contexts

Domain-driven design ensures separation between the business logic and the technical details. Bounded contexts are central to domain-driven design because they provide the means to tackle complexity by separating a particular solution into business contexts that are bound

to a particular business concern. So the idea of bounded contexts is somewhat related to and driven by the separation of concerns principle.

Abstraction

Abstraction attempts to hide the operational or technical details and allows you to create a solution that meets business requirements. Since most cloud providers offer a broad range of managed services that free you from having to manage the back-end infrastructure and its operating system, you should favor using these services. To avoid getting locked in to any one cloud service provider, you should adopt a policy of using two providers with interoperability as the cornerstone of your policy. That way, one cloud service can provide a hot or cold standby service to the main one that you use.

Reusability

Favor reusability by tagging, documenting and versioning your components, and thereafter making use of image, document, and component repositories and libraries. A higher level of abstraction enables the reusability of components that provide services. A catalog of services, commonly referred to as a service catalog, will enable you to reuse a particular service when needed as part of the overall service-oriented solution.

Statelessness

State refers to user or system state. As examples, the former could be the user's address, items in the shopping cart, or the person's employee ID; the latter could be the code version that is running, the number of virtual machines (VMs) or containers being used, or the amount of memory being consumed. Since maintaining state in an event-driven, microservices-based architecture is difficult, you should favor statelessness as much as possible. The advantages of stateless components are:

1. Scalability: To scale up, you simply spin up more containers; to scale down, your containers are killed. As they do not maintain state, it is easy to scale your application or service.

2. Self-healing: Should a container or a VM (in a scaling group) fail, it can be replaced by another after it is gracefully terminated.

3. Deployment: Especially in a canary deployment, should you need to roll back to the previous deployment, you can do so quite easily if state is not maintained by your services or their hosts.[1]

Practice Defense in Depth

Adopt a zero-trust policy by not assuming trust within a given subnet or zone and ensuring that authentication

is applied between each component. Doing so will minimize security threats from inside your network security boundaries. In a cloud native architecture, you should extend this approach to include parameters such as script injection and rate limiting. In this way each component in a design seeks to protect itself from other components. This makes the architecture resilient and its services easier to deploy within the cloud environment.

Design for Automation

Automate everything and, to make automation pay dividends, monitor and measure every system or service you automate. The key therefore is to choose the correct metrics in accordance with the guidance provided in chapter 9. Because automated processes can monitor, repair, scale and deploy your system faster than humans, you should measure, automate, monitor, and assess your systems continuously. Some of the elements that you can automate include the following.

1. Cloud fabric, or infrastructure: Automate the creation and management of cloud resources using such tools as Cloud Formation (AWS), Azure Automation (Azure), Cloud Deployment Manager (GCP), Terraform (a tool that is agnostic to cloud providers), Helm charts (for Kubernetes), Puppet, Chef, or Ansible.

Adopt a zero-trust policy by not assuming trust within a given subnet or zone and ensuring that authentication is applied between each component.

2. CI/CD pipeline: The continuous integration/continuous delivery pipeline can be automated using tools such as GitHub Actions, Jenkins, Google Cloud Build, Spinnaker, or Azure DevOps.

3. Scalability: Use autoscaling groups to scale your computational resources or Kubernetes to scale your microservices.

4. Monitoring: Monitor user activity and system resources to assess what services are most in demand, what resources have the highest utilization, and whether there are any security breaches. Should your system automatically take action after analyzing the monitoring metrics, it is said to incorporate observability. So, for example, you could monitor CPU and RAM utilization, and should their utilization reach a certain level, then it would trigger an automated action to spin up another VM or container.

Well-Architected Frameworks

The general architecture principles described in the previous section form a common thread that is supported by the well-architected frameworks (WFs) espoused by all three major cloud providers.[2] Generally these are similar

in scope and practice, so we shall consider Amazon's AWS framework and its principles as a proxy.

The following terminology used by AWS to describe the WAF has been extended in terms of service reusability:

Component: This is the AWS resource, its configuration, and the software that, combined, deliver a service. A component is a single unit that is decoupled from other components. It is equivalent to the cloud cell paradigm discussed in chapter 2.

Workload: Workload refers to a set of components that together create a solution to a business problem. Workload is usually what business and technology leaders communicate about.

Architecture: Architecture denotes a high-level design with a number of components as part of a given workload. Architecture diagrams show how components work together within the workload to deliver a solution.

Service catalog: To aid reusability of workloads and components, the service catalog catalogs all the services used within a workload or component. Doing so enables architects and engineers to reuse components and workloads that have already been built, either by others or themselves.

Life cycle: Life cycle is a term that can be applied to a solution, a service, or a product. It describes how that solution (service, product) evolves from requirements gathering, design, implementation, testing, through to production.

Milestones: These mark key changes in the architecture as it evolves through the product or service lifecycle.

Technology portfolio: The technology portfolio is a collection of workloads that are required for the business to operate using technology.

As table 6 depicts, the five pillars of the WF are (1) **c**ost optimization, (2) **o**perational excellence, (3) **p**erformance **e**fficiency, (4) **r**eliability, and (5) **s**ecurity—or COPERS as a mnemonic.

With this general idea of best practice in mind, let us look at a reference architecture for a web application using

Table 6 Five Pillars of the AWS Well-Architected Framework

Name	Description
Cost optimization	Run systems that deliver business value at the lowest price point.
Operational excellence	Continuously improve the efficiency of workloads by developing, managing, running, and monitoring them.
Performance efficiency	Use cloud resources efficiently to meet system requirements and to maintain that efficiency as requirements change and technologies evolve.
Reliability	Run a workload to perform its intended function correctly and consistently across its life cycle.
Security	Protect data, cloud resources, and assets in conformance with your overall security policy.

cloud services, as well as using cloud native, event-driven services.

Reference Architecture for a Web Application

A web application traditionally implements the model-view-controller (MVC) architecture as it separates the application into three component layers:

Model layer: This layer manages the application's data. It sends information from a database to the controller in response to user requests.

View layer: This is the presentation layer as it interfaces with users. Data received from the model layer or the controller layer are presented to the user as a response. User requests received are sent over to the controller layer.

Controller layer: This layer is responsible for the business logic. It implements the algorithms and uses data from the model layer to provide a response to user requests via the view layer.

This MVC pattern effectively decouples the data, logic, and presentation layers so that changes in one layer do not affect any of the others. An advantage of this approach is that it speeds development because separate teams can develop and deploy the layers in parallel. Another advantage is that separation of the layers makes the architecture

more secure and adaptable to implement the zero-trust model's principles.

Figure 22 shows the reference design for a web application implemented using AWS. It makes use of two different Availability Zones in the same region to provide redundancy during a failover scenario. Amazon's CloudFront is used to scale traffic at the edge and to provide a low latency response to users. (Route 53 for DNS in conjunction with CDN for edge caching of content could have been used instead of CloudFront.) The Elastic Load Balancer, ELB, then routes traffic to web servers located in both Availability Zones, while static content is provided from the S3 bucket. A Lambda Function gets fired when a certain threshold is reached for data stored on the S3 bucket in order to send a notification via SNS to increase the size of the S3 bucket. This optimizes storage costs related to S3. The web server is hosted on EC2 instances (i.e., VM instances), which form an Auto Scaling Group to provide scalability when serving dynamic web content. The web servers are located in public subnets and a NAT gateway is used to traverse through to the private subnets where the app servers are located. These app servers form the application or the business logic layer and respond to HTTP requests from users. The data layer consists of RDS databases in a separate private network in conformance with the principles of the zero-trust model. The databases

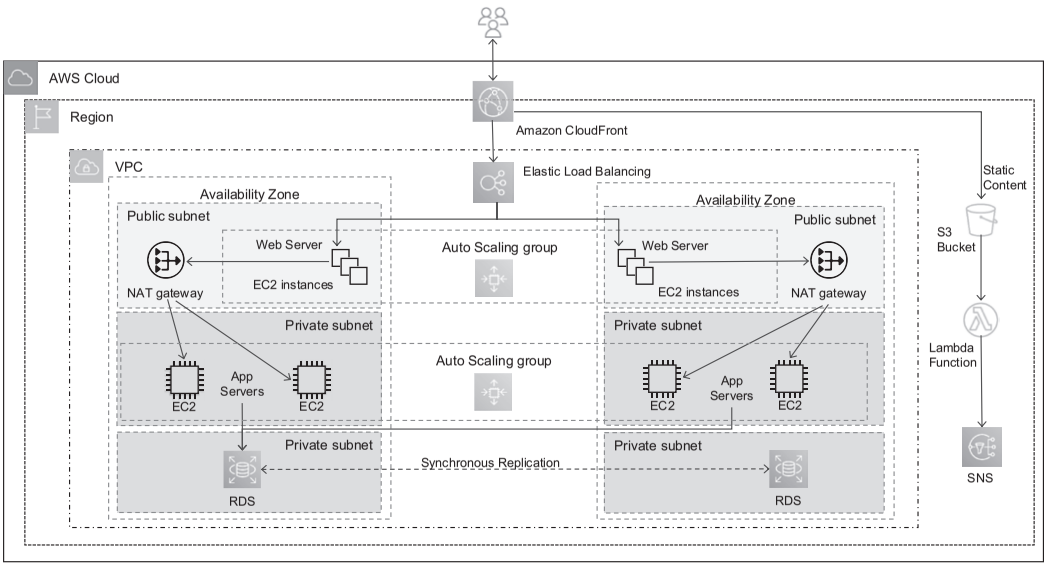


Figure 22 Web application reference design with AWS.

in the two Availability Zones are replicated so that either can provide data to the app servers.

Let us now translate the design of figure 22 to an Azure one. Instead of the AWS Cloud account, we would have an Azure Subscription, AWS regions would map to Azure regions, and the VPC would map to Azure Virtual Networks (VNets). The EC2 instances would map to VM instances and Amazon RDS would map to Azure DB. The Scaling groups used in AWS are very similar to Azure's VM Scale Sets.

On Azure by default, a VM on a VNet can connect to any other VM on the same Vnet even if the VMs are on different subnets. This is because Azure configures the routing tables to route traffic to every resource within a VNet by default. Therefore, to have separation between subnets so that you can mimic the private/public subnet separation of AWS, you would need to define network Security Group rules for each subnet to implement differentiation and isolation between the public and private subnets.

Event-Driven Serverless Reference Architecture

An event-driven architecture is one in which the computation is triggered by the event of a request. So it is asynchronous in nature, and the components in the architecture rely on a publish-subscribe (pub-sub) messaging

paradigm to make them decoupled, and hence scalable and interoperable. Also, if you ensure that each component has its own security perimeter, then you can implement a zero-trust security model whereby each request, service, or user can access only the data or resource that is necessary for its legitimate purpose.

Serverless is defined as a cloud native development model for applications that "allows developers to build and run applications without having to manage servers."[3] It is a model that does have servers, but they are abstracted away so that the developers may concentrate on building microservices. In a nutshell, a serverless compute node is one for which you do not pay for any idle time as you would with a VM or EC2 instance.

So an event-driven, serverless application is one that is composed of small, easily deployable, loosely coupled, independently scalable, serverless components using an event bus (usually using a pub-sub model) for communication between the components to implement loose coupling. The pub-sub messages are sometimes referred to as an event stream, and best practice is for each component to have its own event stream (i.e., pub-sub topic) for communication purposes. In the past, APIs were used for internal communication between the microservice components, but this practice is deprecated in favor of using events.

Figure 23 shows an event-driven serverless reference architecture for a web application implemented using

In a nutshell, a serverless compute node is one for which you do not pay for any idle time as you would with a VM or EC2 instance.

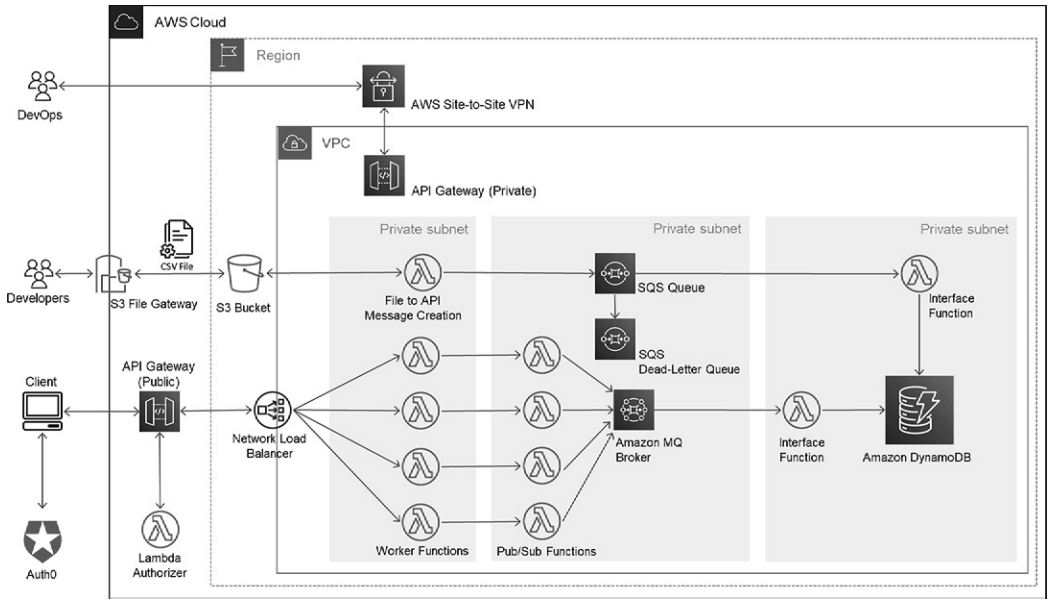


Figure 23 Event-driven serverless web application.

AWS. A VPC is defined within an AWS region for the application. Of course, in a real-life production environment, you will need to use a duplicate VPC in another region, or perhaps even on another AWS account, to implement redundancy for failure recovery. In any case, three distinct actors (and their respective paths of ingress) are shown: DevOps, developers, and a machine client that consumes services via an API.

The DevOps team uses a site-to-site VPN connection to access the AWS resources and the developers use a file share using an S3 file gateway to upload CSV files to an S3 bucket. The CSV file contains the data for the API end-points that need to be used by the DynamoDB database. A lambda function is triggered whenever a file arrives on the S3 bucket, which then creates the API messages to submit to the SQS queue. Any error messages or undeliverable messages are sent to the dead-letter queue for refactoring and troubleshooting. When the SQS queue sends a message, it is picked up by the interface function, which updates the database with the message data. This is the producer path for creating API end-point data.

The data path for API consumers starts with the client on the lower left side of figure 23 that uses an HTTP- or HTTPS-based API. The client is identified and authenticated by Auth0 in order to access the API; on the AWS side, the API gateway has a lambda authorizer function configured that ensures only valid and identified users or

machines can consume the API services. The network load balancer balances HTTP(S) traffic and routes it to worker lambda functions. Notice that we are using a set of two functions instead of just one as we wish to have microservices that do specific tasks. The worker functions implement the business logic for the API end-points and the pub-sub functions interface with Amazon MQ Broker. We could just as well have used SQS instead of the MQ Broker. The MQ Broker then uses an interface lambda function to interact with the DynamoDB database.

Notice that we are using layers in conjunction with the lambda functions to implement zero trust. In essence, three subnets are used to separate the components in their respective layers:

1. the business logic layer, where the worker nodes are located,

2. the MOM (message-oriented middleware) layer, which is effectively the data access layer, where Amazon MQ Broker resides together with its interface nodes, and

3. the data layer, where DynamoDB together with its supporting lambda functions are.

This provides network segregation at no additional cost and is aligned with best practice. Additionally, we use access controls at the lambda function, or at the resource

level, whereby each resource requires role-based access control for it to be invoked or managed, even if the network is breached at the subnet level. This effectively implements best practices defined by the zero trust model.

The reference architecture of figure 23 can be translated to similar architectures for GCP and Azure by referring to table 5 of the previous chapter. For example, you would use Azure functions for Azure and Cloud functions for Google instead of the AWS lambda functions. Similarly, you would use Azure's Cosmos DB or Google's Cloud Firestore instead of Amazon's DynamoDB. For the event bus, you would use Service Bus (Azure) or Cloud Pub/Sub (Google) in place of Amazon MQ Broker. The concepts of the reference architecture provided are therefore transferrable to the other public cloud providers.

13

FUTURE OUTLOOK

In this chapter we look at the future and try to predict what might develop over the horizon. It is with mixed feelings of joy and pride, on the one hand, and sadness on the other that I write this chapter—joy and pride because many of the developments predicted in the first edition of this book have actually come to pass, sadness because it is a shortened—though revised—version of the first edition as a result.

Cloud computing, being an enabling technology for automation and abstraction, is in a unique position to effect paradigm shifts related to your work, society, and life. This chapter discusses some emerging technologies and trends that are related to cloud computing because they are the catalysts of change for the future technology landscape. I discuss these trends and make some extrapolations to what could happen in future. Of course, as with

Cloud computing, being an enabling technology for automation and abstraction, is in a unique position to effect paradigm shifts related to your work, society, and life.

predictions in general, not all may come to pass. Nevertheless, it is always good to know the art of the possible and what doors cloud computing may open for us in shaping the future.

Internet of Things and Services

In your home, you will find many appliances that contain microprocessors and microcontrollers: ovens, washing machines, televisions, refrigerators, and even some rubbish bins. This list is set to get longer owing to the very low prices of microcontrollers. Even disposable items such as light bulbs now include microcontrollers. When you connect these appliances to the internet, you get connected devices, and the network they use is called the "internet of things," or IoT.

Why should you want to connect such devices to the internet? Let us consider your rubbish bin as an example. What if it became full and its sensor could let the facilities department in your locality know that this is the case. Facilities could then empty the bins when needed rather than send out a van on a regular basis, or perhaps it would send out the van when most bins in your locality were full—thus they would aggregate the data received from each bin. Suppose further that your bin could predict when it would become full (it can do this simply by

extrapolating the increase in weight that it senses daily) and then sent out an email or an API call to let the facilities department know a few days in advance. This way the department could plan ahead and ensure that an optimum refuse collection route was in place. This would save fuel and time for your locality. As a result, your local taxes may be reduced (or perhaps not be increased) so that you may benefit too. Something similar is already happening with your printers at your workplace. The printers monitor the paper, ink, and toner—known as consumable resources—and send out an email to a central desk when these materials are about to run out. The printer's replacement ink and toner are then sent to your facilities department for replacement. And your company does not get charged for these services! This is because the printer is "owned" by the printer company and leased to your work company. Your company simply pays the printer company a monthly amount that is based on the number of pages printed for that month. Thus the IoT printer has a cloud-based utility price model. The monitoring of several printers, the sending out of bills based on pages printed per month, and the automated ordering of supplies such as ink can be thought of as a business service, and hence be categorized as BPaaS.

When such a service is automated in the cloud and is there purely to support IoT devices, then you have an "internet of services," or IoS.

An example of IoS is when you have many streetlights in a particular locality connected to the internet. They are monitored for the replacement of light bulbs in an automated manner, thus making it unnecessary for a person to go out regularly on an inspection tour. Additionally, the streetlights could have multiple uses: they could collect weather-related information on temperature, condensation, and atmospheric pressure, or measure the flow of traffic to assess the convenience of certain roads. In any case, monitoring streetlights for light bulb replacement would be an IoS function for the various IoT-connected lamps. As you can tell from these examples, IoT and IoS are related and so should be considered together.

But not all applications are benign. For instance, the cameras fitted to cars could be connected to the internet to provide a third party with a view of your surroundings, the car's GPS could send your location, and so on. Technology can therefore be used to proliferate the powers of a police state, making Aldous Huxley's Brave New World that much more real.

Cloud of Things and Services (CoTS)

What have IoT and IoS got to do with cloud computing, you may well ask. The processing that takes place in monitoring the IoT devices and executing the services as part

of a workflow to support those IoT devices will usually be in the cloud. This is where the IoS-type services will reside. So the IoT devices and their related IoS-type services will have their own cloud. Such a cloud can be a private, public, community, or hybrid cloud, depending on the application and your need. I have coined the term "cloud of things and services," or CoTS, to refer to such a specialist cloud.

Even though there may be few, if any, CoTS clouds at present, they will become ubiquitous in the future with the proliferation of IoT devices and their IoS services. The key premise of CoTS is automation. This supports the dual premises of IoT and IoS: convenience, service, and predictive analytics when and where needed. The "where" is important in this regard. Considering our example of the streetlight, you will know the exact lamp (and its location) that needs a replacement light bulb. For nonstationary devices, geopositioning using GPS may be used to obtain the exact geolocation of the IoT device in terms of longitude, latitude, and elevation.[1] The ubiquitous characteristic of cloud computing has considerable synergy with IoT devices because of the need to monitor them and provide services related to them wherever they may be.

Some broad areas where CoTS-based services can be provided are:

IoT devices and their related IoS-type services will have their own cloud. . . . I have coined the term “cloud of things and services,” or CoTS, to refer to such a specialist cloud.

- Smart home: Nest Protect, a smoke detector and fire alarm system, is an example of connected devices in the home or office.

- Wearables: Google Glass and Apple Watch are good examples of these connected devices. They provide a service not only for communication but also for sensors to monitor your health or environment.

- Smart city: Traffic management, water distribution, waste management, urban security, environmental monitoring, pedestrian congestion management, and street lighting are some example use cases where a CoTS service could connect IoT devices.

- Smart grids: Smart metering augmented with IoT services for information about electricity consumption to improve the efficiency, reliability, and trading of electricity is a clear example; this configuration can be extended by connecting smart batteries to the electricity grid so that stored electricity that has been generated locally can be sold on the grid.

- Retail: Proximity-based advertising, smart wallets, and near-field communication-related purchasing in the retail sector are examples.[2]

This list is just the tip of the iceberg. It can easily be extended to cover IoT services in areas such as banking,

manufacturing, health care, farming, and transport as CoTS applications.

Personal Clouds

If you can have a cloud for things, then why not have a cloud for people? A personal cloud is one that you own and use for your personal needs. Examples are plentiful: (1) to store documents (your financial statements, driver's license, etc.); (2) to store your electronic wallet (e.g., your financial wallet or your health wallet), (3) to store your shopping basket for electronic shopping; (4) to store your music and videos. All these are available services today but are not aggregated in one place. What if they all could be found in one place, such as your personal cloud?

Consider the use case of your health wallet. With the onset of IoT and wearable devices, it is becoming common for people to measure various health metrics such as blood pressure, heart rate, and weight so that the data may be stored in a health wallet. Microsoft, Google, and Apple provide, or will soon provide, such wallets. If you automate the monitoring of your health metrics using IoT and wearables, then you have an interesting intersection between the personal cloud and CoTS. Your health wallet in your personal cloud connects with IoT devices and obtains your health data, and so is a CoTS as well. The CoTS would have

a service (remember IoS) that would alert you, your doctor, or your family if the health metrics reached a certain critical level. But you can take this service one step further by using predictive analytics. What if you received an alert that told you of an impending deterioration in your health as a factor of a combination of those metrics? You could then be proactive and ensure that your health improved by addressing the factors that produced the alert.

A Cloud Service Exchange

In this section I discuss an innovative idea that I have developed for purchasing cloud services using a cloud service exchange. Just as stocks are exchanged for money in a stock market, a cloud exchange could transact cloud services at a given price in real time. This would enable cloud services to be bought and sold just as commodities are. This is obviously easier to implement for services that are at a lower abstraction layer, such IaaS and PaaS, compared to higher-level ones such as BPaaS. However, cloud native services would be just as easy to exchange even though they offer a higher level of abstraction since they are consumable using API calls. A major component of such an exchange would be the interoperability of the cloud services. Thus, one service should be replaceable by another seamlessly and instantaneously by way of the exchange. Service producers and

consumers would then share the exchange platform to effect a deal, and each service would be classified in terms of the cloud patterns discussed in this book. The outcome would be a fair price for the cloud service user since the price paid would depend on the supply and demand parameters of a service. A real-time mechanism for price and service discovery would need to be implemented so that an instantaneous trade could be realized by matching the right service to its prospective buyer. This is something that the future could bring to the world of IT because cloud computing is the natural evolutionary step in the increasing commoditization of IT technologies and services.

Conclusion

With the increased use of IoT devices, cloud computing will mushroom as each device will connect to a cloud service. And the further adaption of 5G and 6G technologies is bound to accelerate the use of IoT and CoTS. Beyond devices, there is no reason why every person, house, and car could not have their own separate cloud to store health and status information, extrapolate future outcomes, and provide appropriate situational information. The challenge will be in creating appropriate policies and governance frameworks regarding the use and availability of information stored in such clouds.

ACKNOWLEDGMENTS

My dear wife, Leena, has been the force behind this book. Without her encouragement, I do not think it would have seen the light of day. Thanks, Leena, for your constant nagging!

Elizabeth P. Swayze, Senior Editor for Computer Science at the MIT Press, was instrumental in getting this second edition published. Her patience and support ensured that it got approved for publication for the Essential Knowledge series. Her assistant, Matthew Valades, helped ensure that the publishing process ran smoothly through its course, which is much appreciated. My production editor, Deborah Cantor-Adams, helped make the manuscript ready for publication with much enthusiasm and commitment. And my thanks to the rest of the MIT Press team: designer Emily Gutheinz, art coordinator Sean Reilly, production coordinator Tori Bodozian, and publicist Oscar Sarkes. You are true heroes!

APPENDIX A

Common Security Terms

This appendix supplies the meaning of some commonly used terms in IT security. Knowing these terms and what they mean will equip you with the knowledge to ensure that your systems and data are secure when using cloud services.

Access control	Ensures that access to services is granted only to entitled users.
Backdoor	Code installed to enable a user (e.g., an attacker) easier access to a system by bypassing security mechanisms that are in place.
Biometrics	The use of a user's physical characteristics to determine access to a system or service.
Business continuity plan	A plan that takes measures to ensure that a business remains operational in the face of a disaster, an emergency, a security breach, or an attack.
Certificate	A small data file that binds a cryptographic key to an organization's details; it certifies the authenticity and identity of the issuer. Also known as a digital certificate or an SSL/TLS certificate.
Checksum	A value computed using the data that are stored or transmitted; it is used to verify the received data. A checksum is intended to verify the integrity of data and identify data-transmission errors, while a hash (see *Hash function*) is designed to create a unique digital fingerprint of the data. A checksum protects against accidental changes, whereas a hash protects against a purposeful attacker.

Confidentiality	Ensures that information is disclosed only to those who are authorized to view it.
Custodian	The user or application that is currently using or manipulating the data and so is temporarily taking responsibility for it.
Data integrity	Integrity of the data in terms of its veracity, completeness, and immutable state at the point of use.
Decryption	The transformation of an encrypted message into its original plaintext form.
Demilitarized zone	Abbreviated as DMZ, this is a perimeter network area that sits between an organization's internal network and an external network, usually the internet. It forms part of a layered security model in which network segmentation is used to ensure the transit of data between a secure layer and an insecure one.
Digest authentication	A process that ensures that a client application or user can provide proof that they have a password to use the system or service.
Encryption	Cryptographic transformation of data (called "plaintext") into a form (called "cipher text") that conceals the data's original meaning to prevent it from being known or used.
Escrow passwords	Passwords that are written down and stored in a secure location (e.g., a safe or an encrypted USB drive); they are used by emergency personnel when privileged personnel are unavailable.
False negative	A false negative is an error that classifies a test result incorrectly indicating the absence of a condition when it is present (e.g., result stating the absence of a security vulnerability when one is actually present).
False positive	A false positive is an error that classifies incorrectly the presence of a condition (e.g., presence of a security vulnerability when it is not present).

False reject	Failure of an authentication system to recognize a valid user.
Firewall	A logical or physical discontinuity in a network to prevent unauthorized access to data or resources.
Fragmentation	Storing a data file in several "chunks" or fragments rather than in a single contiguous sequence of bits in one place on the storage medium.
Fuzzing	Regression testing that generates out-of-specification input for an application in order to find its security vulnerabilities.
Hash function	An algorithm that maps data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or hashes; the mapped value is used as an index in the hash table. An example of a hash function use is in scrambling passwords before storing them in a database so that they cannot be understood or copied by a human.
Identity and access management	See *Identity management*.
Identity management	The process that verifies a user's identity and their level of access to a particular resource, service, or application. Under the auspices of identity and access management (IAM), authentication and access control play a vital role in securing data.
Integrity star property	Approach whereby a user cannot read data of a lower integrity level than their own.
Intrusion detection and prevention	Identifies and prevents security breaches, including intrusions (attacks from outside the organization) and misuse (attacks from within the organization). IDP is usually performed at the inside and outside boundaries of a DMZ.

Lattice techniques	Use of security designations to determine access to different types of information.
Malware	General term used to denote viruses, worms, trojans, and any form of unauthorized software. Malware is a composite term derived from "*mal*icious soft*ware*."
Masquerade attack	A type of attack in which an illegitimate user or application poses as a legitimate one.
Multifactor authentication (MFA)	An authentication method that requires the user to provide two or more verification factors (or evidence) to gain access to a resource; the factors are based on knowledge (something only the user knows), possession (something only the user has, such as a device), and inherence (something related to the user, such as their fingerprint). MFA is a core component of a strong identity and access management (IAM) policy.
Nonrepudiation	System to ascertain that a specific user and only that specific user sent a message and that the message has not been modified.
Penetration	Gaining unauthorized access to sensitive data by circumventing a system's protections.
Pharming	Technique whereby a user's session is redirected to a masquerading website or application.
Phishing	Derived from "fishing," a practice adopted by attackers to obtain sensitive information from a human by fraudulent means. These can take the form of emails, website links, texts, or phone calls that pretend to be from authentic sources. Once the sensitive information is obtained, it can be used in a number of ways: for identity theft, money transfer, deployment of malicious software on the victim's system, or blackmail.

Plaintext	Data before they have been encrypted into ciphertext or data that have been decrypted.
Proprietary information	Information that belongs to an organization and gives it the ability to do business. Such information includes customer lists, technical data, costs, and trade secrets.
Risk	Security risk can be computed as the product of the threat level and vulnerability level to quantify the likelihood of a security breach.
Role-based access control	Shortened as RBAC; an approach to restricting access to resources or services to authorized users that have defined roles. The roles could be functional (such as sales, accounting, or marketing) or technical (e.g., administrative, developers, architects) and they define what resources are accessible to users assigned to the particular role.
Threat assessment	The identification of types of threats that an organization might be exposed to.
Trojans or implants	Malware disguised as benign programs or applications.
Two-factor authentication	Also denoted as 2FA; a special case of multifactor authentication wherein two factors are used for authenticating the identity of a user. See *Multifactor authentication*.
User	A person, organization, entity, application, or automated process that accesses a system, whether authorized to do so or not.
Virus	A hidden, self-replicating section of computer software that usually contains malicious logic and propagates by infecting another program or application.

Wallet	An app that can store, in a single place, financial information (reward, debit or credit cards), health information, keys, or documents (tickets, IDs, boarding passes) for the consumption of services in a convenient manner.
Worm	A program that can run independently and propagate a complete working version of itself onto other computers on a network; it may consume computer resources destructively.
Zombie	A computer that has been compromised by malware.

GLOSSARY

Aggregation
Aggregation denotes an "is-a" relationship. For example, a forest is an aggregation of trees. The underlying assumption here is that all the trees are similar in terms of their attributes or properties. Should the trees be of varying types, with different features that you need to use, then aggregation would no longer apply. Instead, the relationship would be said to be composition.

Application programming interface
An application programming interface (API) is a software interface that allows computer programs to communicate with each other. APIs work by sending requests for information and receiving a response; often the HTTP protocol is used in conjunction with its verbs PUT, POST, GET, PATCH, and DELETE to perform CRUD (create-replace-update-delete) operations. An API end-point is one end of the communication channel between the software programs. A software development kit, or SDK, facilitates the creation and usage of an API.

Archive
An archive houses data that require long-term preservation for regulatory compliance, historical, or evidentiary reasons. It is accessed less often than a backup copy but requires speedy lookup and retrieval of data. See also *Cold storage*.

Backup
A backup is a copy of your current data (which may be programs, source code, VM images or their definition, YAML manifest files for Kubernetes, etc.) that you use to restore a system to its working state should a failure, data corruption, or disaster occur.

BPaaS
Business Process-as-a-Service; see *Service model*.

Cache
A cache stores data on a temporary basis to help websites, browsers, and apps be more responsive. A cache hit occurs when the requested data can be found in the cache, while a cache miss occurs when the data cannot be found. To increase cache hits, data that are most often requested are stored in the cache. Two main approaches are used to make caches more efficient, spatial locality and temporal locality. In the former, the cache (i.e., the requested data) is stored physically close to the data source; in the latter, the requested data have been requested recently.

Canary deployment
Canary deployment is a term for the practice of staging a deployment whereby a subset of users receive a release, followed by a broader group once any initial teething problems have been ironed out.

Capital expenditure
Capital expenditure is the up-front expense borne by you or a third party to create a product or service. CapEx is also referred to as nonrecurring or implementation costs.

Change management
Change management addresses the addition, modification, or removal of services or service components while reducing incidents, disruption and rework.

CIA triad
The CIA triad—confidentiality, integrity, and availability—describes a security model used to find vulnerabilities and define secure solutions.

Cloud bursting
Cloud bursting describes the situation in which your cloud service or application instantaneously jumps (bursts out) to use the resources of another cloud. It might do so, for example, to meet an inordinate increase in demand or for business continuity reasons.

Cloud cell
The cloud cell is a novel concept developed in this book to denote a discrete cloud service that performs a single business or technical function; a number of cloud cells could be used together to provide a conglomerate unique cloud service. Cloud cells may have relationships with other cells, such as composition, encapsulation, or federation.

Cloud native
Cloud native describes an approach to building applications that exploits features such as scalability, elasticity, resiliency, and flexibility, which cloud computing provides.

Cloud service
A cloud service is the implementation of a business process—provided through a set of related functional components and resources—that provides business value to the end consumer of the service. It can be categorized in terms of five service models.

Cold storage
Generally, cold storage of data is an archival tier that retains data in a tape-based storage system for long-term retention. This is akin to AWS Glacier. Azure offers three tiers:

Hot: For storing data that are accessed frequently.

Cool: For storing data that are accessed infrequently and are stored for at least thirty days.

Archive: For storing data that are rarely accessed and are intended to be stored for at least 180 days.

Community cloud
A shared cloud computing service that is targeted to a limited group, the members of which share a similar need or use case. These can be diverse and can vary from business needs (compliance requirements) to individual ones (a hobby or a pension fund) to a societal one (a state or nation).

Component
A component is an atomic unit comprising the computation engine, its configuration and software that, combined, delivers a service; a component is an example of a cloud cell.

Composition
A composition denotes a "has-a" relationship. Thus, a car is a composition of a number of distinct units, such as steering wheel, tires, hood, doors, engine, and so on. Each unit that contributes to creating the composed object has its own distinct attributes or properties, and is referred to as an object.

Container
A container is a self-contained unit of code, its libraries and dependencies, and a lightweight operating system that runs as an application in a common way on a host machine.

Content delivery network
A content delivery network (CDN) is a geographically distributed group of servers that work together to deliver web content (e.g., HTML pages, JavaScript files, style sheets, images, videos) speedily to improve responsiveness.

Control bus
A control bus is a communication channel that enables microservices or software components to send control commands or messages to each other in a decoupled manner.

Currency
Currency describes how technically current your computational resources are. Currency is usually considered in terms of the version number of the software and the generation of the hardware. It is expressed in terms of N, N-1, N-2; the latter being 2 versions old and the former being current.

Delivery model
A deliver model represents the efficient combination of end-to-end processes, methods, and quality procedures together with the right skills on a global basis that will enable your IT department to meet its business needs.

Domain name server
A domain name server (or service) (DNS) provides a mapping of a domain name to its IP address, similar to a telephone directory.

Dynamic host configuration protocol
Dynamic host configuration protocol (DHCP) is a network service that automatically assigns a dynamic IP address to each host, or server, on a network.

Elasticity
Elasticity denotes the capacity of a cloud service to scale out horizontally when demand is high and scale in when demand is low.

Encapsulation
Encapsulation may refer either to the composition or to the aggregation of cloud services. The concept is derived from object-oriented programming and design wherein one object can encapsulate another.

Event bus
An event bus is a communication channel that enables microservices or software components to inform each other, in a decoupled manner, of events that take place so that workflow may be orchestrated or the status of a job monitored.

Federation
Federation denotes the creation of a composite cloud service through creating an alliance with other, disparate clouds or cloud cells. Although the other clouds or cloud cells would usually be provided by different cloud service providers, they may also be a mixture of your own cloud cells or a mixture of a third party's cloud cells.

Hybrid cloud
A hybrid cloud is a combination of private and public cloud services that is usually based on a policy-driven, coordinated approach to the consumption and management of those cloud services.

IaaS
Infrastructure-as-a-Service; see *Service model*.

Identity and access management
Identity and access management (IAM) denotes a set of policies, processes, and technologies used to manage digital identities and control user access to cloud services.

INaaS
Information-as-a-service; see *Service model*.

Information Technology Infrastructure Library
The Information Technology Infrastructure Library (ITIL) is a framework that provides guidance on the full life cycle of defining, developing, managing, delivering, and improving IT services from an operational perspective rather than a delivery perspective. ITIL is descriptive rather than prescriptive as it provides best practice advice on IT service management.

Interoperability
Interoperability denotes the capability to use the same or similar cloud services offered by different cloud service providers in such a manner that all services work seamlessly together. This goal relies on interoperability in terms of billing, management, reporting, data management, and application or process functions.

Load balancer
A load balancer enables the even distribution of the workload over two or more resources with the aim of optimizing computational or response times.

Maturity model
A cloud maturity model gauges your maturity in the use and management of cloud computing services and points out areas in need of improvement. It allows your organization to have its methods and processes assessed according to best practices against a clear set of benchmarks.

Microservice
Microservices are an architectural and organizational approach using service orientation where software is composed of small independent services, which offer a single distinct service; they communicate over decoupled channels. A microservice implementation is an example of a cloud cell. See *Cloud cell*.

Multifactor authentication
Multifactor authentication (MFA) requires two or more factors of authentication (e.g., code sent over email or phone, biometric information) in addition to passwords for access to be granted.

Multitenancy
Multitenancy denotes the pooling of resources, virtual or physical, such as software, storage, or virtual machines, to provide a shared common service to each user of the cloud service.

Operational level agreements
An operational level agreement is an internal agreement between the supplier and internal consumer (or resupplier to the end consumer) within an organization that spells out the terms and condition for providing a service.

Operating expenditure
Operating expenditure (OpEx) comprises the ongoing operational costs that you incur when consuming a cloud service. It is also referred to as run-time or recurring costs.

PaaS
Platform-as-a-Service; see *Service model*.

Pattern
A pattern represents the best practice design and implementation of a solution to a problem that others face consistently in engineering.

Private cloud
A private cloud is a computing cloud that is used by only one entity; the entity can be an organization, person, or thing. In a private cloud the cloud computing resources are shared only by cloud services meant for that entity, as demand dictates.

Public cloud
A public cloud is a form of computing cloud that is shared by a number of entities; an entity can be an organization, person, or thing. In a public cloud, the cloud computing resources are pooled together for the use of the entities as demand dictates.

Recovery point objective
Recovery point objective (RPO) denotes the interval of time during which data are lost in the event of a disaster or disruption before the organization's maximum tolerance of amount of lost data is breached.

Recovery time objective
Recovery time objective (RTO) is the duration of time needed to recover normal operation following a disaster or a disruptive event.

Release management
Release management encompasses the planning, design, development, building, testing, deploying, and releasing phases of a service or software product to end-users.

Roadmap
A roadmap is a future plan for upgrading and updating the cloud components.

Role-based access management
Role-based access management (RBAM) grants access to cloud resources on the basis of permissions attributed to a user role. Also sometimes referred to as role-based access control (RBAC).

SaaS
Software-as-a-Service; see *Service model*.

Serverless
A cloud-native development model for applications that allows developers to build and run applications without having to manage servers, which are abstracted away so that the developers may concentrate on building microservices. Examples are Lambda Functions (AWS), Cloud Functions (GCP), and Azure Functions.

Service
A service is a collection of IT systems, components, and resources that work together to provide value to the end user or consumer. See also *Cloud service*.

Service-level agreement
A service-level agreement (SLA) is a contract, usually written, between the service consumer and the service supplier that specifies how quickly the service will be delivered (when), its quality (what), and its scope (where and how much).

Service-level objective
A service-level objective (SLO) is a metric used by a service provider to measure its performance to ensure that it meets the SLA or the OLA.

Service model
A service model is the type of cloud service that you can use or create. The five service models of cloud computing are IaaS, PaaS, SaaS, INaaS, and BPaaS.

Thin client
Thin client may refer to a device, an application, or a service. A thin client device is one that does not have a service or application installed on it; you use the thin client device to access a remote application or service that may be hosted in the cloud. A thin client application or service is one that is hosted on a server (in traditional hosting) or in the cloud and can be accessed using a zero, or a thin or a fat client device.

Utility price model
The utility price model, also known as a pay-per-use model, charges for using cloud resources based on usage. The usage can be measured in terms of a single metric or a combination of metrics such as number of virtual CPU cores, amount of memory, and network traffic.

Web application firewall
A web application firewall (WAF) is a type of application firewall that filters, monitors, and blocks HTTP and HTTPS traffic to and from a web service based on the traffic content.

Zero client
A zero client is similar to a thin client device except that its operating system and browser are embedded in the hardware.

Zero trust
Zero trust designates an approach to building secure environments that rests on the principles of (1) assuming security has been or will be breached, (2) providing the lowest level of privilege to users necessary for them to perform their tasks, and (3) always authenticating and authorizing requests for services and data.

NOTES

Chapter 1

1. Peter Mell and Timothy Grance, "The NIST Definition of Cloud Computing" (draft), http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909616 (accessed November 12, 2015).
2. A thin client machine is a computer that runs from computing resources on a central server instead of from a local hard drive.
3. Michael Armbrust, Armando Fox, Rean Griffith, et al., "Above the Clouds: A Berkeley View of Cloud Computing" (University of California, Berkeley, February 10, 2009).

Chapter 3

1. I first introduced the concept of a cloud cell as the precursor of the microservice paradigm in an article published in the twenty-first volume of Microsoft's *Architecture Journal,* titled "Extending Service Boundaries to Infrastructure Resources."
2. General packet radio service (GPRS) is a wireless communication service that provides continuous connection to the internet.
3. The Gang of Four refers to the four authors—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides—of the book *Design Patterns: Elements of Reusable Object-Oriented Software*, which was published by Addison-Wesley in 1994. The book was a seminal work that has since become the authoritative guide on software patterns.

Chapter 4

1. See the website of the Cloud Native Computing Foundation at https://github.com/cncf/foundation/blob/master/charter.md (accessed June 22, 2021).
2. Immutable infrastructure describes servers (physical or virtual) that are never modified after they have been deployed. Mutable infrastructure describes servers that require regular maintenance and administration and are modifiable.
3. A root directory in Unix or Linux systems is the uppermost level of hierarchy in a directory structure. It should not be confused with a root user. A root user is a user account that has the highest level of privileges in the system.
4. A service bus enables interacting software applications to communicate with each other. The applications can be implemented with different technologies

and can be hosted on heterogeneous infrastructure. A service bus is used to decouple applications and services from each other.
5. Borg was first made public in a research paper published by Google authors. Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, et al., "Large-Scale Cluster Management at Google with Borg," in ACM, *Proceedings of the European Conference on Computer Systems*, 2015, https://research.google/pubs/pub43438 (accessed July 20, 2021).

Chapter 5

1. James Lewis and Martin Fowler, "Microservices: A Definition of This New Architectural Term," March 25, 2014, https://martinfowler.com/articles/microservices.html (accessed September 11, 2021).
2. An enterprise service bus (ESB) connects services in a service-oriented architecture. The ESB includes a lot of extra functionality beyond just providing a message bus, such as formatting of messaging, translating mesages from one format to another, and routing them to a service on the basis of the content of the message and the rules applied. It is the antithesis of a dumb pipe.
3. Transactional consistency, also referred to as hard consistency, offers up-to-date data to all participants without any data lag but at the cost of high latency time since it takes time to update the database so that all participants have a consistent view of it.
4. Eventual consistency is generally found in distributed systems and NoSQL databases, which are usually designed to be distributed. There is a lag before all participants get a consistent view of the database and so a read request can result in stale data. Eventually consistency is maintained, but at the cost of the data lag, while low latency responses are provided to read requests.
5. See the Chaos Monkey website at https://github.com/netflix/chaosmonkey (accessed September 12, 2021).

Chapter 6

1. Formally, a use case identifies, clarifies, and organizes a service's or a system's requirements in a methodical manner. The use case is made up of a set of possible sequences of interactions between a user and the service provided when related to a particular goal. Even the informal use you put a cloud service to or the requirement you have for it is considered a use case.

Chapter 8

1. Erika McCallister, Tim Grance, and Karen Scarfone, *Guide to Protecting the Confidentiality of Personally Identifiable Information (PII),* NIST Special Publication 800–122 (US Department of Commerce, April 2010).

2. Firmware refers to the software that resides within a device and provides its electronics with instructions to function as intended. Firmware is changed rarely, if at all, during a device's lifetime; indeed, some firmware is permanently installed nonerasable memory (referred to as nonvolatile memory) and cannot be changed after manufacture. VOIP stands for voice-over-internet-protocol. It sends voice traffic over the internet so that you may have a conversation using software such as FaceTime or Skype.
3. Murphy's law states that whatever can go wrong will go wrong eventually.

Chapter 9

1. Michael Nieles, Kelley L. Dempsey, and Victoria Y. Pillitteri, *An Introduction to Information Security,* NIST Special Publication 800–12, rev. 1. (US Department of Commerce, 2017).
2. Voice biometrics can be used for identification as well as verification. Identification determines an unknown speaker's identity. Authentication verifies whether a speaker's identity is correct. In either case, voice biometrics use the acoustical properties of a person's voice to identify or authenticate since those properties are unique to every individual.
3. Adaptive access has been defined by NIST as access control that uses an authorization policy that considers operational need, risk, and heuristics.
4. TCP (transmission control protocol) is a connection-oriented protocol that provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts that are linked over an IP network. UDP (user datagram protocol) is a connectionless protocol that sends messages (known as datagrams) over the IP network.
5. An IP network uses the internet protocol (IP) to deliver packets of data from a source host to a destination host based entirely on the IP addresses of the hosts that are contained in the packet headers. When TCP and UDP use IP internetworking, they are referred to as TCP/IP and UDP/IP internet protocols, respectively. NTP (network time protocol) is a networking protocol for clock synchronization between computer systems. NTP is one of the oldest protocols in use, having been operational since 1985. It is used to send and receive time stamps using UDP on port number 123. Examples include time delivery to personal computers by Apple and Microsoft by synchronizing the PCs to their NTP servers on time.apple.com and time.windows.com, respectively.
6. Cookies are stored in a text file used by the internet browser but are accessed by websites to provide persistence between your visits to the sites. A tracking cookie tracks the usage of your browser by recording your entries and sending the information to the cookie designer. Not all tracking cookies are malicious, although they can be used in such a manner.

Chapter 10

1. For details, see Michael Hammer and James Champy, *Reengineering the Corporation: A Manifesto for Business Revolution* (HarperBusiness, 2006).

Chapter 11

1. A list of AWS regions and a map of where they are can be found at https://aws.amazon.com/about-aws/global-infrastructure/regions_az (accessed June 18, 2022).

Chapter 12

1. Canary deployment is a way of reducing risk when introducing a new software or service version by first rolling it out to a subset of users (or in a sandboxed testing environment) for testing purposes before deploying it fully in production. The term comes from the idiom "canary in a coal mine," which alludes to coal miners taking a canary with them to assess the safety of the coal mine.
2. Well-architected frameworks are available from all three major public cloud providers: https://cloud.google.com/architecture/framework for GCP, https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html for AWS, and https://docs.microsoft.com/en-us/azure/architecture/framework for Azure (all accessed June 18, 2022).
3. Please see the RedHat website at https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless (accessed September 7, 2021).

Chapter 13

1. The Global Positioning System (GPS) is an American satellite-based system, owned by the US government, that provides location and time information anywhere on or near Earth as long as there is an unobstructed line of sight to four or more geopositioning satellites. Similar systems have been developed by Russia (GLONASS) and China (BeiDou) using a constellation of satellites. Indian (IRNSS) and European (Galileo) systems are also under development.
2. Near-field communication uses a technology that detects and identifies you, when you are in its proximity, in order to initiate an instantaneous payment.

FURTHER READING

Bellemare, Adam. *Building Event-Driven Microservices: Leveraging Organizational Data at Scale*. O'Reilly, 2020.

Brynjolfsson, Erik, and Andrew McAfee. *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. New York: Norton, 2014.

Culkin, John, and Zazon, Mike. *AWS Cookbook: Recipes for Success on AWS*. O'Reilly, 2021.

Denzil, Watson. *Corporate Finance: Principles and Practice*, 8th ed. Pearson, 2019.

Erl, Thomas. *Cloud Computing: Concepts, Technology and Architecture*. Englewood Cliffs, NJ: Prentice Hall, 2013.

Eyskens, Stephane, and Ed Price. *The Azure Cloud Native Architecture Mapbook: Explore Microsoft Cloud's Infrastructure, Application, Data, and Security Architecture*. Pakt Publishing, 2021.

Kim, Gene, and Jez Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. IT Revolution Press, 2021.

Millard, Christopher. *Cloud Computing Law*. Oxford University Press, 2021.

Newman, Sam. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly: 2021.

Perry, Lea. *IoT and Edge Computing for Architects: Implementing Edge and IoT Systems from Sensors to Clouds with Communication Systems, Analytics, and Security*, 2nd ed. Pakt Publishing, 2020.

INDEX

NAYAN RUPARELIA has over thirty years of experience in IT, of which ten years have been as a CTO at organizations ranging from startups to multinationals. He has demonstrable experience in leading transformation projects that make companies more agile and efficient through the application of technology.