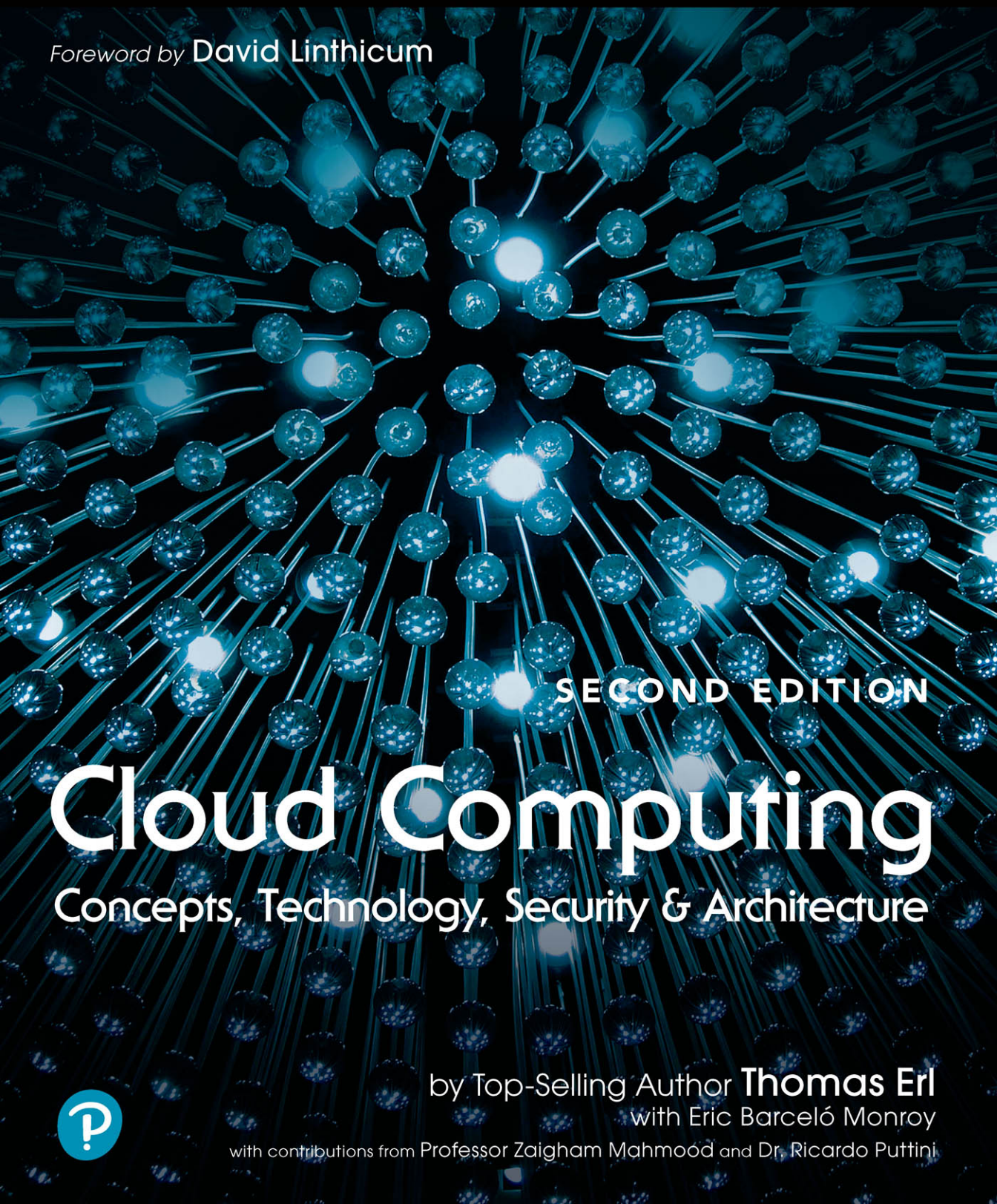
THE PEARSON DIGITAL ENTERPRISE SERIES FROM THOMAS ERL



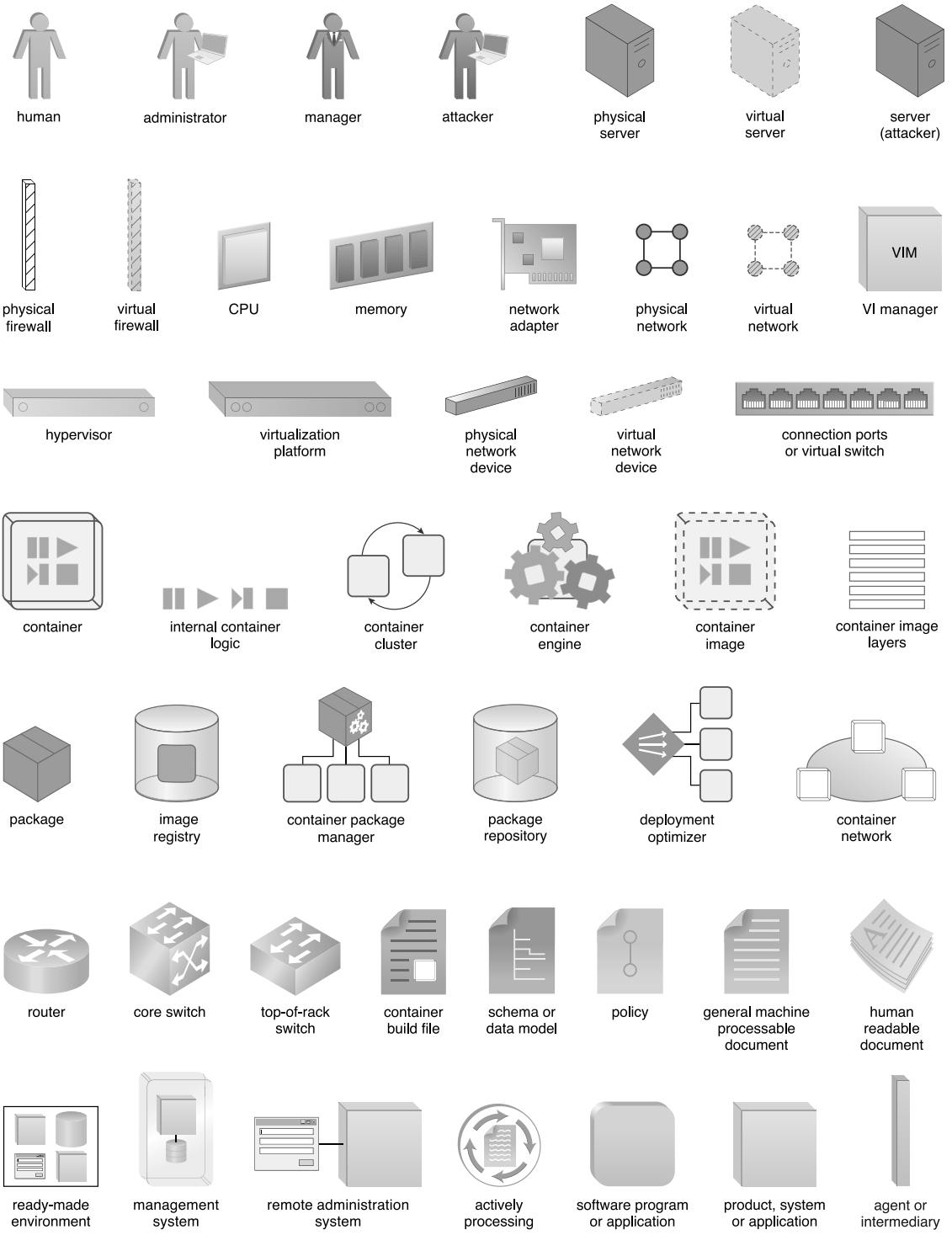
Foreword by David Linthicum

SECOND EDITION

Cloud Computing

Concepts, Technology, Security & Architecture

by Top-Selling Author **Thomas Erl**
with Eric Barceló Monroy
with contributions from Professor Zaigham Mahmood and Dr. Ricardo Puttini

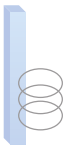

human
administrator
manager
attacker
physical server
virtual server
server (attacker)
physical firewall
virtual firewall
CPU
memory
network adapter
physical network
virtual network
VIM
VI manager
hypervisor
virtualization platform
physical network device
virtual network device
connection ports or virtual switch
container
internal container logic
container cluster
container engine
container image
container image layers
package
image registry
container package manager
package repository
deployment optimizer
container network
router
core switch
top-of-rack switch
container build file
schema or data model
policy
general machine processable document
human readable document
ready-made environment
management system
remote administration system
actively processing
software program or application
product, system or application
agent or intermediary


traffic monitor


network intrusion monitor


activity log monitor


authentication log monitor


VPN monitor

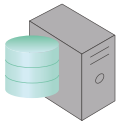

data loss protection monitor

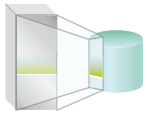

machine learning system


AI system


biometric scanner

multi-factor authentication (MFA) system


identity & access management (IAM) system


digital virus scanning & decryption system


malicious code analysis system


data loss prevention (DLP) system


intrusion detection system (IDS)


penetration testing tool


user behavior analytics (UBA) system

third-party software update utility


trusted platform module (TPM)


data backup & recovery system

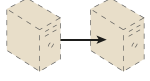

cybersecurity solution

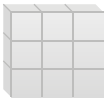
malware

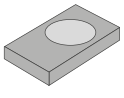

malicious packet


virus


virtual desktops

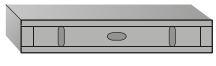

live VM migration


multitenant application

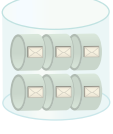

hard disk


hard disk with enclosure


storage device (internal)

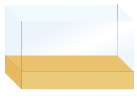

storage controller


databases


message queue

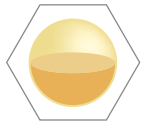

repository or storage device


shared storage


state data in memory


service with state data (stateful service)


repository with state data


grid service

service or proxy

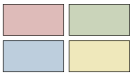
service composition

service layer

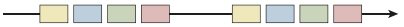
service contract (chorded circle notation)

decoupled service contract

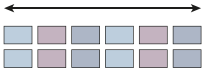
service inventory



LUNs



LUN migration



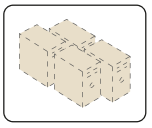
storage replication

live storage migration

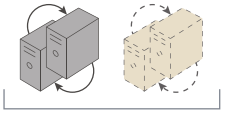
security element or locked resource

message

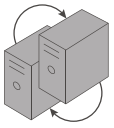
encrypted message



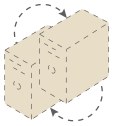
resource pool



resource clusters



physical server cluster



virtual server cluster

malicious component or program

trusted attacker

attacker

malicious service agent

private key

public key

heartbeat message

conflict symbol

web browser

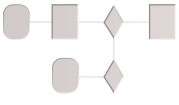
web user interface

folder

workstation

mobile computer

mobile devices

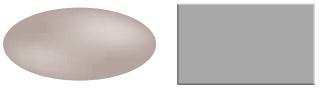


business process/ workflow logic

file system

runtime

transition arrow



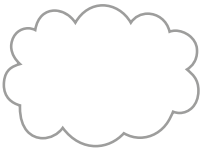
symbols used in conceptual relationship diagrams

organization

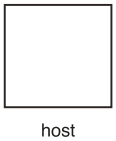
zone or region

internet

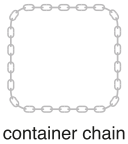
virtual private network



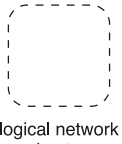
cloud



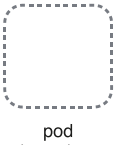
host boundary



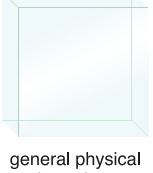
container chain boundary



logical network perimeter or logical boundary



pod boundary



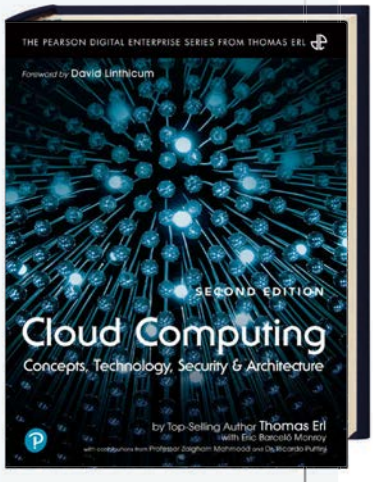
general physical boundary

container boundary

system or program boundary

Arcitura®
the IT education company

ARCITURA EDUCATION
PROFESSIONAL ACCREDITATION PROGRAMS



This book is an official supplement for the following Arcitura certification programs:

- Cloud Technology Professional
- Cloud Architect
- Cloud Security Specialist
- Cloud Computing Consultant
- Containerization Architect
- Cybersecurity Specialist
- Digital Transformation Technology Professional
- Digital Transformation Technology Architect

For more information, visit: www.arcitura.com

Learn more about Arcitura courses and certifications at:
youtube.com/@arcitura

itcc PCC PROFESSIONAL CERTIFICATION COALITION Pearson | VUE OnVUE Acclaim Credly

THE PEARSON DIGITAL ENTERPRISE SERIES FROM THOMAS ERL

Learn from Top-Selling Authors and Leading Industry Experts

ABOUT THE SERIES

The Pearson Digital Enterprise Series from Thomas Erl aims to provide the IT industry with comprehensive, unbiased coverage of the contemporary practices and technology innovations that are driving the international adoption and evolution of digital transformation and the realization of digital enterprises. Each title in this book series is authored in relation to other titles so as to establish a library of complementary knowledge. Although the series covers a broad spectrum of topics, each title is authored in compliance with common language, vocabulary, and illustration conventions so as to enable readers to continually explore cross-topic research and education.

Book Series Website: DigitalEnterpriseBookSeries.com

Book Series LinkedIn Group: LinkedIn.com/groups/2954416

ThomasErl.com

LinkedIn.com/in/thomaserl

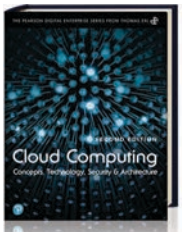
YouTube.com/@terl

YouTube.com/@arcitura

ABOUT THE SERIES EDITOR

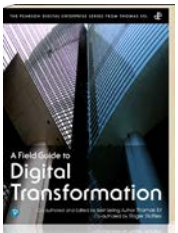
Thomas Erl is a best-selling IT author and the series editor of the Pearson Digital Enterprise Series from Thomas Erl (www.thomaserl.com/books). You can find Thomas on the Thomas Erl YouTube channel (youtube.com/@terl). He is also the host of the *Real Digital Transformation* podcast series (available via Spotify, Apple, Google Podcasts, and most other platforms) and also publishes the LinkedIn newsletter *The Digital Enterprise*. Over 100 articles and interviews by Thomas have been published in numerous publications, including *CEO World*, *The Wall Street Journal*, *Forbes*, and *CIO Magazine*. Thomas has also toured over 20 countries as a keynote speaker for various conferences and events.

As CEO of Arcitura Education (www.arcitura.com), Thomas has led the development of curricula for internationally recognized, vendor-neutral training and accreditation programs. Arcitura's portfolio currently consists of over 100 course modules, over 100 Pearson VUE exams, and over 40 certification tracks, covering topics such as Digital Transformation, Robotic Process Automation (RPA), DevOps, Blockchain, IoT, Containerization, Machine Learning, Artificial Intelligence (AI), Cybersecurity, Service-Oriented Architecture (SOA), Cloud Computing, and Big Data Analytics. Thomas is also the founder and senior advisor at Transformative Digital Solutions (www.transformative.digital), as well as a freelance LinkedIn Learning instructor and courseware author.



Cloud Computing: Concepts, Technology, Security & Architecture (Second Edition)
by T. Erl, E. Barceló

ISBN: 9780138052256
Paperback

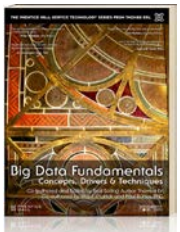


A Field Guide to Digital Transformation
by T. Erl, R. Stoffers

ISBN: 9780137571840
Paperback, 278 pages

Service-Oriented Architecture: Analysis & Design for Services and Microservices (Second Edition)
by T. Erl, P. Merson, R. Stoffers

ISBN: 9780133858587
Paperback, 393 pages



Big Data Fundamentals: Concepts, Drivers & Techniques
by P. Buhler, T. Erl, W. Khattak

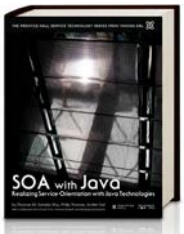
ISBN: 9780134291079
Paperback, 218 pages

Cloud Computing Design Patterns
by T. Erl, R. Cope, A. Naserpour

ISBN: 9780133858563
Paperback, 564 pages

Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation
by T. Erl, C. Gee, J. Kress, B. Maier, H. Normann, P. Raj, L Shuster, B. Trops, C. Utschig-Utschig, P. Wik, T. Winterberg

ISBN: 9780133859041
Paperback, 208 pages



SOA with Java: Realizing Service-Orientation with Java Technologies
by T. Erl, S. Roy, P. Thomas, A. Tost

ISBN: 9780133859034
Paperback, 590 pages

Cloud Computing: Concepts, Technology & Architecture
by T. Erl, Z. Mahmood, R. Puttini

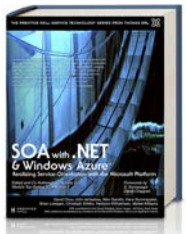
ISBN: 9780133387520
Paperback, 528 pages

SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST
by R. Balasubramanian, B. Carlyle, T. Erl, C. Pautasso

ISBN: 9780137012510
Paperback, 577 pages

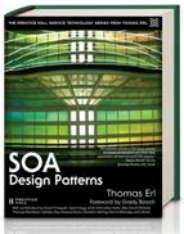
SOA Governance: Governing Shared Services On-Premise & in the Cloud
by S. Bennett, T. Erl, C. Gee, R. Laird, A. Manes, R. Schneider, L. Shuster, A. Tost, C. Venable

ISBN: 9780138156756
Paperback, 675 pages



SOA with .NET & Windows Azure: Realizing Service-Orientation with the Microsoft Platform
by D. Chou, J. de Vadoss, T. Erl, N. Gandhi, H. Kommalapati, B. Loesgen, C. Schittko, H. Wilhelmsen, M. Williams

ISBN: 9780131582316
Paperback, 893 pages



SOA Design Patterns
by T. Erl

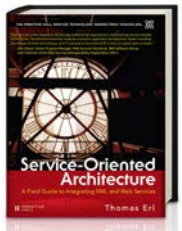
ISBN: 9780136135166
Paperback, 865 pages

Web Service Contract Design and Versioning for SOA
by T. Erl, H. Haas, A. Karmarkar, C. Liu, D. Orchard, J. Pasley, A. Tost, P. Walmsley, U. Yalcinalp

ISBN: 9780136135173
Paperback, 826 pages

SOA Principles of Service Design
by T. Erl

ISBN: 9780132344821
Paperback, 573 pages



Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services
by T. Erl

ISBN: 9780131428980
Paperback, 534 pages

Praise for the Second Edition

"This book is a solid and comprehensive overview of the cloud concepts and the real mechanics of how the cloud works. It does a nice job of not only explaining cloud function and architecture but the business impact. It's a great foundation for creating a cloud roadmap."

—Jo Peterson, VP Cloud and Security, Clarify360

"The book provides a comprehensive and well-researched guide to cloud computing. It covers a wide range of topics, from the basics of cloud computing to advanced architectural considerations. The book is written in a clear and concise style and is packed with valuable information and Case Studies. I highly recommend this book to anyone who wants to learn more about cloud computing."

—Jorge Blanco, Managing Director, Corporate Reinvention and Education Director, Glumin

"With its comprehensive insights and vendor-neutral approach, this book truly shines as a beacon of knowledge in the complex world of cloud computing. The author's emphasis on informed decision-making and alignment with business objectives sets the stage for success in adopting this transformative technology. By navigating the dangers and challenges, the book empowers readers to make strategic choices, safeguarding their organizations from potential pitfalls. From fundamental concepts to practical considerations, each chapter provides a roadmap for unlocking the full potential of cloud computing. With its invaluable case studies, architectural insights, and focus on service quality metrics and SLAs, this book is an indispensable resource for anyone seeking to harness the power of the cloud. It serves as a cornerstone in the Cloud Certified Professional program, providing a solid foundation for further exploration in this dynamic field. A must-have for every forward-thinking professional."

—Valther Galván, Chief Information Security Officer

"This is the most complete book on cloud computing concepts available. It starts with a solid foundation and builds from there. The case studies complement the content and illustrate the possibilities and difficulties."

—Emmett Dulaney, University Professor and Author

Praise for the First Edition

(Affiliations were current when the first edition was released, but may have changed.)

"Cloud computing, more than most disciplines in IT, suffers from too much talk and not enough practice. Thomas Erl has written a timely book that condenses the theory and buttresses it with real-world examples that demystify this important technology. An important guidebook for your journey into the cloud."

—Scott Morrison, Chief Technology Officer, Layer 7 Technologies

"An excellent, extremely well-written, lucid book that provides a comprehensive picture of cloud computing, covering multiple dimensions of the subject. The case studies presented in the book provide a real-world, practical perspective on leveraging cloud computing in an organization. The book covers a wide range of topics, from technology aspects to the business value provided by cloud computing. This is the best, most comprehensive book on the subject—a must-read for any cloud computing practitioner or anyone who wants to get an in-depth picture of cloud computing concepts and practical implementation."

—Suzanne D'Souza, SOA/BPM Practice Lead, KBACE Technologies

"This is a great book on the topic of cloud computing. It is impressive how the content spans from taxonomy, technology, and architectural concepts to important business considerations for cloud adoption. It really does provide a holistic view to this technology paradigm."

—Kapil Bakshi, Architecture and Strategy, Cisco Systems Inc.

"I have read every book written by Thomas Erl and *Cloud Computing* is another excellent publication and demonstration of Thomas Erl's rare ability to take the most complex topics and provide critical core concepts and technical information in a logical and understandable way."

—Melanie A. Allison, Principal, Healthcare Technology Practice, Integrated Consulting Services

"Companies looking to migrate applications or infrastructure to the cloud are often misled by buzzwords and industry hype. This work cuts through the hype and provides a detailed look, from investigation to contract to implementation to termination, at what it takes for an organization to engage with cloud service providers. This book really lays out the benefits and struggles with getting a company to an IaaS, PaaS, or SaaS solution."

—Kevin Davis, Ph.D., Solutions Architect

"Thomas, in his own distinct and erudite style, provides a comprehensive and a definitive book on cloud computing. Just like his previous masterpiece, *Service-Oriented Architecture: Concepts*, *Technology*, *and Design*, this book is sure to engage CxOs, cloud architects, and the developer community involved in delivering software assets on the cloud. Thomas and his authoring team have taken great pains in providing great clarity and detail in documenting cloud architectures, cloud delivery models, cloud governance, and economics of cloud, without forgetting to explain the core of cloud computing that revolves around Internet architecture and virtualization. As a reviewer for this outstanding book, I must admit I have learned quite a lot while reviewing the material. A 'must have' book that should adorn everybody's desk!"

—Vijay Srinivasan, Chief Architect - Technology, Cognizant Technology Solutions

"This book provides comprehensive and descriptive vendor-neutral coverage of cloud computing technology, from both technical and business aspects. It provides a deep-down analysis of cloud architectures and mechanisms that capture the real-world moving parts of cloud platforms. Business aspects are elaborated on to give readers a broader perspective on choosing and defining basic cloud computing business models. Thomas Erl's *Cloud Computing: Concepts*, *Technology & Architecture* is an excellent source of knowledge of fundamental and in-depth coverage of cloud computing."

—Masykur Marhendra Sukmanegara, Communication Media & Technology, Consulting Workforce Accenture

"The richness and depth of the topics discussed are incredibly impressive. The depth and breadth of the subject matter are such that a reader could become an expert in a short amount of time."

—Jamie Ryan, Solutions Architect, Layer 7 Technologies

"Demystification, rationalization, and structuring of implementation approaches have always been strong parts in each and every one of Thomas Erl's books. This book is no exception. It provides the definitive, essential coverage of cloud computing and, most importantly, presents this content in a very comprehensive manner. Best of all, this book follows the conventions of the previous service technology series titles, making it read like a natural extension of the library. I strongly believe that this will be another bestseller from one of the top-selling IT authors of the past decade."

—Sergey Popov, Senior Enterprise Architect SOA/Security, Liberty Global International

"A must-read for anyone involved in cloud design and decision making! This insightful book provides in-depth, objective, vendor-neutral coverage of cloud computing concepts, architecture models, and technologies. It will prove very valuable to anyone who needs to gain a solid understanding of how cloud environments work and how to design and migrate solutions to clouds."

—Gijs in 't Veld, Chief Architect, Motion10

"A reference book covering a wide range of aspects related to cloud providers and cloud consumers. If you would like to provide or consume a cloud service and need to know how, this is your book. The book has a clear structure to facilitate a good understanding of the various concepts of cloud."

—Roger Stoffers, Solution Architect

"Cloud computing has been around for a few years, yet there is still a lot of confusion around the term and what it can bring to developers and deployers alike. This book is a great way of finding out what's behind the cloud, and not in an abstract or high-level manner: It dives into all of the details that you'd need to know in order to plan for developing applications on cloud and what to look for when using applications or services hosted on a cloud. There are very few books that manage to capture this level of detail about the evolving cloud paradigm as this one does. It's a must for architects and developers alike."

—Dr. Mark Little, Vice President, Red Hat

"This book provides a comprehensive exploration of the concepts and mechanics behind clouds. It's written for anyone interested in delving into the details of how cloud environments function, how they are architected, and how they can impact business. This is the book for any organization seriously considering adopting cloud computing. It will pave the way to establishing your cloud computing roadmap."

—Damian Maschek, SOA Architect, Deutsche Bahn

"One of the best books on cloud computing I have ever read. It is complete yet vendor technology neutral and successfully explains the major concepts in a well-structured and disciplined way. It goes through all the definitions and provides many hints for organizations or professionals who are approaching and/or assessing cloud solutions. This book gives a complete list of topics playing fundamental roles in the cloud computing discipline. It goes through a full list of definitions very clearly stated. Diagrams are simple to understand and self-contained. Readers with different skill sets, expertise, and backgrounds will be able to understand the concepts seamlessly."

—Antonio Bruno, Infrastructure and Estate Manager, UBS AG

"*Cloud Computing: Concepts, Technology & Architecture* is a comprehensive book that focuses on what cloud computing is really all about…. This book will become the foundation on which many organizations will build successful cloud adoption projects. It is a must-read reference for both IT infrastructure and application architects interested in cloud computing or involved in cloud adoption projects. It contains extremely useful and comprehensive information for those who need to build cloud-based architectures or need to explain it to customers thinking about adopting cloud computing technology in their organization."

—Johan Kumps, SOA Architect, RealDolmen

"This book defines the basic terminology and patterns for the topic—a useful reference for the cloud practitioner. Concepts from multitenancy to hypervisor are presented in a succinct and clear manner. The underlying case studies provide wonderful real-worldness."

—Dr. Thomas Rischbeck, Principal Architect, ipt

"The book provides a good foundation to cloud services and issues in cloud service design. Chapters highlight key issues that need to be considered in learning how to think in cloud technology terms; this is highly important in today's business and technology environments where cloud computing plays a central role in connecting user services with virtualized resources and applications."

—Mark Skilton, Director, Office of Strategy and Technology, Global Infrastructure Services, Capgemini

"The book is well organized and covers basic concepts, technologies, and business models about cloud computing. It defines and explains a comprehensive list of terminologies and glossaries about cloud computing so cloud computing experts can speak and communicate with the same set of standardized language. The book is easy to understand and consistent with early published books from Thomas Erl..… It is a must-read for both beginners and experienced professionals."

—Jian "Jeff" Zhong, Chief Technology Officer (Acting) and Chief Architect for SOA and Cloud Computing, Futrend Technology Inc.

"Students of the related specialties can fulfill their educational process with very easily understood materials that are broadly illustrated and clearly described. Professors of different disciplines, from business analysis to IT implementation—even legal and financial monitoring—can use the book as an on-table lecturing manual. IT specialists of all ranks and fields of application will find the book as a practical and useful support for sketching solutions unbound to any particular vendor or brand."

—Alexander Gromoff, Director of Science & Education, Center of Information Control Technologies, Chairman of BPM Chair in Business Informatics Department, National Research University "Higher School of Economics"

"*Cloud Computing: Concepts, Technology & Architecture* is a comprehensive compendium of all the relevant information about the transformative cloud technology. Erl's latest title concisely and clearly illustrates the origins and positioning of the cloud paradigm as the next-generation computing model. All the chapters are carefully written and arranged in an easy-to-understand manner. This book will be immeasurably beneficial for business and IT professionals. It is set to shake up and help organize the world of cloud computing."

—Pethuru Raj, Ph.D., Enterprise Architecture Consultant, Wipro

This page intentionally left blank

Cloud Computing

Concepts, Technology, Security & Architecture

SECOND EDITION

Thomas Erl
Eric Barceló Monroy

with contributions from
Professor Zaigham Mahmood and Dr. Ricardo Puttini

HOBOKEN, NJ • BOSTON • INDIANAPOLIS • SAN FRANCISCO
NEW YORK • TORONTO • MONTREAL • LONDON • MUNICH • PARIS • MADRID
CAPE TOWN • SYDNEY • TOKYO • SINGAPORE • MEXICO CITY

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com

Library of Congress Control Number: 2023939909

Copyright © 2024 Arcitura Education Inc.

Cover image: Thomas Erl

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

ISBN-13: 978-0-13-805225-6
ISBN-10: 0-13-805225-5

$PrintCode

Vice President
Mark L. Taub

Director, ITP Product Management
Brett Bartow

Executive Editor
Nancy Davis

Production Editors
Mary Roth
Tonya Simpson

Publishing Coordinator
María Pareni Barceló Nieves

Technical Reviewers
Jo Peterson
Emmett Dulaney

Development Editor
Eleanor Bru

Indexer
Cheryl Lenser

Proofreader
Paula Lowell

Compositor
Bumpy Design

Photos
Thomas Erl

Graphics
Kamilla Bieska

To my family and friends
—Thomas Erl

To Eva, Pareni, Víctor, and Diego with all my love
—Eric Barceló Monroy

This page intentionally left blank

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

This page intentionally left blank

Contents at a Glance

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

by David S. Linthicum

Finally, an owner's manual for cloud computing.

Most enterprises got cloud computing wrong. Not "Going out of business" wrong, but the majority ended up with under-optimized cloud-based systems that failed to return the value stakeholders expected.

What happened? Most people blame over-hyped technology, "cloud washing," and faster-than-needed movement to cloud-based platforms. The honest answer is that there were and still are not enough qualified cloud computing solutions designers and builders to go around. Even the cloud salespeople started with too little cloud expertise to adequately advise their clients.

It's hard to gain experience and qualifications with a complex new technology that requires a mostly custom solution for every implementation, especially when the cloud "pioneers" are in such high demand that they rarely have time to teach others their skills.

For far too long we've worked off the assumption that if something works, it's also optimized. The unoptimized result in a cloud deployment is a solution that removes value from the business over time. Keep replicating these mistakes and you will soon enjoy a negative value from the use of cloud computing.

Back in 2008 and 2009, when cloud computing hype first arose in the fast-moving technology market, promises of 10-fold cloud ROIs were common. Instead of getting $10 back on every dollar invested, most enterprises only return about $0.50 back on each dollar invested.

Think of the problem this way: Flying from LA to New York on a budget carrier in coach class will cost about 1% of the fare on a private jet. Both planes will get you from Point A to Point B, but too many enterprise clouds are chartered jets. As with flight costs, many happy medium choices are available in cloud computing that will result in a satisfactory compromise between efficiency and costs. This compromise requires understanding the data, security, governance, and required application behaviors that need to be addressed with a carefully configured cloud computing architecture and enabling technology that creates a fully optimized solution pattern.

The Missing Manual

What we have is an education problem rather than a technology problem. Most enterprises faked their way through their initial cloud implementations using bits and pieces of what they understood from more traditional technology platforms. There are too many vastly reaching assumptions about the capabilities of the emerging cloud computing technology.

Of course, no single source can provide all the knowledge of what "the cloud" is and does. This book stands out as a source of practical knowledge that offers a comprehensive understanding of cloud technology and how it can be effectively utilized to solve most business problems using standard and advanced cloud architecture concepts. Better put, this book provides you with the knowledge needed to find the value of cloud computing that was initially promised.

Like most good owner's manuals, this book includes the basics that serve as a "quick start" guide as well as advice to successfully leverage cloud mechanisms. Erl then delves into advanced concepts that can only be learned through experience. The basics will get you through a cloud job interview. Erl's discussions on advanced concepts surpass what most of us in the cloud architecture field have currently considered.

What I find most engaging is that Erl does not focus on specific technology brands, understanding that those technologies will quickly evolve. Good solutions begin as a concept. Unfortunately, we often misunderstand what those solutions should do or be by inserting specific branded technology too early in the process. This is especially true when designing and building cloud computing solutions. Erl leaves brands out of the discussion, making the concepts in this book much more useful and applicable across different technologies and through the evolution of technologies over time.

With the heart of a teacher, Erl puts what others understand into a useful aggregation of that knowledge. Read this book to educate yourself on cloud computing concepts, designs, architecture, and other advanced concepts in a structure that builds upon other concepts in logical ways. The information imparted will make sense to those who are in the early days of their cloud journey, as well as to those who are more advanced. This is a manual that's useful to all levels and for all needs. It's a reference you will return to many times in your own cloud computing journey to ensure you're doing things correctly.

Finally, Find the Value of Cloud Computing

I suspect that most of you are here because you've seen cloud computing underwhelm your business and you are wondering how to fix it. This is the only well-structured and complete manual you'll need to figure out how you can get cloud computing right. Turn the concepts presented in this book into optimized solutions that maximize the value returned to the business.

This book is about making the right choices, understanding why those choices are made, and determining the best choices for the business. If there is a user manual for cloud computing, both advanced and basic concepts, this is it.

It will help you better understand the correct application of any technology and its usefulness in solving your problems. Indeed, avoiding going down many of the "rabbit holes" that can either waste time, or more likely lead you to the wrong decisions.

Happy computing.

David S. Linthicum
Author, Speaker, Educator, and Consultant

This page intentionally left blank

About the Authors

Thomas Erl

Thomas Erl is a best-selling IT author and series editor of the Pearson Digital Enterprise Series from Thomas Erl. Thomas has authored and co-authored 15 books published by Pearson Education and Prentice Hall dedicated to contemporary business technology and practices. You can find Thomas on the Thomas Erl YouTube channel (youtube.com/@terl). He is also the host of the *Real Digital Transformation* podcast series (available via Spotify, Apple, Google Podcasts, and most other platforms) and also publishes the weekly LinkedIn newsletter *The Digital Enterprise*. Over 100 articles and interviews by Thomas have been published in numerous publications, including *CEO World*, *The Wall Street Journal*, *Forbes*, and *CIO Magazine*. Thomas has also toured over 20 countries as a keynote speaker for various conferences and events.

At Arcitura Education (www.arcitura.com), Thomas leads the development of curricula for internationally recognized, vendor-neutral training and accreditation programs. Arcitura's portfolio currently consists of over 100 courses, over 100 Pearson VUE exams, and over 40 certification tracks, covering topics such as Cloud Computing Architecture, Security, and Governance, as well as Digital Transformation, Robotic Process Automation (RPA), DevOps, Blockchain, IoT, Containerization, Machine Learning, Artificial Intelligence (AI), Cybersecurity, Service-Oriented Architecture (SOA), and Big Data Analytics. Thomas is also the founder and senior advisor at Transformative Digital Solutions (www.transformative.digital) and a freelance LinkedIn instructor and courseware author.

www.thomaserl.com

Eric Barceló Monroy

Eric Barceló Monroy is an IT professional with extensive experience in IT strategic planning, operational and administrative process re-engineering, system implementation project management, and IT operations. He has a proven track record of implementing systems that exceed user expectations while reducing costs and improving response times. He has held various high-level positions in both the private and public sectors, including Director of Information Technology at Farmacéuticos MAYPO, a pharmaceutical distributor; Vice-president of Telecommunications and Technology Operations at iExplore, an internet-based adventure travel agency; and Director of Information Technology and Telecommunications at the Ministry of Education in Tabasco, Mexico, where he oversaw the implementation of telecommunication networks among schools and develops and delivers computer literacy training programs for faculty.

Additionally, he is a partner and Technical Consulting Director at EGN, a cloud technology consulting and training firm, where he provides IT consultancy on state of-the-art topics like Big Data, Cloud Computing, Virtualization, Advanced Networking, and Strategic IT Management. Eric is a Certified Cloud Computing Technology Professional, Certified Cloud Virtualization Specialist, and Certified Cloud Architect, among others. He is also a VMware Certified Professional, Red Hat Certified System Administrator, Red Hat Certified Engineer, and Certified Amazon Web Services Solutions Architect.

Acknowledgments

We would like to acknowledge the co-authors of the first edition of this book:

- Prof Zaigham Mahmood, Derby, UK
- Ricardo Puttini, PhD, Core Consulting

Acknowledgments for the second edition in alphabetical order by last name:

- Gustavo Azzolin
- Jorge Blanco, Managing Director, Corporate Reinvention and Education Director, Glumin
- Emmett Dulaney, University Professor and Author
- Valther Galván, Chief Information Security Officer
- David Linthicum, Deloitte Consulting
- Vinícius Pacheco, University of Brasília, Brazil
- Jo Peterson, VP Cloud and Security, Clarify360
- Pamela J. Wise-Martinez, Global Chief Architect, Whirlpool Corporation
- Matthias Ziegler

Acknowledgments for the first edition in alphabetical order by last name (affiliations were current when the first edition was released, but may have changed):

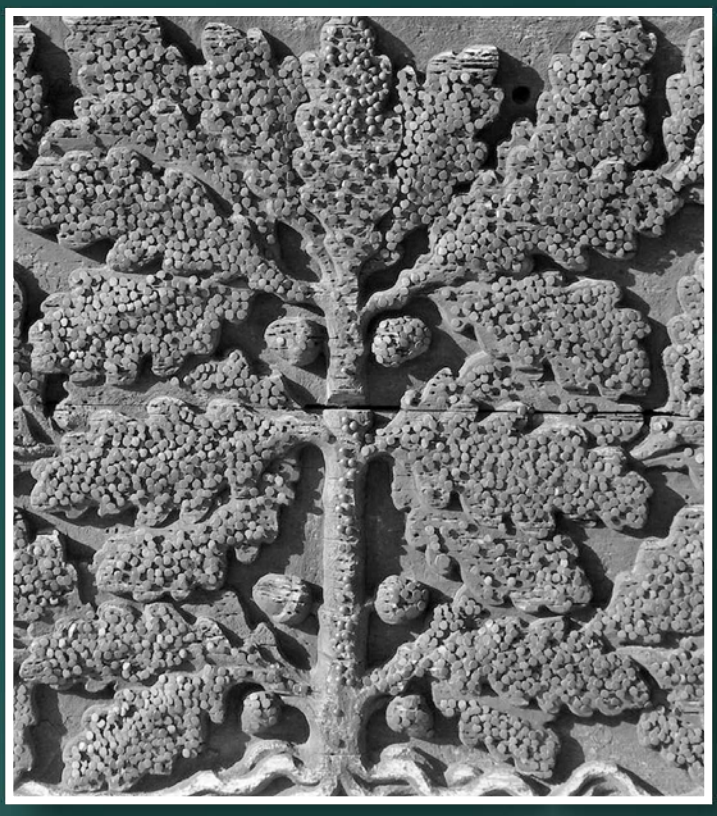
- Ahmed Aamer, AlFaisaliah Group
- Randy Adkins, Modus21
- Melanie Allison, Integrated Consulting Services
- Gabriela Inacio Alves, University of Brasilia
- Marcelo Ancelmo, IBM Rational Software Services
- Kapil Bakshi, Cisco Systems
- Toufic Boubez, Metafor Software
- Antonio Bruno, UBS AG
- Dr. Paul Buhler, Modus21
- Pethuru Raj Cheliah, Wipro
- Kevin Davis, Ph.D.
- Suzanne D'Souza, KBACE Technologies
- Yili Gong, Wuhan University
- Alexander Gromoff, Center of Information Control Technologies
- Chris Haddad, WSO2
- Richard Hill, University of Derby
- Dr. Michaela Iorga, Ph.D.
- Johan Kumps, RealDolmen
- Gijs in 't Veld, Motion10
- Masykur Marhendra, Consulting Workforce Accenture
- Damian Maschek, Deutshe Bahn
- Claynor Mazzarolo, IBTI
- Charlie Mead, W3C
- Steve Millidge, C2B2

- Jorge Minguez, Thales Deutschland
- Scott Morrison, Layer 7
- Amin Naserpour, HP
- Vicente Navarro, European Space Agency
- Laura Olson, IBM WebSphere
- Tony Pallas, Intel
- Cesare Pautasso, University of Lugano
- Sergey Popov, Liberty Global International
- Olivier Poupeney, Dreamface Interactive
- Alex Rankov, EMC
- Dan Rosanova, West Monroe Partners
- Jaime Ryan, Layer 7
- Filippos Santas, Credit Suisse
- Christoph Schittko, Microsoft
- Guido Schmutz, Trivadis
- Mark Skilton, Capgemini
- Gary Smith, CloudComputingArchitect.com
- Kevin Spiess
- Vijay Srinivasan, Cognizant
- Daniel Starcevich, Raytheon
- Roger Stoffers, HP
- Andre Toffanello, IBTI
- Andre Tost, IBM Software Group
- Bernd Trops, talend
- Clemens Utschig, Boehringer Ingelheim Pharma
- Ignaz Wanders, Archimiddle

- Philip Wik, Redflex
- Jorge Williams, Rackspace
- Dr. Johannes Maria Zaha
- Jeff Zhong, Futrend Technologies

Special thanks to the research and development teams at Arcitura Education (www.arcitura.com) that produced the Cloud Computing, Cloud Architecture, Containerization Architecture, Cloud Security and Cybersecurity courses upon which this book is based.

Chapter 1



Introduction

Cloud computing is, at its essence, a form of service provisioning. As with any type of service we intend to hire or outsource (IT-related or otherwise), it is commonly understood that we will be confronted with a marketplace comprised of service providers of varying quality and reliability. Some may offer attractive rates and terms, but may have unproven business histories or highly proprietary environments. Others may have a solid business background, but may demand higher rates and less flexible terms. Others yet may simply be insincere or temporary business ventures that unexpectedly disappear or are acquired within a short period of time.

There is no greater danger to a business than approaching cloud computing adoption with ignorance. The magnitude of a failed adoption effort not only correspondingly impacts IT departments, but can actually regress a business to a point where it finds itself steps behind from where it was prior to the adoption—and, perhaps, even more steps behind competitors that have been successful at achieving their goals in the meantime.

Cloud computing has much to offer, but its roadmap is riddled with pitfalls, ambiguities, and mistruths. The best way to navigate this landscape is to chart each part of the journey by making educated decisions about how and to what extent your project should proceed. The scope of an adoption is equally important to its approach, and both of these aspects need to be determined by business requirements—not by a product vendor, not by a cloud vendor, and not by self-proclaimed cloud experts. Your organization's business goals must be fulfilled in a concrete and measurable manner with each completed phase of the adoption. This validates your scope, your approach, and the overall direction of the project. In other words, it keeps your project aligned.

Gaining a vendor-neutral understanding of cloud computing from an industry perspective empowers you with the clarity necessary to determine what is factually cloud-related and what is not, as well as what is relevant to your business requirements and what is not. With this information you can establish criteria that will allow you to filter out irrelevant parts of the cloud computing product and service provider marketplaces to focus only on what has the most potential to help you and your business to succeed. We developed this book to assist you with this goal.

—Thomas Erl

1.1 Objectives of This Book

This book is the result of much research and analysis of the commercial cloud computing industry, cloud computing vendor platforms, and further innovation and contributions made by cloud computing industry standards organizations and practitioners. The purpose of this book is to break down proven and mature cloud computing technologies and practices into a series of well-defined concepts, models, and technology mechanisms and architectures. The resulting chapters establish concrete, academic coverage of fundamental aspects of cloud computing concepts and technologies. The range of topics covered is documented using vendor-neutral terms and descriptions, carefully defined to ensure full alignment with the cloud computing industry as a whole.

1.2 What This Book Does Not Cover

Due to the vendor-neutral basis of this book, it does not contain any significant coverage of cloud computing vendor products, services, or technologies. This book is complementary to other titles that provide product-specific coverage and to vendor product literature itself. If you are new to the commercial cloud computing landscape, you are encouraged to use this book as a starting point before proceeding to books and courses that are proprietary to vendor product lines.

1.3 Who This Book Is For

This book is aimed at the following target audience:

- IT practitioners and professionals who require vendor-neutral coverage of cloud computing technologies, concepts, mechanisms, and models
- IT managers and decision-makers who seek clarity regarding the business and technological implications of cloud computing
- professors and students and educational institutions that require well-researched and well-defined academic coverage of fundamental cloud computing topics
- business managers who need to assess the potential economic gains and viability of adopting cloud computing resources
- technology architects and developers who want to understand the different moving parts that comprise contemporary cloud platforms

1.4 How This Book Is Organized

The book begins with Chapters 1 and 2 providing introductory content and background information for the case studies. All subsequent chapters are organized into the following parts:

- Part I: Fundamental Cloud Computing
- Part II: Cloud Computing Mechanisms
- Part III: Cloud Computing Architecture
- Part IV: Working with Clouds
- Part V: Appendices

Part I: Fundamental Cloud Computing

The five chapters in this part cover introductory topics in preparation for all subsequent chapters. Note that Chapters 3 and 4 do not contain case study content.

Chapter 3: Understanding Cloud Computing

Following a brief history of cloud computing and a discussion of business drivers and technology innovations, basic terminology and concepts are introduced, along with descriptions of common benefits and challenges of cloud computing adoption.

Chapter 4: Fundamental Concepts and Models

Cloud delivery and cloud deployment models are discussed in detail, followed by sections that establish common cloud characteristics and roles and boundaries.

Chapter 5: Cloud-Enabling Technology

Contemporary technologies that realize modern-day cloud computing platforms and innovations are discussed, including data centers, virtualization, containerization, and web-based technologies.

Chapter 6: Understanding Containerization

A comparison of virtualization and containerization is provided, along with in-depth coverage of containerization environments and components.

Chapter 7: Understanding Cloud Security and Cybersecurity

Cloud security and cybersecurity topics and concepts relevant and distinct to cloud computing are introduced, including descriptions of common cloud security threats and attacks.

Part II: Cloud Computing Mechanisms

Technology mechanisms represent well-defined IT artifacts that are established within an IT industry and commonly distinct to a certain computing model or platform. The technology-centric nature of cloud computing requires the establishment of a formal level of mechanisms to be able to explore how solutions can be assembled via different combinations of mechanism implementations.

This part formally documents 48 technology mechanisms that are used within cloud environments to enable generic and specialized forms of functionality. Each mechanism description is accompanied by a case study example that demonstrates its usage. The utilization of select mechanisms is further explored throughout the technology architectures covered in Part III.

Chapter 8: Cloud Infrastructure Mechanisms

Technology mechanisms foundational to cloud platforms are covered, including logical network perimeter, virtual server, cloud storage device, cloud usage monitor, resource replication, hypervisor, ready-made environment, and container.

Chapter 9: Specialized Cloud Mechanisms

A range of specialized technology mechanisms is described, including automated scaling listener, load balancer, SLA monitor, pay-per-use monitor, audit monitor, failover system, resource cluster, multi-device broker, and state management database.

Chapter 10: Cloud Security and Cybersecurity Access-Oriented Mechanisms

Access-related security mechanisms that can be used to counter and prevent some of the threats described in Chapter 7 are covered, including encryption, hashing, digital signature, cloud-based security groups, public key infrastructure (PKI) system, single sign-on (SSO) system, hardened virtual server image, firewall, virtual private network (VPN), biometric scanner, multi-factor authentication (MFA) system, identity and access management (IAM) system, intrusion detection system (IDS), penetration testing tool, user behavior analytics (UBA) system, third-party software update utility, network intrusion monitor, authentication log monitor, and VPN monitor.

Chapter 11: Cloud Security and Cybersecurity Data-Oriented Mechanisms

Data-related security mechanisms that can be used to counter and prevent some of the threats described in Chapter 7 are covered, including digital virus scanning and decryption system, malicious code analysis system, data loss prevention (DLP) system, trusted platform module (TPM), data backup and recovery system, activity log monitor, traffic monitor, and data loss protection monitor.

Chapter 12: Cloud Management Mechanisms

Mechanisms that enable the hands-on administration and management of cloud-based IT resources are explained, including remote administration system, resource management system, SLA management system, and billing management system.

Part III: Cloud Computing Architecture

Technology architecture within the realm of cloud computing introduces requirements and considerations that manifest themselves in broadly scoped architectural layers and numerous distinct architectural models.

This set of chapters builds upon the coverage of cloud computing mechanisms from Part II by formally documenting 38 cloud-based technology architectures and scenarios in which different combinations of the mechanisms are documented in relation to fundamental, advanced, and specialized cloud architectures.

Chapter 13: Fundamental Cloud Architectures

Fundamental cloud architectural models establish baseline functions and capabilities. The architectures covered in this chapter are Workload Distribution, Resource Pooling, Dynamic Scalability, Elastic Resource Capacity, Service Load Balancing, Cloud Bursting, Elastic Disk Provisioning, Redundant Storage, and Multicloud.

Chapter 14: Advanced Cloud Architectures

Advanced cloud architectural models establish sophisticated and complex environments, several of which directly build upon fundamental models. The architectures covered in this chapter are Hypervisor Clustering, Virtual Server Clustering, Load-Balanced Virtual Server Instances, Nondisruptive Service Relocation, Zero Downtime, Cloud Balancing, Resilient Disaster Recovery, Distributed Data Sovereignty, Resource Reservation, Dynamic Failure Detection and Recovery, Rapid Provisioning, Storage Workload Management, and Virtual Private Cloud.

Chapter 15: Specialized Cloud Architectures

Specialized cloud architectural models address distinct functional areas. The architectures covered in this chapter are Direct I/O Access, Direct LUN Access, Dynamic Data Normalization, Elastic Network Capacity, Cross-Storage Device Vertical Tiering, Intra-Storage Device Vertical Data Tiering, Load-Balanced Virtual Switches, Multipath Resource Access, Persistent Virtual Network Configuration, Redundant Physical Connection for Virtual Servers, Storage Maintenance Window, Edge Computing, Fog Computing, Virtual Data Abstraction, Metacloud, and Federated Cloud Application.

Part IV: Working with Clouds

Cloud computing technologies and environments can be adopted to varying extents. An organization can migrate select IT resources to a cloud, while keeping all other IT resources on premises—or it can form significant dependencies on a cloud platform by migrating larger amounts of IT resources or even using the cloud environment to create them.

For any organization, it is important to assess a potential adoption from a practical and business-centric perspective to pinpoint the most common factors that pertain to financial investments, business impact, and various legal considerations. This set of chapters explores these and other topics related to the real-world considerations of working with cloud-based environments.

Chapter 16: Cloud Delivery Model Considerations

Cloud environments need to be built and evolved by cloud providers in response to cloud consumer requirements. Cloud consumers can use clouds to create or migrate IT resources to, subsequent to their assuming administrative responsibilities. This chapter provides a technical understanding of cloud delivery models from both the provider and consumer perspectives, each of which offers revealing insights into the inner workings and architectural layers of cloud environments.

Chapter 17: Cost Metrics and Pricing Models

Cost metrics for network, server, storage, and software usage are described, along with various formulas for calculating integration and ownership costs related to cloud environments. The chapter concludes with a discussion of cost management topics as they relate to common business terms used by cloud provider vendors.

Chapter 18: Service Quality Metrics and SLAs

Service-level agreements (SLAs) establish the guarantees and usage terms for cloud services and are often determined by the business terms agreed upon by cloud consumers and cloud providers. This chapter provides detailed insight into how cloud provider guarantees are expressed and structured via SLAs, along with metrics and formulas for calculating common SLA values, such as availability, reliability, performance, scalability, and resiliency.

Part V: Appendices

Appendix A: Case Study Conclusions

The individual storylines of the case studies are concluded and the results of each organization's cloud computing adoption efforts are summarized.

Appendix B: Common Containerization Technologies

This appendix acts as a supplement to Chapter 6 by providing a breakdown of the Docker and Kubernetes environments and relating those environments to the terms and components established in Chapter 6.

1.5 Resources

These sections provide supplementary information and resources.

Pearson Digital Enterprise Book Series

Information about the books in the Pearson Digital Enterprise Series from Thomas Erl and various supporting resources can be found at:

www.thomaserl.com/books

Thomas Erl on YouTube

Subscribe to the Thomas Erl YouTube channel for animated videos with storytelling and podcasts with industry experts. This YouTube channel is dedicated to digital technology, digital business, and digital transformation.

Subscribe at: www.youtube.com/@terl

The Digital Enterprise Newsletter on LinkedIn

The Digital Enterprise newsletter on LinkedIn publishes regular articles and videos relevant to contemporary digital technology and business topics.

Subscribe at: www.linkedin.com/newsletters/6909573501767028736

Cloud Certified Professional (CCP) Program

Arcitura Education offers vendor-neutral training and accreditation programs with a portfolio of more than 100 course modules and 40 certifications. This textbook is an official part of Arcitura's Cloud Certified Professional (CCP) curriculum.

Learn more at: www.arcitura.com

This page intentionally left blank

Chapter 2

Case Study Background

Case study examples provide scenarios in which organizations assess, use, and manage cloud computing models and technologies. Three organizations from different industries are presented for analysis in this book, each of which has distinctive business, technological, and architectural objectives that are introduced in this chapter.

The organizations presented for case study are:

- Advanced Telecom Networks (ATN) – a global company that supplies network equipment to the telecommunications industry
- DTGOV – a public organization that specializes in IT infrastructure and technology services for public sector organizations
- Innovartus Technologies Inc. – a medium-sized company that develops virtual toys and educational entertainment products for children

Most chapters after Part I include one or more *Case Study Example* sections. A conclusion to the storylines is provided in Appendix A.

2.1 Case Study #1: ATN

ATN is a company that provides network equipment to telecommunications industries across the globe. Over the years, ATN has grown considerably and their product portfolio has expanded to accommodate several acquisitions, including companies that specialize in infrastructure components for internet, GSM, and cellular providers. ATN is now a leading supplier of a diverse range of telecommunications infrastructure.

In recent years, market pressure has been increasing. ATN has begun looking for ways to increase its competitiveness and efficiency by taking advantage of new technologies, especially those that can assist in cost reduction.

Technical Infrastructure and Environment

ATN's various acquisitions have resulted in a highly complex and heterogeneous IT landscape. A cohesive consolidation program was not applied to the IT environment after each acquisition round, resulting in similar applications running concurrently and

an increase in maintenance costs. Years ago, ATN merged with a major European telecommunications supplier, adding another applications portfolio to its inventory. The IT complexity snowballed into a serious obstruction and became a source of critical concern to ATN's board of directors.

Business Goals and New Strategy

ATN management decided to pursue a consolidation initiative and outsource applications maintenance and operations overseas. This lowered costs but unfortunately did not address their overall operational inefficiency. Applications still had overlapping functions that could not be easily consolidated. It eventually became apparent that outsourcing was insufficient, as consolidation became a possibility only if the architecture of the entire IT landscape changed.

As a result, ATN decided to explore the potential of adopting cloud computing. However, subsequent to their initial inquiries they became overwhelmed by the plenitude of cloud providers and cloud-based products.

Roadmap and Implementation Strategy

ATN is unsure of how to choose the right set of cloud computing technologies and vendors—many solutions appear to still be immature and new cloud-based offerings continue to emerge in the market.

A preliminary cloud computing adoption roadmap is discussed to address a number of key points:

- *IT Strategy* – The adoption of cloud computing needs to promote optimization of the current IT framework and produce both lower short-term investments and consistent long-term cost reduction.
- *Business Benefits* – ATN needs to evaluate which of the current applications and IT infrastructure can leverage cloud computing technology to achieve the desired optimization and cost reductions. Additional cloud computing benefits such as greater business agility, scalability, and reliability need to be realized to promote business value.
- *Technology Considerations* – Criteria need to be established to help choose the most appropriate cloud delivery and deployment models and cloud vendors and products.

- *Cloud Security* – The risks associated with migrating applications and data to the cloud must be determined.

ATN fears that they might lose control over their applications and data if entrusted to cloud providers, leading to noncompliance with internal policies and telecom market regulations. They also wonder how their existing legacy applications would be integrated into the new cloud-based domain.

To define a succinct plan of action, ATN hires an independent IT consulting company called CloudEnhance, who are well recognized for their technology architecture expertise in the transition and integration of cloud computing IT resources. CloudEnhance consultants begin by suggesting an appraisal process consisting of five steps:

1. A brief evaluation of existing applications to measure factors such as complexity, business-criticality, usage frequency, and number of active users. The identified factors are then placed in a hierarchy of priority to help determine the most suitable candidate applications for migration to a cloud environment.
2. A more detailed evaluation of each selected application using a proprietary assessment tool.
3. The development of a target application architecture that exhibits the interaction between cloud-based applications, their integration with ATN's existing infrastructure and legacy systems, and their development and deployment processes.
4. The authoring of a preliminary business case that documents projected cost savings based on performance indicators, such as cost of cloud readiness, effort for application transformation and interaction, ease of migration and implementation, and various potential long-term benefits.
5. The development of a detailed project plan for a pilot application.

ATN proceeds with the process and resultantly builds its first prototype by focusing on an application that automates a low-risk business area. During this project, ATN ports several of the business area's smaller applications that were running on different technologies over to a platform as a service (PaaS) platform. Based on positive results and feedback received for the prototype project, ATN decides to embark on a strategic initiative to garner similar benefits for other areas of the company.

2.2 Case Study #2: DTGOV

DTGOV is a public company that was created in the early 1980s by the Ministry of Social Security. The decentralization of the ministry's IT operations to a public company under private law gave DTGOV an autonomous management structure with significant flexibility to govern and evolve its IT enterprise.

At the time of its creation, DTGOV had approximately 1,000 employees and operational branches in 60 localities nationwide, and operated two mainframe-based data centers. Over time, DTGOV has expanded to more than 3,000 employees and branch offices in more than 300 localities, with three data centers running both mainframe and low-level platform environments. Its main services are related to processing social security benefits across the country.

DTGOV has enlarged its customer portfolio in the last two decades. It now serves other public-sector organizations and provides basic IT infrastructure and services, such as server hosting and server colocation. Some of its customers have also outsourced the operation, maintenance, and development of applications to DTGOV.

DTGOV has sizable customer contracts that encompass various IT resources and services. However, these contracts, services, and associated service levels are not standardized; instead, negotiated service provisioning conditions are typically customized for each customer individually. DTGOV's operations are resultantly becoming increasingly complex and difficult to manage, which has led to inefficiencies and inflated costs.

The DTGOV board realized, some time ago, that the overall company structure could be improved by standardizing its services portfolio, which implies the reengineering of both IT operational and management models. This process has started with the standardization of the hardware platform through the creation of a clearly defined technological lifecycle, a consolidated procurement policy, and the establishment of new acquisition practices.

Technical Infrastructure and Environment

DTGOV operates three data centers: one is exclusively dedicated to low-level platform servers, while the other two have both mainframe and low-level platforms. The mainframe systems are reserved for the Ministry of Social Security and therefore not available for outsourcing.

The data center infrastructure occupies approximately 20,000 square feet of computer room space and hosts more than 100,000 servers with different hardware configurations.

The total storage capacity is approximately 10,000 terabytes. DTGOV's network has redundant high-speed data links connecting the data centers in a full-mesh topology. Internet connectivity is considered to be provider-independent since their network interconnects all the major national telecom carriers.

Server consolidation and virtualization projects have been in place for five years, considerably decreasing the diversity of hardware platforms. As a result, systematic tracking of the investments and operational costs related to the hardware platform has revealed significant improvement. However, there is still remarkable diversity in DTGOV's software platforms and configurations due to customer service customization requirements.

Business Goals and New Strategy

A chief strategic objective of the standardization of DTGOV's service portfolio is to achieve increased levels of cost-effectiveness and operational optimization. An internal executive-level commission was established to define the directions, goals, and strategic roadmap for this initiative. The commission has identified cloud computing as a guidance option and an opportunity for further diversification and improvement of services and customer portfolios.

The roadmap addresses the following key points:

- *Business Benefits* – Concrete business benefits associated with the standardization of service portfolios under the umbrella of cloud computing delivery models need to be defined. For example, how can the optimization of IT infrastructure and operational models result in direct and measurable cost reductions?
- *Service Portfolio* – Which services should become cloud-based, and which customers should they be extended to?
- *Technical Challenges* – The limitations of the current technology infrastructure in relation to the runtime processing requirements of cloud computing models must be understood and documented. Existing infrastructure must be leveraged to whatever extent possible to optimize up-front costs assumed by the development of the cloud-based service offerings.
- *Pricing and SLAs* – An appropriate contract, pricing, and service quality strategy needs to be defined. Suitable pricing and service-level agreements (SLAs) must be determined to support the initiative.

One outstanding concern relates to changes to the current format of contracts and how they may impact business. Many customers may not want to—or may not be prepared to—adopt cloud contracting and service delivery models. This becomes even more critical when considering the fact that 90% of DTGOV's current customer portfolio consists of public organizations that typically do not have the autonomy or the agility to switch operating methods on such short notice. Therefore, the migration process is expected to be long-term, which may become risky if the roadmap is not properly and clearly defined. A further outstanding issue pertains to IT contract regulations in the public sector—existing regulations may become irrelevant or unclear when applied to cloud technologies.

Roadmap and Implementation Strategy

Several assessment activities were initiated to address the aforementioned issues. The first was a survey of existing customers to probe their level of understanding, ongoing initiatives, and plans regarding cloud computing. Most of the respondents were aware of and knowledgeable about cloud computing trends, which was considered a positive finding.

An investigation of the service portfolio revealed clearly identified infrastructure services related to hosting and colocation. Technical expertise and infrastructure were also evaluated, determining that data center operation and management are key areas of expertise of DTGOV IT staff.

With these findings, the commission decided to:

1. choose infrastructure as a service (IaaS) as the target delivery platform to start the cloud computing provisioning initiative
2. hire a consulting firm with sufficient cloud provider expertise and experience to correctly identify and rectify any business and technical issues that may negatively affect the initiative
3. deploy new hardware resources with a uniform platform into two different data centers, aiming to establish a new, reliable environment to use for the provisioning of initial IaaS-hosted services
4. identify three customers that plan to acquire cloud-based services to establish pilot projects and define contractual conditions, pricing, and service-level policies and models

5. evaluate service provisioning of the three chosen customers for the initial period of six months before publicly offering the service to other customers

As the pilot project proceeds, a new web-based management environment is released to allow for the self-provisioning of virtual servers, as well as SLA and financial tracking functionality in realtime. The pilot projects are considered highly successful, leading to the next step of opening the cloud-based services to other customers.

2.3 Case Study #3: Innovartus Technologies Inc.

The primary business line of Innovartus Technologies Inc. is the development of virtual toys and educational entertainment products for children. These services are provided through a web portal that employs a role-playing model to create customized virtual games for PCs and mobile devices. The games allow users to create and manipulate virtual toys (cars, dolls, pets) that can be outfitted with virtual accessories that are obtained by completing simple educational quests. The main demographic is children under 12 years. Innovartus further has a social network environment that enables users to exchange items and collaborate with others. All of these activities can be monitored and tracked by the parents, who can also participate in a game by creating specific quests for their children.

The most valuable and revolutionary feature of Innovartus's applications is an experimental end-user interface that is based on natural interface concepts. Users can interact via voice commands, simple gestures that are captured with a webcam, and directly by touching tablet screens.

The Innovartus portal has always been cloud-based. It was originally developed via a PaaS platform and has been hosted by the same cloud provider ever since. Recently, however, this environment has revealed several technical limitations that impact features of Innovartus's user interface programming frameworks.

Technical Infrastructure and Environment

Many of Innovartus's other office automation solutions, such as shared file repositories and various productivity tools, are also cloud-based. The on-premises corporate IT environment is relatively small, consisting of mainly work area devices, laptops, and graphic design workstations.

Business Goals and Strategy

Innovartus has been diversifying the functionality of the IT resources that are used for their web-based and mobile applications. The company has also increased efforts to internationalize their applications: both the website and the mobile applications are currently offered in five different languages.

Roadmap and Implementation Strategy

Innovartus intends to continue building upon its cloud-based solutions. However, the current cloud hosting environment has limitations that need to be overcome:

- scalability needs to be improved to accommodate increased and less predictable cloud consumer interaction
- service levels need to be improved to avoid outages that currently occur more frequently than expected
- cost-effectiveness needs to be improved, as leasing rates are higher with the current cloud provider when compared to others

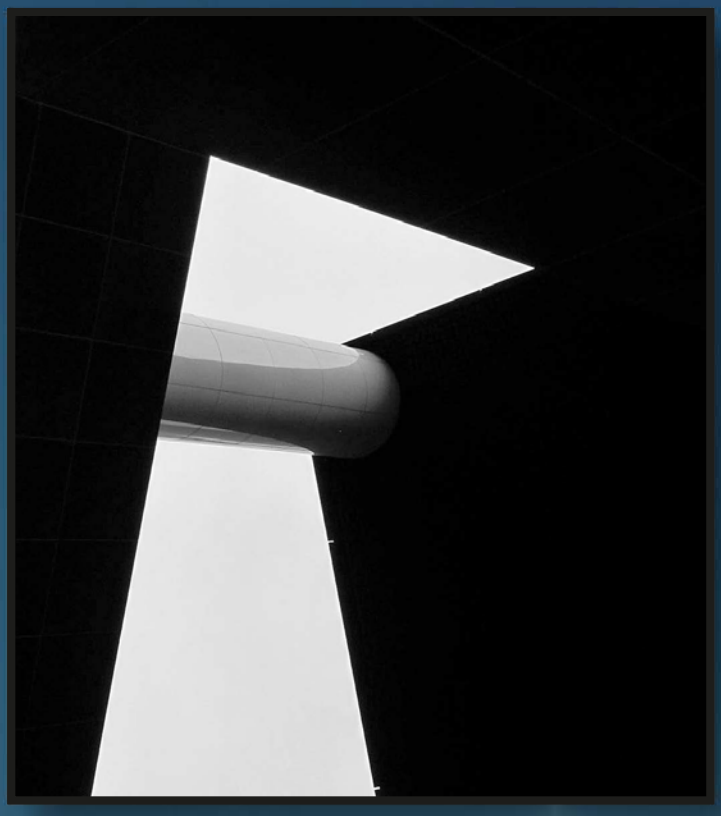
These and other factors have led Innovartus to decide to migrate to a larger, more globally established cloud provider.

The roadmap for this migration project includes:

- a technical and economic report about the risks and impacts of the planned migration
- a decision tree and a rigorous study initiative focused on the criteria for selecting the new cloud provider
- portability assessments of applications to determine how much of each existing cloud service architecture is proprietary to the current cloud provider's environment

Innovartus is further concerned about how and to what extent the current cloud provider will support and cooperate with the migration process.

This page intentionally left blank



Part I

Fundamental Cloud Computing

The upcoming chapters establish concepts and terminology that are referenced throughout subsequent chapters and parts in this book. It is recommended that Chapters 3 and 4 be reviewed, even for those already familiar with cloud computing fundamentals. Sections in Chapters 5, 6, and 7 can be selectively skipped by those already familiar with the corresponding technology and security topics.

Chapter 3

Understanding Cloud Computing

This is the first of two chapters that provide an overview of introductory cloud computing topics. It begins with a brief history of cloud computing along with short descriptions of its business and technology drivers. This is followed by definitions of basic concepts and terminology, in addition to explanations of the primary benefits and challenges of cloud computing adoption.

3.1 Origins and Influences

A Brief History

The idea of computing in a "cloud" traces back to the origins of utility computing, a concept that computer scientist John McCarthy publicly proposed in 1961:

> *"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility.… The computer utility could become the basis of a new and important industry."*

In 1969, Leonard Kleinrock, a chief scientist of the Advanced Research Projects Agency Network (ARPANET) project that seeded the internet, stated:

> *"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities'…."*

The general public has been leveraging forms of internet-based computer utilities since the mid-1990s through various incarnations of search engines, email services, open publishing platforms, and other types of social media. Though consumer-centric, these services popularized and validated core concepts that form the basis of modern-day cloud computing.

In 1999, Salesforce.com pioneered the notion of bringing remotely provisioned services into the enterprise. In 2006, Amazon.com launched the Amazon Web Services (AWS) platform, a suite of enterprise-oriented services that provide remotely provisioned storage, computing resources, and business functionality.

A slightly different evocation of the term "network cloud" or "cloud" was introduced in the early 1990s throughout the networking industry. It referred to an abstraction layer derived from the methods for delivering data across heterogeneous public and semi-public networks that were primarily packet-switched, although cellular networks used the "cloud" term as well. The networking method at this point supported the transmission of data from one endpoint (local network) to the "cloud" (wide area network), with the data then being further decomposed to another intended endpoint. This is relevant, as the networking industry still references the use of this term and is considered an early adopter of the concepts that underlie utility computing.

It wasn't until 2006 that the term "cloud computing" emerged in the commercial arena. It was during this time that Amazon launched its Elastic Compute Cloud (EC2) services, which enabled organizations to "lease" computing capacity and processing power to run their enterprise applications. Google Apps also began providing browser-based enterprise applications in the same year, and three years later, the Google App Engine became another historic milestone.

Definitions

A Gartner report listing cloud computing at the top of its strategic technology areas further reaffirmed its prominence as an industry trend by announcing its formal definition as:

> *"…a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies."*

This is a slight revision of Gartner's original definition from 2008, in which "massively scalable" was used instead of "scalable and elastic." This change acknowledges the importance of scalability in relation to the ability to scale vertically, and not just to enormous proportions.

Forrester Research provided its own definition of cloud computing as:

> *"…a standardized IT capability (services, software, or infrastructure) delivered via Internet technologies in a pay-per-use, self-service way."*

The definition that received industry-wide acceptance was composed by the National Institute of Standards and Technology (NIST). NIST published its original definition in 2009, followed by a revised version after further review and industry input that was published in September of 2011:

> *"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 is composed of five essential characteristics, three service models, and four deployment models."*

This book provides a more concise definition:

> *"Cloud computing is a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured resources."*

This simplified definition is in line with all the preceding definition variations that were put forth by other organizations within the cloud computing industry. The characteristics, service models, and deployment models referenced in the NIST definition are further covered in Chapter 4.

Business Drivers

Before delving into the layers of technologies that underlie clouds, the motivations that led to their creation by industry leaders must first be understood. Several of the primary business drivers that fostered modern cloud-based technology are presented in this section.

The origins and inspirations of many of the characteristics, models, and mechanisms covered throughout subsequent chapters can be traced back to the upcoming business drivers. It is important to note that these influences shaped clouds and the overall cloud computing market from both ends. They have motivated organizations to adopt cloud computing in support of their business automation requirements. They have correspondingly motivated other organizations to become providers of cloud environments and cloud technology vendors to create demand and fulfill consumer needs.

Cost Reduction

A direct alignment between IT costs and business performance can be difficult to maintain. The growth of IT environments often corresponds to the assessment of their maximum usage requirements. This can make the support of new and expanded business automations an ever-increasing investment. Much of this required investment is funneled into infrastructure expansion because the usage potential of a given automation solution will always be limited by the processing power of its underlying infrastructure.

Two costs need to be accounted for: the cost of acquiring new infrastructure and the cost of its ongoing ownership. Operational overhead represents a considerable share of IT budgets, often exceeding up-front investment costs.

Common forms of infrastructure-related operating overhead include the following:

- technical personnel required to keep the environment operational
- upgrades and patches that introduce additional testing and deployment cycles
- utility bills and capital expense investments for power and cooling
- security and access control measures that need to be maintained and enforced to protect infrastructure resources
- administrative and accounts staff that may be required to keep track of licenses and support arrangements

The ongoing ownership of internal technology infrastructure can encompass burdensome responsibilities that impose compound impacts on corporate budgets. An IT department can consequently become a significant—and at times overwhelming—drain on the business, potentially inhibiting its responsiveness, profitability, and overall evolution.

Business Agility

Businesses need the ability to adapt and evolve to successfully face change caused by both internal and external factors. Business agility (or organizational agility) is the measure of an organization's responsiveness to change.

An IT enterprise often needs to respond to business change by scaling its IT resources beyond the scope of what was previously predicted or planned for. For example, infrastructure may be subject to limitations that prevent the organization from responding to usage fluctuations—even when they are anticipated—if previous capacity planning efforts were restricted by inadequate budgets.

In other cases, changing business needs and priorities may require IT resources to be more available and reliable than before. Even if sufficient infrastructure is in place for an organization to support anticipated usage volumes, the nature of the usage may generate runtime exceptions that bring down hosting servers. Due to a lack of reliability controls within the infrastructure, responsiveness to consumer or customer requirements may be reduced to a point whereby a business's overall continuity is threatened.

On a broader scale, the up-front investments and infrastructure ownership costs required to implement new or expanded business automation solutions may themselves be prohibitive enough for a business to settle for an IT infrastructure of less-than-ideal quality, thereby decreasing its ability to meet real-world requirements.

Worse yet, the business may decide against proceeding with an automation solution altogether upon review of its infrastructure budget, because it simply cannot afford to. This form of inability to respond can inhibit an organization from keeping up with market demands, competitive pressures, and its own strategic business goals.

Technology Innovations

Established technologies are often used as inspiration and, at times, the actual foundations upon which new technology innovations are derived and built. This section briefly describes the preexisting technologies considered to be the primary influences on cloud computing.

Clustering

A cluster is a group of independent IT resources that are interconnected and work as a single system. System failure rates are reduced while availability and reliability are increased, since redundancy and failover features are inherent to the cluster.

A general prerequisite of hardware clustering is that its component systems have reasonably identical hardware and operating systems to provide similar performance levels when one failed component is to be replaced by another. Component devices that form a cluster are kept in synchronization through dedicated, high-speed communication links.

The basic concept of built-in redundancy and failover is core to cloud platforms. Clustering technology is explored further in Chapter 9 as part of the resource cluster mechanism description.

Grid Computing

A computing grid (or "computational grid") provides a platform in which computing resources are organized into one or more logical pools. These pools are collectively coordinated to provide a high-performance distributed grid, sometimes referred to as a "super virtual computer." Grid computing differs from clustering in that grid systems are much more loosely coupled and distributed. As a result, grid computing systems can involve computing resources that are heterogeneous and geographically dispersed, which is generally not possible with cluster computing–based systems.

Grid computing has been an ongoing research area in computing science since the early 1990s. The technological advancements achieved by grid computing projects have influenced various aspects of cloud computing platforms and mechanisms, specifically in relation to common feature sets such as networked access, resource pooling, and scalability and resiliency. These types of features can be established by both grid computing and cloud computing, using their own distinctive approaches.

For example, grid computing is based on a middleware layer that is deployed on computing resources. These IT resources participate in a grid pool that implements a series of workload distribution and coordination functions. This middle tier can contain load balancing logic, failover controls, and autonomic configuration management, each having previously inspired similar—and sometimes more sophisticated—cloud computing technologies. It is for this reason that some classify cloud computing as a descendant of earlier grid computing initiatives.

Capacity Planning

Capacity planning is the process of determining and fulfilling future demands of an organization's IT resources, products, and services. Within this context, *capacity* represents the maximum amount of work that an IT resource is capable of delivering in a given period of time. A discrepancy between the capacity of an IT resource and its demand can result in a system becoming either inefficient (over-provisioning) or unable to fulfill user needs (under-provisioning). Capacity planning is focused on minimizing this discrepancy to achieve predictable efficiency and performance.

Different capacity planning strategies exist:

- *Lead Strategy* – adding capacity to an IT resource in anticipation of demand
- *Lag Strategy* – adding capacity when the IT resource reaches its full capacity
- *Match Strategy* – adding IT resource capacity in small increments as demand increases

Planning for capacity can be challenging because it requires estimating usage load fluctuations. There is a constant need to balance peak usage requirements without unnecessary over-expenditure on infrastructure. An example is outfitting IT infrastructure to accommodate maximum usage loads, which can impose unreasonable financial investments. In such cases, moderating investments can result in under-provisioning, leading to transaction losses and other usage limitations from lowered usage thresholds.

Virtualization

Virtualization is the process of converting a physical IT resource into a virtual IT resource.

Most types of IT resources can be virtualized, including:

- *Servers* – A physical server can be abstracted into a virtual server.
- *Storage* – A physical storage device can be abstracted into a virtual storage device or a virtual disk.
- *Network* – Physical routers and switches can be abstracted into logical network fabrics, such as VLANs.
- *Power* – A physical UPS and power distribution units can be abstracted into what are commonly referred to as virtual UPSs.

NOTE

The terms *virtual server* and *virtual machine (VM)* are used synonymously throughout this book.

A layer of virtualization software allows physical IT resources to provide multiple virtual images of themselves so that their underlying processing capabilities can be shared by multiple users.

The first step in creating a new virtual server through virtualization software is the allocation of physical IT resources, followed by the installation of an operating system. Virtual servers use their own guest operating systems, which are independent of the operating system in which they were created.

Both the guest operating system and the application software running on the virtual server are unaware of the virtualization process, meaning these virtualized IT resources are installed and executed as if they were running on a separate physical server. This uniformity of execution that allows programs to run on physical systems as they would on virtual systems is a vital characteristic of virtualization. Guest operating systems typically require seamless usage of software products and applications that do not need to be customized, configured, or patched to run in a virtualized environment.

Virtualization software runs on a physical server called a *host* or *physical host*, whose underlying hardware is made accessible by the virtualization software. The virtualization software functionality encompasses system services that are specifically related to

virtual machine management and not normally found on standard operating systems. This is why this software is sometimes referred to as a virtual machine manager or a virtual machine monitor (VMM)—though it is most commonly known as a *hypervisor*. (The hypervisor is formally described as a cloud computing mechanism in Chapter 8.)

Prior to the advent of virtualization technologies, software was limited to residing on and being coupled with static hardware environments. The virtualization process severs this software-hardware dependency, as hardware requirements can be simulated by emulation software running in virtualized environments.

Established virtualization technologies can be traced to several cloud characteristics and cloud computing mechanisms, which inspired many of their core features. As cloud computing evolved, a new generation of *modern* virtualization technologies emerged to overcome the performance, reliability, and scalability limitations of traditional virtualization platforms. Modern virtualization technologies are discussed in Chapter 5.

Containerization

Containerization is a form of virtualization technology that allows for the creation of virtual hosting environments referred to as "containers" without the need to deploy a virtual server for each solution. A container is similar in concept to a virtual server in that it provides a virtual environment with operating system resources that can be used to host software programs and other IT resources.

Containers are briefly introduced in the upcoming *Basic Concepts and Terminology* section, and containerization technology is covered in detail in Chapter 6.

Serverless Environments

A serverless environment is a special operational runtime environment that does not require developers or system administrators to deploy or provision servers. Instead, it is equipped with technology that allows for the deployment of special software packages that already include the required server components and configuration information.

Upon deployment, the serverless environment automatically implements and activates an application deployment together with its packaged server, without the administrator having to do anything further. Programs are designed, coded, and deployed alongside the descriptor of the underlying required runtime and any dependencies that may exist. Once deployed, the serverless environment can run and scale the application and ensure its ongoing availability and scalability.

Contemporary software architectures deployed in clouds can benefit greatly from serverless environments. More details on serverless technology are provided in Chapter 5.

3.2 Basic Concepts and Terminology

This section establishes a set of basic terms that represent the fundamental concepts and aspects pertaining to the notion of a cloud and its most primitive artifacts.

Cloud

A *cloud* refers to a distinct IT environment that is designed for the purpose of remotely provisioning scalable and measured IT resources. This term originated as a metaphor for the internet, which is, in essence, a network of networks providing remote access to a set of decentralized IT resources. Prior to cloud computing becoming its own formalized IT industry segment, the symbol of a cloud was commonly used to represent the internet in a variety of specifications and mainstream documentation of web-based architectures. This same symbol is now used to specifically represent the boundary of a cloud environment, as shown in Figure 3.1.

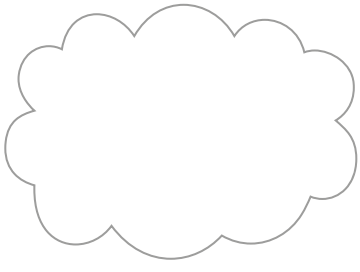


Figure 3.1
The symbol used to denote the boundary of a cloud environment.

It is important to distinguish the term "cloud" and the cloud symbol from the internet. As a specific environment used to remotely provision IT resources, a cloud has a finite boundary. There are many individual clouds that are accessible via the internet. Whereas the internet provides open access to many web-based IT resources, a cloud is typically privately owned and offers access to IT resources that is metered.

Much of the internet is dedicated to the access of content-based IT resources published via the World Wide Web. IT resources provided by cloud environments, on the other hand, are dedicated to supplying back-end processing capabilities and user-based

access to these capabilities. Another key distinction is that it is not necessary for clouds to be web-based, even if they are commonly based on internet protocols and technologies. Protocols refer to standards and methods that allow computers to communicate with each other in a predefined and structured manner. A cloud can be based on the use of any protocols that allow for remote access to its IT resources.

NOTE

Diagrams in this book depict the internet using the globe symbol.

Container

Containers (Figure 3.2) are commonly used in clouds to provide highly optimized virtual hosting environments capable of providing only the resources required for the software programs they host.

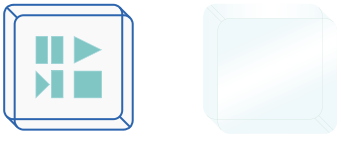


Figure 3.2
The figure on the left is the general symbol used to represent a container. The figure on the right (with rounded edges) is used in architectural diagrams to represent a container, especially when the contents of the container need to be shown.

IT Resource

An *IT resource* is a physical or virtual IT-related artifact that can be either software-based, such as a virtual server or a custom software program, or hardware-based, such as a physical server or a network device (Figure 3.3).

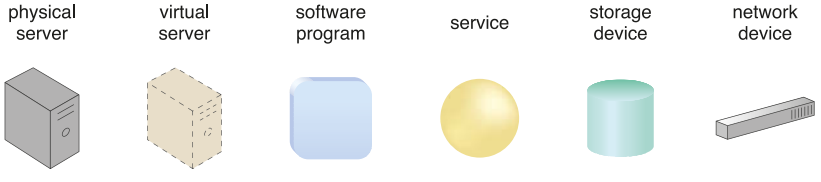


Figure 3.3
Examples of common IT resources and their corresponding symbols.

Figure 3.4 illustrates how the cloud symbol can be used to define a boundary for a cloud-based environment that hosts and provisions a set of IT resources. The displayed IT resources are consequently considered to be cloud-based IT resources.

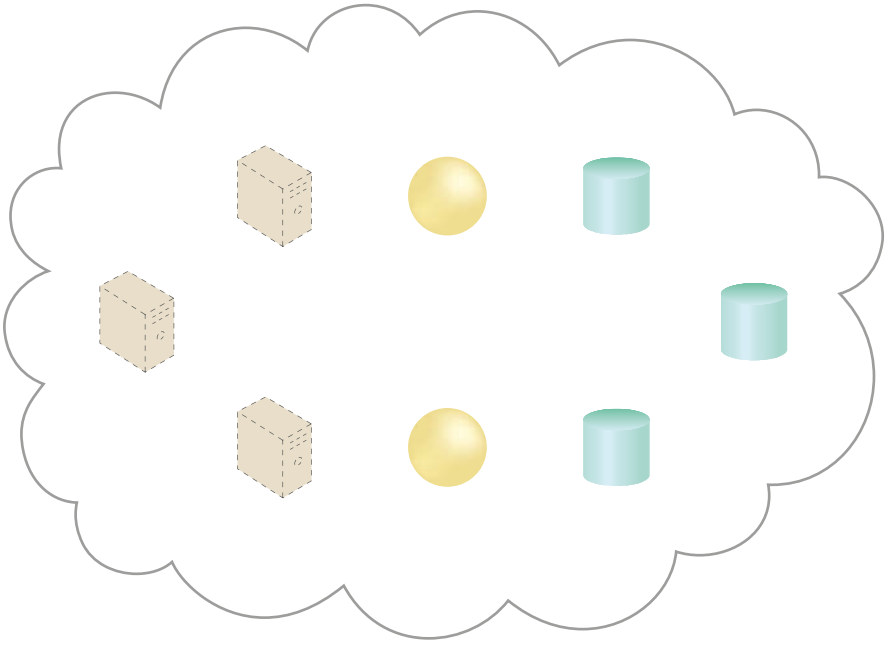


Figure 3.4
A cloud is hosting eight IT resources: three virtual servers, two cloud services, and three storage devices.

Technology architectures and various interaction scenarios involving IT resources are illustrated in diagrams like the one shown in Figure 3.4. It is important to note the following points when studying and working with these diagrams:

- The IT resources shown within the boundary of a given cloud symbol usually do not represent all the available IT resources hosted by that cloud. Subsets of IT resources are generally highlighted for the purpose of demonstrating a particular topic.
- Focusing on the relevant aspects of a topic requires many of these diagrams to intentionally provide abstracted views of the underlying technology architectures. For this reason, only a portion of the actual technical details are shown.

Furthermore, some diagrams will display IT resources outside of the cloud symbol. This convention is used to indicate IT resources that are not cloud-based.

NOTE

The virtual server IT resource displayed in Figure 3.3 is further discussed in Chapters 5 and 8. Physical servers are sometimes referred to as *physical hosts* (or just *hosts*) in reference to the fact that they are responsible for hosting virtual servers.

On Premises

As a distinct and remotely accessible environment, a cloud represents an option for the deployment of IT resources. An IT resource that is hosted in a conventional IT enterprise within an organizational boundary (that does not specifically represent a cloud) is considered to be located on the premises of the IT enterprise. Therefore, for clarification purposes in this book, an IT resource that is on premises cannot be cloud-based, and vice versa.

Note the following key points:

- An on-premises IT resource can access and interact with a cloud-based IT resource.
- An on-premises IT resource can be moved to a cloud, thereby changing it to a cloud-based IT resource.
- Redundant deployments of an IT resource can exist both on premises and in cloud-based environments.

If the distinction between on-premises and cloud-based IT resources is confusing in relation to private clouds (described in the *Cloud Deployment Models* section of Chapter 4), then an alternative qualifier can be used.

Cloud Consumers and Cloud Providers

The party that provides cloud-based IT resources is the *cloud provider*. The party that uses cloud-based IT resources is the *cloud consumer*. These terms represent roles usually assumed by organizations in relation to clouds and corresponding cloud provisioning contracts. These roles are formally defined in Chapter 4, as part of the *Roles and Boundaries* section.

Scaling

Scaling, from an IT resource perspective, represents the ability of the IT resource to handle increased or decreased usage demands.

The following are types of scaling:

- *Horizontal Scaling* – scaling out and scaling in
- *Vertical Scaling* – scaling up and scaling down

The next two sections briefly describe each type.

Horizontal Scaling

The allocating or releasing of IT resources that are of the same type is referred to as *horizontal scaling* (Figure 3.5). The horizontal allocation of resources is referred to as *scaling out* and the horizontal releasing of resources is referred to as *scaling in*. Horizontal scaling is a common form of scaling within cloud environments.

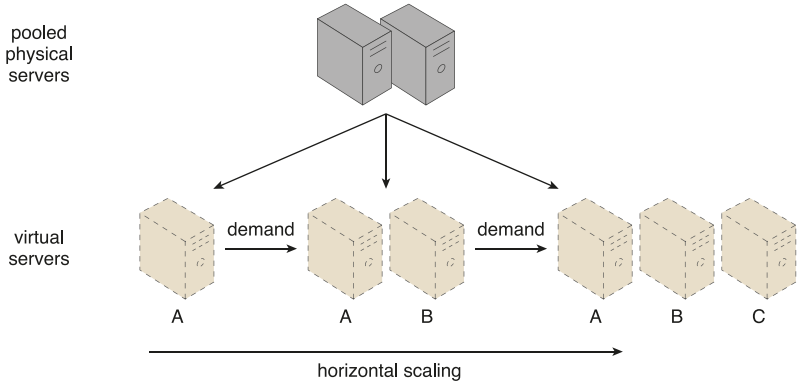


Figure 3.5

An IT resource (Virtual Server A) is scaled out by adding more of the same IT resources (Virtual Servers B and C).

Vertical Scaling

When an existing IT resource is replaced by another with higher or lower capacity, *vertical scaling* is considered to have occurred (Figure 3.6). Specifically, the replacement of an IT resource with another that has a higher capacity is referred to as *scaling up* and the replacement of an IT resource with another that has a lower capacity is considered *scaling down*. Vertical scaling is less common in cloud environments due to the downtime required while the replacement is taking place.

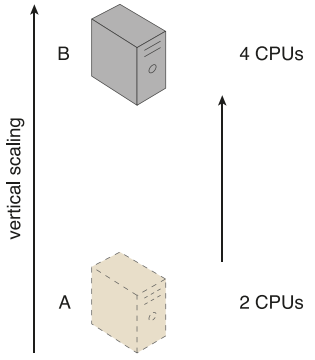


Figure 3.6
An IT resource (a virtual server with two CPUs) is scaled up by replacing it with a more powerful IT resource with increased capacity for data storage (a physical server with four CPUs).

Table 3.1 provides a brief overview of common pros and cons associated with horizontal and vertical scaling.

Horizontal Scaling	Vertical Scaling
less expensive (through commodity hardware components)	more expensive (specialized servers)
IT resources instantly available	IT resources normally instantly available
resource replication and automated scaling	additional setup is normally needed
additional IT resources needed	no additional IT resources needed
not limited by hardware capacity	limited by maximum hardware capacity

Table 3.1
A comparison of horizontal and vertical scaling.

Cloud Service

Although a cloud is a remotely accessible environment, not all IT resources residing within a cloud can be made available for remote access. For example, a database or a physical server deployed within a cloud may only be accessible by other IT resources that are within the same cloud. A software program with a published API may be deployed specifically to enable access by remote clients.

A *cloud service* is any IT resource that is made remotely accessible via a cloud. Unlike other IT fields that fall under the service technology umbrella—such as service-oriented architecture—the term "service" within the context of cloud computing is especially broad. A cloud service can exist as a simple web-based software program with a technical interface invoked via the use of a messaging protocol, or as a remote access point for administrative tools or larger environments and other IT resources.

In Figure 3.7, the yellow circle symbol is used to represent the cloud service as a simple web-based software program. A different IT resource symbol may be used in the latter case, depending on the nature of the access that is provided by the cloud service.

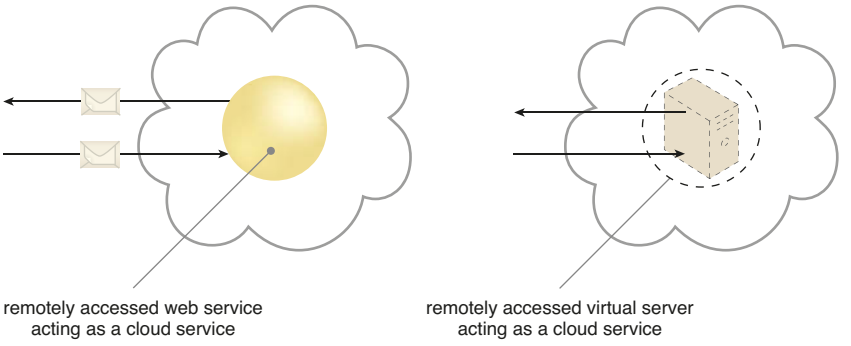


Figure 3.7

A cloud service with a published technical interface is being accessed by a consumer outside of the cloud (left). A cloud service that exists as a virtual server is also being accessed from outside of the cloud's boundary (right). The cloud service on the left is likely being invoked by a consumer program that was designed to access the cloud service's published technical interface. The cloud service on the right may be accessed by a human user that has remotely logged on to the virtual server.

The driving motivation behind cloud computing is to provide IT resources as services that encapsulate other IT resources, while offering functions for clients to use and leverage remotely. A multitude of models for generic types of cloud services have emerged, most of which are labeled with the "as a service" suffix.

NOTE

Cloud service usage conditions are typically expressed in a service-level agreement (SLA), which is the human-readable part of a service contract between a cloud provider and cloud consumer that describes quality of service (QoS) features, behaviors, and limitations of a cloud-based service or other provisions.

An SLA provides details of various measurable characteristics related to IT outcomes, such as uptime, security characteristics, and other specific QoS features, including availability, reliability, and performance. Since the implementation of a service is hidden from the cloud consumer, an SLA becomes a critical specification. SLAs are covered in detail in Chapter 18.

Cloud Service Consumer

The *cloud service consumer* is a temporary runtime role assumed by a software program when it accesses a cloud service.

As shown in Figure 3.8, common types of cloud service consumers can include software programs and services capable of remotely accessing cloud services with published service contracts, as well as workstations, laptops, and mobile devices running software capable of remotely accessing other IT resources positioned as cloud services.

Figure 3.8

Examples of cloud service consumers. Depending on the nature of a given diagram, an artifact labeled as a cloud service consumer may be a software program or a hardware device (in which case it is implied that it is running a software program capable of acting as a cloud service consumer).

3.3 Goals and Benefits

The common benefits associated with adopting cloud computing are explained in this section.

NOTE

The following sections make reference to the terms "public cloud" and "private cloud." These terms are described in the *Cloud Deployment Models* section in Chapter 4.

Increased Responsiveness

Cloud computing plays an essential role in enhancing an organization's business agility by empowering it to be more responsive to business and usage scenarios that can be more effectively addressed by leveraging native cloud capabilities, such as on-demand scalability, data availability, reduced infrastructure maintenance, reduced business complexity, automation, and increased uptime.

For example, greater data availability can enable employees to more easily work remotely, thereby providing staff with increased flexibility and productivity.

Utilizing platforms maintained by cloud providers frees organizations from the responsibilities they would normally have for administering those platforms internally. This can also reduce the complexity of their infrastructure environments, thereby introducing novel ways for employees to collaborate and work through faster and less complex technology rollouts for new business initiatives.

Ultimately, cloud computing empowers organizations to become significantly more responsive by removing or reducing many organizational burdens associated with business solution development, deployment, and maintenance, and by reducing time-to-market timelines.

Reduced Investments and Proportional Costs

Similar to a product wholesaler that purchases goods in bulk at lower price points, public cloud providers base their business model on the mass-acquisition of IT resources that are then made available to cloud consumers via attractively priced leasing packages. This opens the door for organizations to gain access to powerful infrastructure without having to purchase it themselves.

The most common economic rationale for investing in cloud-based IT resources is in the reduction or outright elimination of up-front IT investments—namely, hardware and software purchases and ownership costs. A cloud's Measured Usage characteristic represents a feature set that allows measured operational expenditures (directly related to business performance) to replace anticipated capital expenditures. This is also referred to as *proportional costs*.

This elimination or minimization of up-front financial commitments allows enterprises to start small and accordingly increase IT resource allocation as required. Moreover, the reduction of up-front capital expenses allows for the capital to be redirected to the

core business investment. In its most basic form, opportunities to decrease costs are derived from the deployment and operation of large-scale data centers by major cloud providers. Such data centers are commonly located in destinations where real estate, IT professionals, and network bandwidth can be obtained at lower costs, resulting in both capital and operational savings.

The same rationale applies to operating systems, middleware or platform software, and application software. Pooled IT resources are made available to and shared by multiple cloud consumers, resulting in increased or even maximum possible utilization. Operational costs and inefficiencies can be further reduced by applying proven practices and patterns for optimizing cloud architectures, their management, and their governance.

Common measurable benefits to cloud consumers include:

- On-demand access to pay-as-you-go computing resources on a short-term basis (such as processors by the hour), and the ability to release these computing resources when they are no longer needed.
- The perception of having unlimited computing resources that are available on-demand, thereby reducing the need to prepare for provisioning.
- The ability to add or remove IT resources at a fine-grained level, such as modifying available storage disk space by single gigabyte increments.
- Abstraction of the infrastructure so applications are not locked into devices or locations and can be easily moved if needed.

For example, a company with sizable batch-centric tasks can complete them as quickly as their application software can scale. Using 100 servers for one hour costs the same as using one server for 100 hours. This "elasticity" of IT resources, achieved without requiring steep initial investments to create a large-scale computing infrastructure, can be extremely compelling.

Despite the ease with which many identify the financial benefits of cloud computing, the actual economics can be complex to calculate and assess. The decision to proceed with a cloud computing adoption strategy will involve much more than a simple comparison between the cost of leasing and the cost of purchasing. For example, the financial benefits of dynamic scaling and the risk transference of both over-provisioning (under-utilization) and under-provisioning (over-utilization) must also be accounted for. Chapter 17 explores common criteria and formulas for performing detailed financial comparisons and assessments.

NOTE

Another area of cost savings offered by clouds is the "as a service" usage model, whereby technical and operational implementation details of IT resource provisioning are abstracted from cloud consumers and packaged into "ready-to-use" or "off-the-shelf" solutions. These services-based products can simplify and expedite the development, deployment, and administration of IT resources when compared to performing equivalent tasks with solutions on premises. The resulting savings in time and required IT expertise can be significant and can help justify the adoption of cloud computing.

Increased Scalability

By providing pools of IT resources, along with tools and technologies designed to leverage them collectively, clouds can instantly and dynamically allocate IT resources to cloud consumers, either on-demand or via the cloud consumer's direct configuration. This empowers cloud consumers to scale their cloud-based IT resources to accommodate processing fluctuations and peaks automatically or manually. Similarly, cloud-based IT resources can be released (automatically or manually) as processing demands decrease. A simple example of usage demand fluctuations throughout a 24-hour period is provided in Figure 3.9.

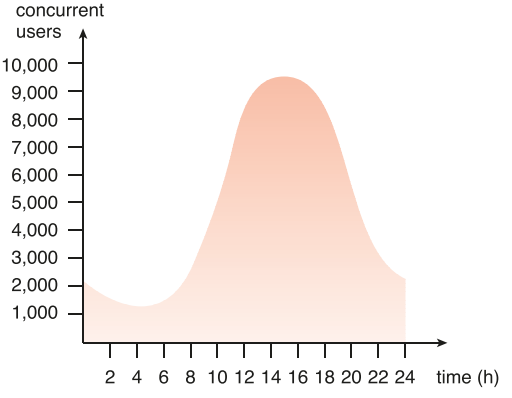


Figure 3.9
An example of an organization's changing demand for an IT resource over the course of a day.

The inherent, built-in feature of clouds to provide flexible levels of scalability to IT resources is directly related to the aforementioned proportional costs benefit. Besides the evident financial gain to the automated reduction of scaling, the ability of IT resources to always meet and fulfill unpredictable usage demands avoids potential loss of business that can occur when usage thresholds are met.

NOTE

When associating the benefit of Increased Scalability with the capacity planning strategies introduced earlier in the *Business Drivers* section, the lag and match strategies are generally more applicable due to a cloud's ability to scale IT resources on-demand.

Increased Availability and Reliability

The availability and reliability of IT resources are directly associated with tangible business benefits. Outages limit the time an IT resource can be "open for business" for its customers, thereby limiting its usage and revenue-generating potential. Runtime failures that are not immediately corrected can have a more significant impact during high-volume usage periods. Not only is the IT resource unable to respond to customer requests, but its unexpected failure can decrease overall customer confidence.

A hallmark of the typical cloud environment is its intrinsic ability to provide extensive support for increasing the availability of a cloud-based IT resource to minimize or even eliminate outages, and for increasing its reliability so as to minimize the impact of runtime failure conditions.

Specifically:

- An IT resource with increased availability is accessible for longer periods of time (for example, 22 hours out of a 24-hour day). Cloud providers generally offer "resilient" IT resources for which they are able to guarantee high levels of availability.
- An IT resource with increased reliability is able to better avoid and recover from exception conditions. The modular architecture of cloud environments provides extensive failover support that increases reliability.

It is important that organizations carefully examine the SLAs offered by cloud providers when considering the leasing of cloud-based services and IT resources. Although many cloud environments are capable of offering remarkably high levels of availability and reliability, it comes down to the guarantees made in the SLA that typically represent their actual contractual obligations.

3.4 Risks and Challenges

Several of the most critical cloud computing challenges are presented and examined in this section.

Increased Vulnerability Due to Overlapping Trust Boundaries

Moving business data to the cloud means that the organization's responsibility over data security is shared with the cloud provider. The remote usage of IT resources requires an expansion of trust boundaries by the cloud consumer to include the cloud, external to the organization. It can be difficult to establish a security architecture that spans such a trust boundary without introducing vulnerabilities unless cloud consumers and cloud providers happen to support the same or compatible security frameworks, which is improbable with public clouds.

Another consequence of overlapping trust boundaries relates to the cloud provider's privileged access to cloud consumer data. The extent to which the data is secure is now limited to the security controls and policies applied by both the cloud consumer and the cloud provider. Furthermore, there can be overlapping trust boundaries from different cloud consumers because cloud-based IT resources are commonly shared.

The overlapping of trust boundaries and the increased exposure of data can provide malicious cloud consumers (human and automated) with greater opportunities to attack IT resources and steal or damage business data. Figure 3.10 illustrates a scenario in which two organizations accessing the same cloud service are required to extend their respective trust boundaries to the cloud, resulting in overlapping trust boundaries. Cloud providers are required to offer security mechanisms that accommodate the security requirements of both cloud service consumers.

Overlapping trust boundaries is a security threat that is discussed in more detail in Chapter 7.

Increased Vulnerability Due to Shared Security Responsibility

Information security related to on-premises resources is clearly the responsibility of the organization that owns those resources. However, information security related to cloud-based resources is not the sole responsibility of the cloud provider, even if the cloud-based resources are owned by the cloud provider. This is because the information stored and processed in them is owned by the cloud consumer.

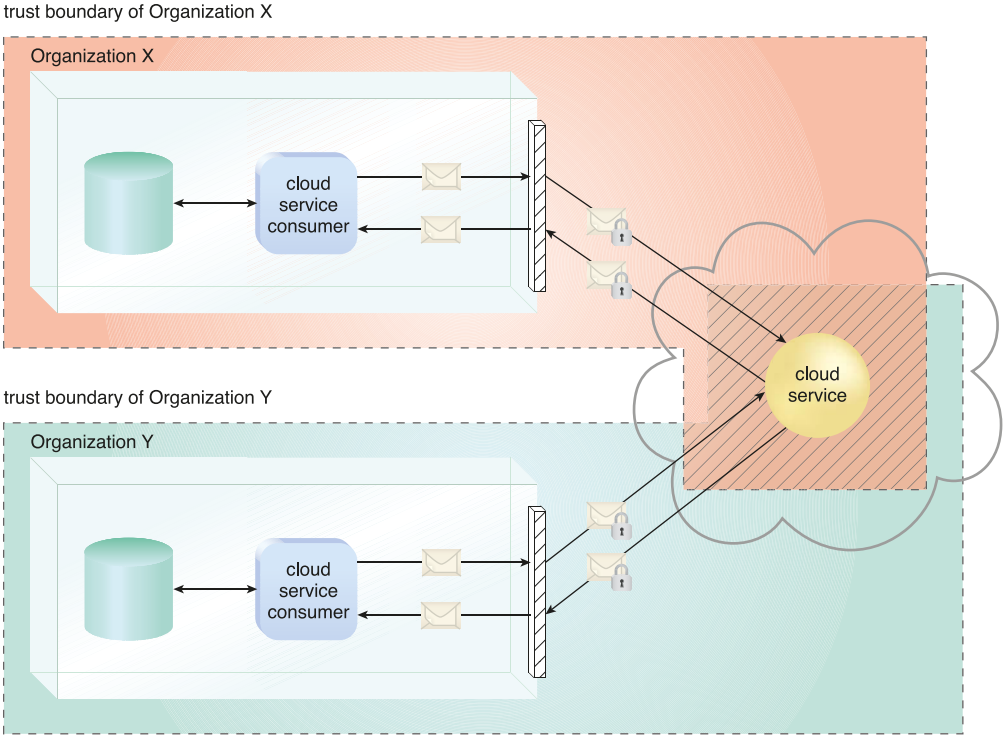


Figure 3.10
The shaded area with diagonal lines indicates the overlap of two organizations' trust boundaries.

As a result, information security in the cloud is a shared responsibility, with both the cloud provider and the cloud consumer having a role to play in securing the cloud environment. It is important to be able to understand and identify where the responsibility for each role begins and ends, as well as to know how to address the security requirements that correspond to the cloud consumer.

A cloud provider will typically propose a cloud shared responsibility model as part of the SLA. This model essentially outlines the respective responsibilities of the cloud provider and the cloud consumer when it comes to securing data and applications in the cloud.

Increased Exposure to Cyber Threats

The increased adoption of contemporary digital technologies and digital transformation practices has led organizations to move more IT resources to and build more solutions within cloud environments. This has opened the door to cybersecurity threats and risks that may be new to organizations and for which they need to be prepared (Figure 3.11).

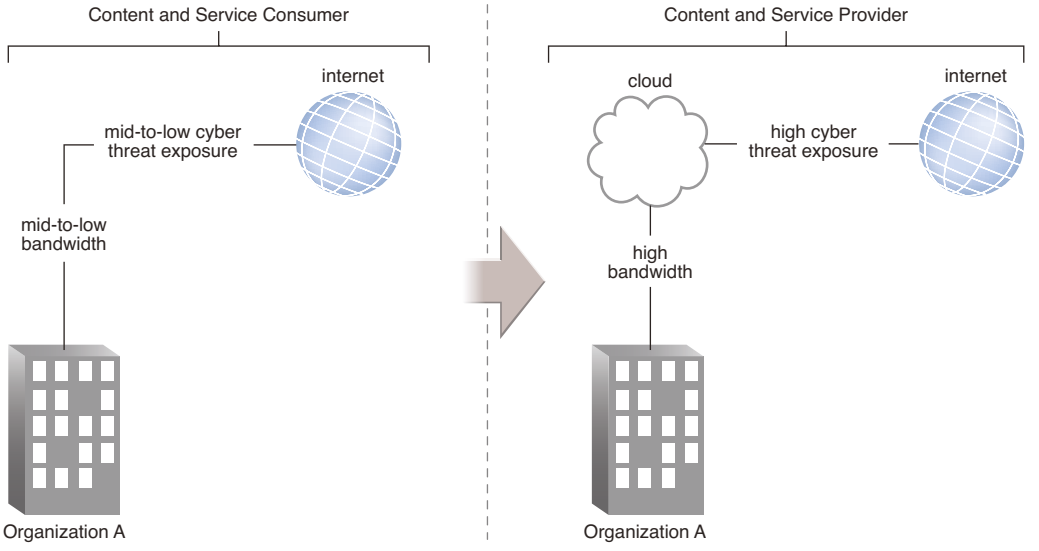


Figure 3.11

An organization that moves from only consuming content and services from the internet to offering its own content and services through the internet increases its exposure to cyber threats.

Augmented exposure to cybersecurity threats due to increased exposure to the internet requires organizations to take action in the protection of their IT assets, both on-premises and cloud-based. Cloud-based IT resources have the benefit of shared responsibility for security and access control, both from the cloud provider and from the cloud consumer. However, the ultimate responsibility lies with the cloud consumer, which needs to address risk management and cybersecurity risks responsibly and methodologically.

Reduced Operational Governance Control

Cloud consumers are usually allotted a level of governance control that is lower than their level of control over their on-premises IT resources. This can introduce risks

associated with how the cloud provider operates its cloud, as well as the external connections that are required for communication between the cloud and the cloud consumer.

Consider the following examples:

- An unreliable cloud provider may not maintain the guarantees it makes in the SLAs that were published for its cloud services. This can jeopardize the quality of the cloud consumer solutions that rely on these cloud services.
- Longer geographic distances between the cloud consumer and the cloud provider can require additional network hops that introduce fluctuating latency and potential bandwidth constraints.

The latter scenario is illustrated in Figure 3.12.

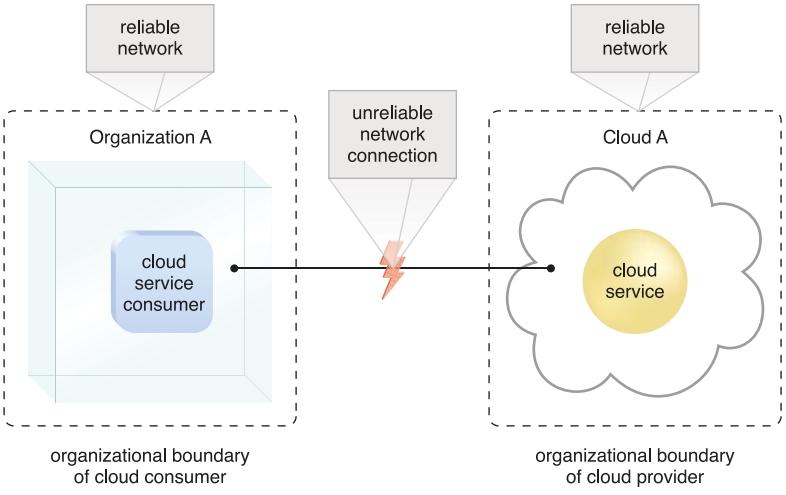


Figure 3.12
An unreliable network connection compromises the quality of communication between the cloud consumer and cloud provider environments.

Legal contracts, when combined with SLAs, technology inspections, and monitoring, can mitigate governance risks and issues. A cloud governance system is established through SLAs, given the "as a service" nature of cloud computing. The cloud consumer must keep track of the actual service level being offered and the other warranties that are made by the cloud provider.

Note that different cloud delivery models offer varying degrees of operational control granted to cloud consumers, as further explained in Chapter 4.

Limited Portability Between Cloud Providers

Due to a lack of established industry standards within the cloud computing industry, public clouds are commonly proprietary to various extents. For cloud consumers that have custom-built solutions with dependencies on these proprietary environments, it can be challenging to move from one cloud provider to another.

Portability is a measure used to determine the impact of moving cloud consumer IT resources and data between clouds (Figure 3.13).

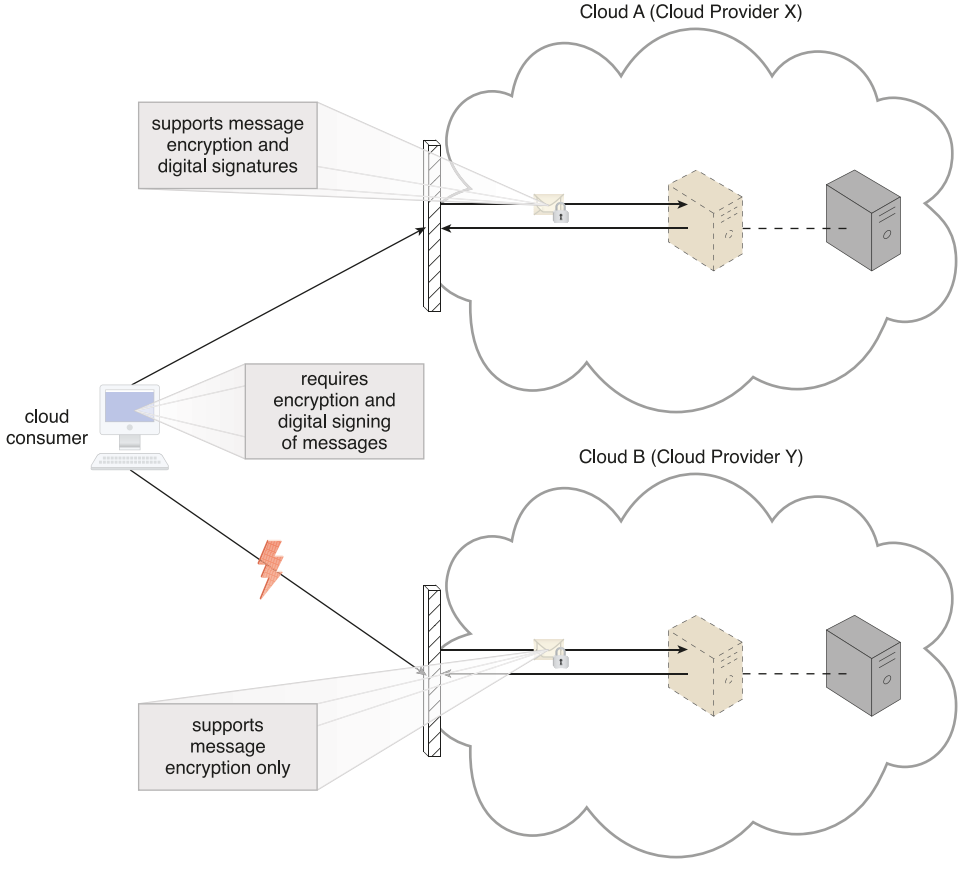


Figure 3.13

A cloud consumer's application has a decreased level of portability when assessing a potential migration from Cloud A to Cloud B, because the cloud provider of Cloud B does not support the same security technologies as Cloud A.

Multiregional Compliance and Legal Issues

Third-party cloud providers will frequently establish data centers in affordable or convenient geographical locations. Cloud consumers will often not be aware of the physical location of their IT resources and data when hosted by public clouds. For some organizations, this can pose serious legal concerns pertaining to industry or government regulations that specify data privacy and storage policies. For example, some UK laws require personal data belonging to UK citizens to be kept within the United Kingdom.

Another potential legal issue pertains to the accessibility and disclosure of data. Countries typically have laws that require some types of data to be disclosed to certain government agencies or to the subject of the data. For example, a European cloud consumer's data that is located in the United States can be more easily accessed by government agencies (due to the USA Patriot Act) when compared to data located in many European Union countries.

Most regulatory frameworks recognize that cloud consumer organizations are ultimately responsible for the security, integrity, and storage of their own data, even when it is held by an external cloud provider.

Cost Overruns

Creating a business case for cloud computing can be a difficult undertaking due to the number of requirements, considerations, and stakeholders that need to be accommodated. Many organizations proceed with cloud migration initiatives without creating a proper business case for those projects. This has been one of the root causes of cost overruns in cloud projects and has led to poor planning, poor or absent governance, and costly cloud risk-mitigation policies.

Traditionally, a business case process is triggered by the need to justify large capital investments. However, with cloud environments that allow users to quickly obtain desired capabilities, organizations can incorrectly assume that they will not require additional capital investments. As cloud adoption then grows, there is eventually a recognition that this operating model requires capital investment beyond the migration itself—but there may not be a method of estimating the amount of the investment necessary prior to adoption or migration.

This page intentionally left blank

Chapter 4

Fundamental Concepts and Models

The upcoming sections cover introductory topic areas pertaining to the fundamental models used to categorize and define clouds and their most common service offerings, along with definitions of organizational roles and the specific set of characteristics that collectively distinguish a cloud.

4.1 Roles and Boundaries

Organizations and humans can assume different types of predefined roles depending on how they relate to and/or interact with a cloud and its hosted IT resources. Each of the upcoming roles participates in and carries out responsibilities in relation to cloud-based activity. The following sections define these roles and identify their main interactions.

Cloud Provider

The organization that provides cloud-based IT resources is the *cloud provider*. When assuming the role of cloud provider, an organization is responsible for making cloud services available to cloud consumers, as per agreed-upon SLA guarantees. The cloud provider is further tasked with any required management and administrative duties to ensure the ongoing operation of the overall cloud infrastructure.

Cloud providers normally own the IT resources that are made available for lease by cloud consumers; however, some cloud providers also "resell" IT resources leased from other cloud providers.

Cloud Consumer

A *cloud consumer* is an organization (or a human) that has a formal contract or arrangement with a cloud provider to use IT resources made available by the cloud provider. Specifically, the cloud consumer uses a cloud service consumer to access a cloud service (Figure 4.1).

The figures in this book do not always explicitly label symbols as "cloud consumers." Instead, it is generally implied that organizations or humans shown remotely accessing cloud-based IT resources are considered cloud consumers.

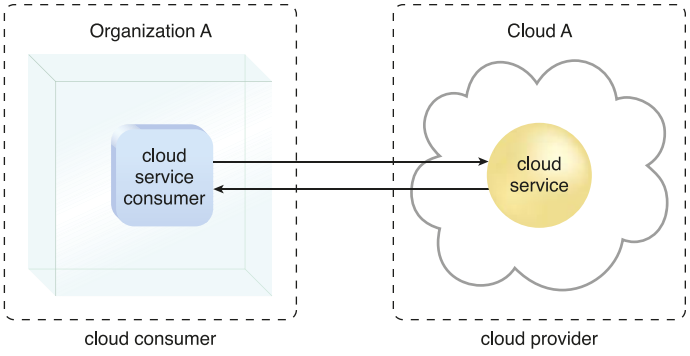


Figure 4.1
A cloud consumer (Organization A) interacts with a cloud service from a cloud provider (that owns Cloud A). Within Organization A, the cloud service consumer is being used to access the cloud service.

NOTE

When depicting interaction scenarios between cloud-based IT resources and consumer organizations, there are no strict rules as to how the terms "cloud service consumer" and "cloud consumer" are used in this book. The former is usually used to label software programs or applications that programmatically interface with a cloud service's technical contract or API. The latter term is broader in that it can be used to label an organization, an individual accessing a user interface, or a software program that assumes the role of cloud consumer when interacting with a cloud, a cloud-based IT resource, or a cloud provider. The broad applicability of the "cloud consumer" term is intentional, as it allows it to be used in figures that explore different types of consumer–provider relationships within different technical and business contexts.

Cloud Broker

A third-party organization that assumes the responsibility of negotiating, managing, and operating cloud services on behalf of a cloud consumer is assuming the role of *cloud broker*. Cloud brokers can provide mediation services between cloud consumers and cloud providers, including intermediation, aggregation, arbitrage, and others.

A cloud broker commonly provides these services for multiple cloud consumers engaging with multiple cloud providers alternatively or simultaneously, acting as an integrator of cloud services and an aggregator of cloud consumers, as shown in Figure 4.2.

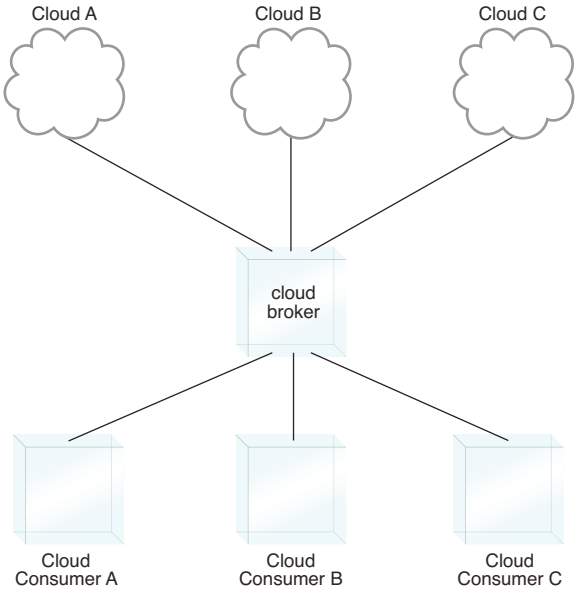


Figure 4.2
A cloud broker offers the cloud-based services and IT resources from three different cloud providers to its customers, Cloud Consumers A, B, and C.

Cloud Service Owner

The person or organization that legally owns a cloud service is called a *cloud service owner*. The cloud service owner can be the cloud consumer, or the cloud provider that owns the cloud within which the cloud service resides.

For example, either the cloud consumer of Cloud X or the cloud provider of Cloud X could own Cloud Service A (Figures 4.3 and 4.4).

Note that a cloud consumer that owns a cloud service hosted by a third-party cloud does not necessarily need to be the user (or consumer) of the cloud service. Several cloud consumer organizations develop and deploy cloud services in clouds owned by other parties for the purpose of making the cloud services available to the general public.

The reason a cloud service owner is not called a cloud resource owner is because the cloud service owner role applies only to cloud services (which, as explained in Chapter 3, are externally accessible IT resources that reside in a cloud).

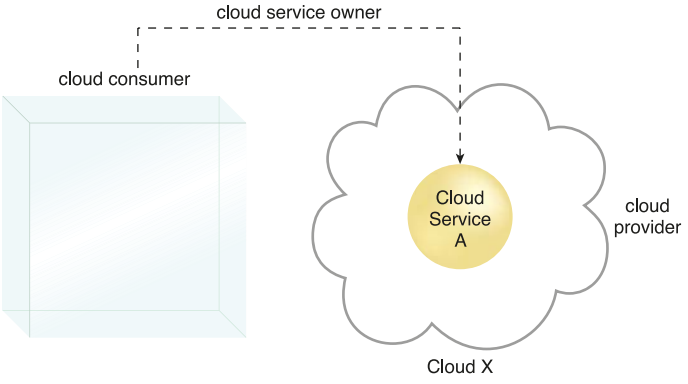


Figure 4.3
A cloud consumer can be a cloud service owner when it deploys its own service in a cloud.

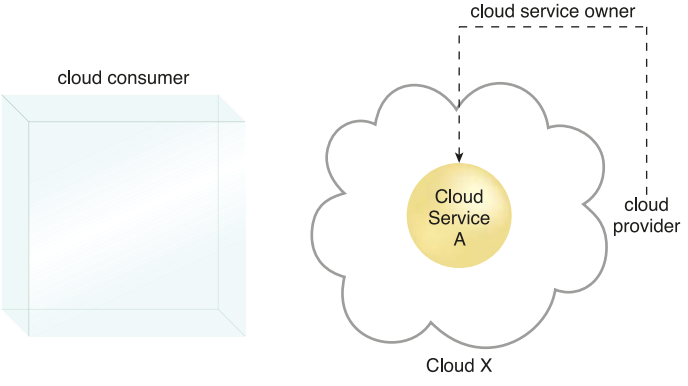


Figure 4.4
A cloud provider becomes a cloud service owner if it deploys its own cloud service, typically for other cloud consumers to use.

Cloud Resource Administrator

A *cloud resource administrator* is the person or organization responsible for administering a cloud-based IT resource (including cloud services). The cloud resource administrator can be (or belong to) the cloud consumer or cloud provider of the cloud within which the cloud service resides. Alternatively, it can be (or belong to) a third-party organization contracted to administer the cloud-based IT resource.

For example, a cloud service owner can contract a cloud resource administrator to administer a cloud service (Figures 4.5 and 4.6).

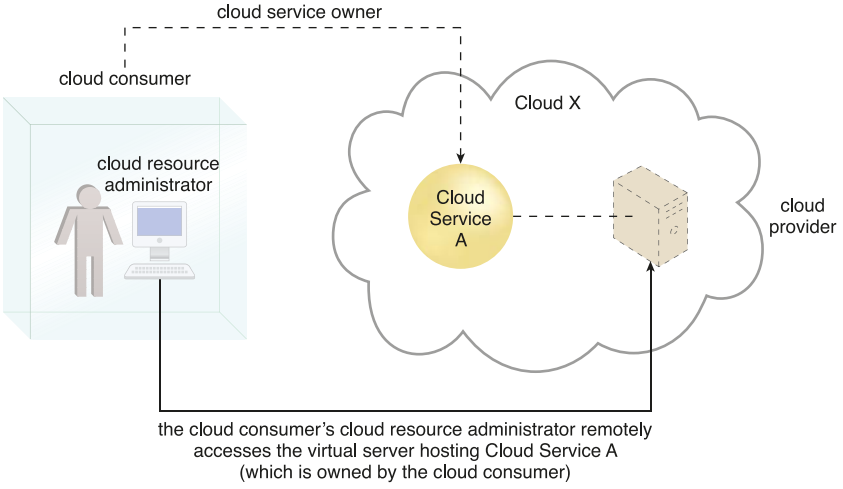


Figure 4.5

A cloud resource administrator can be with a cloud consumer organization and administer remotely accessible IT resources that belong to the cloud consumer.

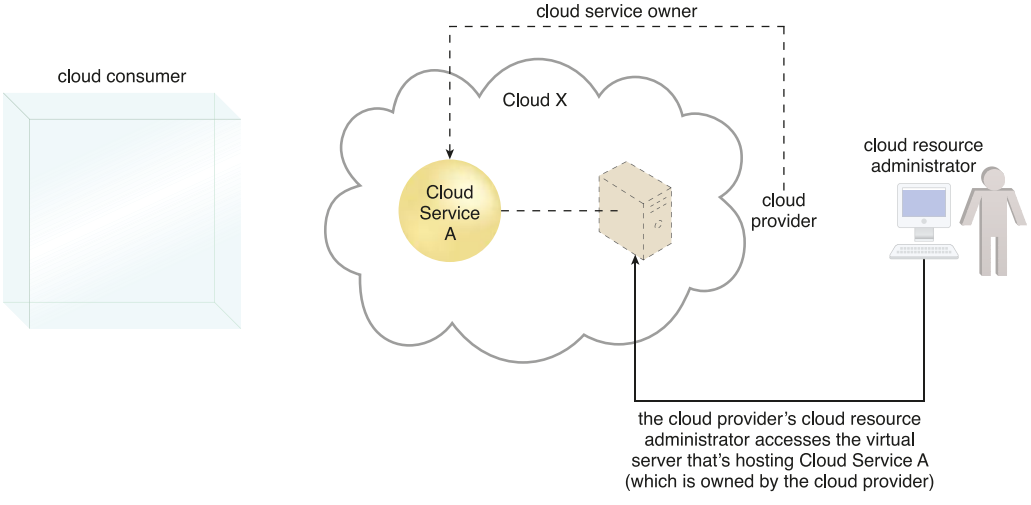


Figure 4.6

A cloud resource administrator can be with a cloud provider organization, for which it can administer the cloud provider's internally and externally available IT resources.

The reason a cloud resource administrator is not referred to as a "cloud service administrator" is because this role may be responsible for administering cloud-based IT resources that don't exist as cloud services. For example, if the cloud resource administrator belongs to (or is contracted by) the cloud provider, IT resources not made remotely accessible may be administered by this role (and these types of IT resources are not classified as cloud services).

Additional Roles

The NIST Cloud Computing Reference Architecture defines the following supplementary roles:

- *Cloud Auditor* – A third party (often accredited) that conducts independent assessments of cloud environments assumes the role of the *cloud auditor*. The typical responsibilities associated with this role include the evaluation of security controls, privacy impacts, and performance. The main purpose of the cloud auditor role is to provide an unbiased assessment (and possible endorsement) of a cloud environment to help strengthen the trust relationship between cloud consumers and cloud providers.
- *Cloud Carrier* – The party responsible for providing the wire-level connectivity between cloud consumers and cloud providers assumes the role of the *cloud carrier*. This role is often assumed by network and telecommunication providers.

While each is legitimate, most architectural scenarios covered in this book do not include these roles.

Organizational Boundary

An *organizational boundary* represents the physical perimeter that surrounds a set of IT resources that are owned and governed by an organization. The organizational boundary does not represent the boundary of an actual organization, only an organizational set of IT assets and IT resources. Similarly, clouds have an organizational boundary (Figure 4.7).

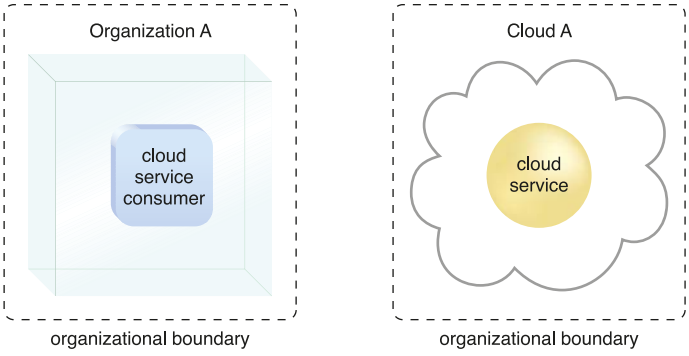


Figure 4.7
Organizational boundaries of a cloud consumer (left) and a cloud provider (right), represented by a broken line notation.

Trust Boundary

When an organization assumes the role of cloud consumer to access cloud-based IT resources, it needs to extend its trust beyond the physical boundary of the organization to include parts of the cloud environment.

A *trust boundary* is a logical perimeter that typically spans beyond physical boundaries to represent the extent to which IT resources are trusted (Figure 4.8). When analyzing cloud environments, the trust boundary is most frequently associated with the trust issued by the organization acting as the cloud consumer.

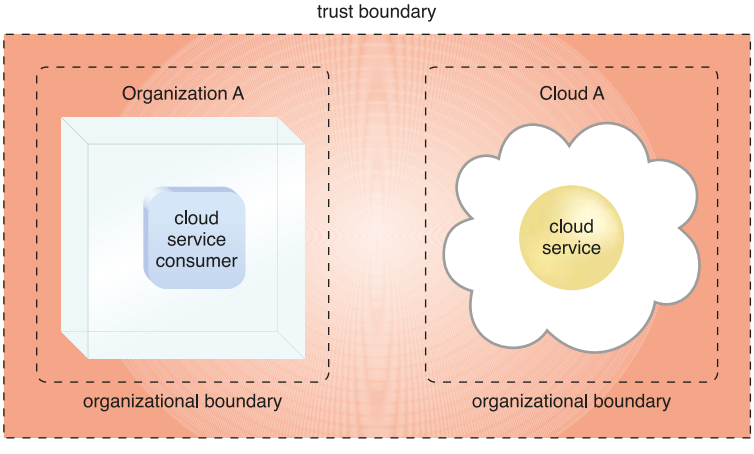


Figure 4.8
An extended trust boundary encompasses the organizational boundaries of the cloud provider and the cloud consumer.

NOTE

Another type of boundary relevant to cloud environments is the logical network perimeter. This type of boundary is classified as a cloud computing mechanism and is covered in Chapter 8.

4.2 Cloud Characteristics

An IT environment requires a specific set of characteristics to enable the remote provisioning of scalable and measured IT resources in an effective manner. These characteristics need to exist to a meaningful extent for the IT environment to be considered an effective cloud.

The following characteristics are common to the majority of cloud environments:

- on-demand usage
- ubiquitous access
- multitenancy (and resource pooling)
- elasticity
- measured usage
- resiliency

Cloud providers and cloud consumers can assess these characteristics both individually and collectively to measure the value offering of a given cloud platform. Although cloud-based services and IT resources will inherit and exhibit individual characteristics to varying extents, usually the greater the degree to which they are supported and utilized, the greater the resulting value proposition.

On-Demand Usage

A cloud consumer can unilaterally access cloud-based IT resources, giving the cloud consumer the freedom to self-provision these IT resources. Once configured, usage of the self-provisioned IT resources can be automated, requiring no further human involvement by the cloud consumer or cloud provider. This results in an *on-demand usage* environment. Also known as "on-demand self-service usage," this characteristic enables the service-based and usage-driven features found in mainstream clouds.

Ubiquitous Access

Ubiquitous access represents the ability for a cloud service to be widely accessible. Establishing ubiquitous access for a cloud service can require support for a range of devices, transport protocols, interfaces, and security technologies. To enable this level of access generally requires that the cloud service architecture be tailored to the particular needs of different cloud service consumers.

Multitenancy (and Resource Pooling)

The characteristic of a software program that enables an instance of the program to serve different consumers (tenants) whereby each is isolated from the other is referred to as *multitenancy*. A cloud provider pools its IT resources to serve multiple cloud service consumers by using multitenancy models that frequently rely on the use of virtualization technologies. Through the use of multitenancy technology, IT resources can be dynamically assigned and reassigned according to cloud service consumer demands.

Resource pooling allows cloud providers to pool large-scale IT resources to serve multiple cloud consumers. Different physical and virtual IT resources are dynamically assigned and reassigned according to cloud consumer demand, typically followed by execution through statistical multiplexing. Resource pooling is commonly achieved through multitenancy technology, and therefore encompassed by the multitenancy characteristic. See the *Resource Pooling Architecture* section in Chapter 13 for a more detailed explanation.

Figures 4.9 and 4.10 illustrate the difference between single-tenant and multitenant environments.

As illustrated in Figure 4.10, multitenancy allows several cloud consumers to use the same IT resource or its instance while each remains unaware that it may be used by others.

Elasticity

Elasticity is the automated ability of a cloud to transparently scale IT resources, as required in response to runtime conditions or as predetermined by the cloud consumer or cloud provider. Elasticity is often considered a core justification for the adoption of cloud computing, primarily due to the fact that it is closely associated with the Reduced Investment and Proportional Costs benefit. Cloud providers with vast IT resources can offer the greatest range of elasticity.

Figure 4.9
In a single-tenant environment, each cloud consumer has a separate IT resource instance.

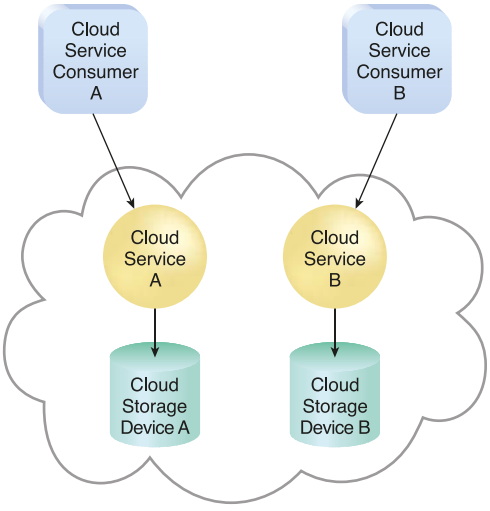
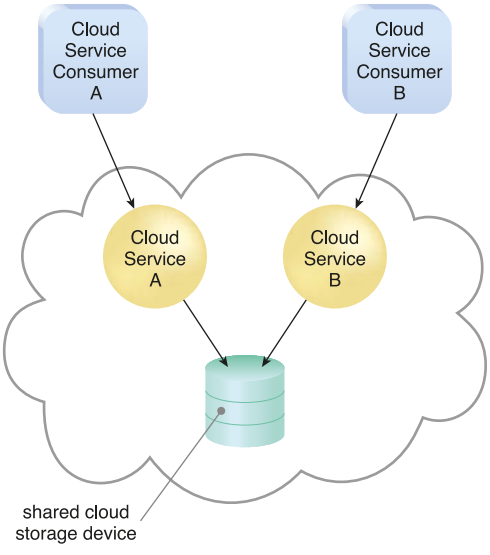


Figure 4.10
In a multitenant environment, a single instance of an IT resource, such as a cloud storage device, serves multiple consumers.



Measured Usage

The *measured usage* characteristic represents the ability of a cloud platform to keep track of the usage of its IT resources, primarily by cloud consumers. Based on what is measured, the cloud provider can charge a cloud consumer only for the IT resources actually used and/or for the timeframe during which access to the IT resources was granted. In this context, measured usage is closely related to the on-demand characteristic.

Measured usage is not limited to tracking statistics for billing purposes. It also encompasses the general monitoring of IT resources and related usage reporting (for both cloud provider and cloud consumers). Therefore, measured usage is also relevant to clouds that do not charge for usage (which may be applicable to the private cloud deployment model described in the upcoming *Cloud Deployment Models* section).

Resiliency

Resilient computing is a form of failover that distributes redundant implementations of IT resources across physical locations. IT resources can be preconfigured so that if one becomes deficient, processing is automatically handed over to another redundant implementation. Within cloud computing, the characteristic of *resiliency* can refer to redundant IT resources within the same cloud (but in different physical locations) or across multiple clouds. Cloud consumers can increase both the reliability and availability of their applications by leveraging the resiliency of cloud-based IT resources (Figure 4.11).

4.3 Cloud Delivery Models

A *cloud delivery model* represents a specific, prepackaged combination of IT resources offered by a cloud provider. Three common cloud delivery models have become widely established and formalized:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

These three models are interrelated in how the scope of one can encompass that of another, as explored in the *Combining Cloud Delivery Models* section later in this chapter.

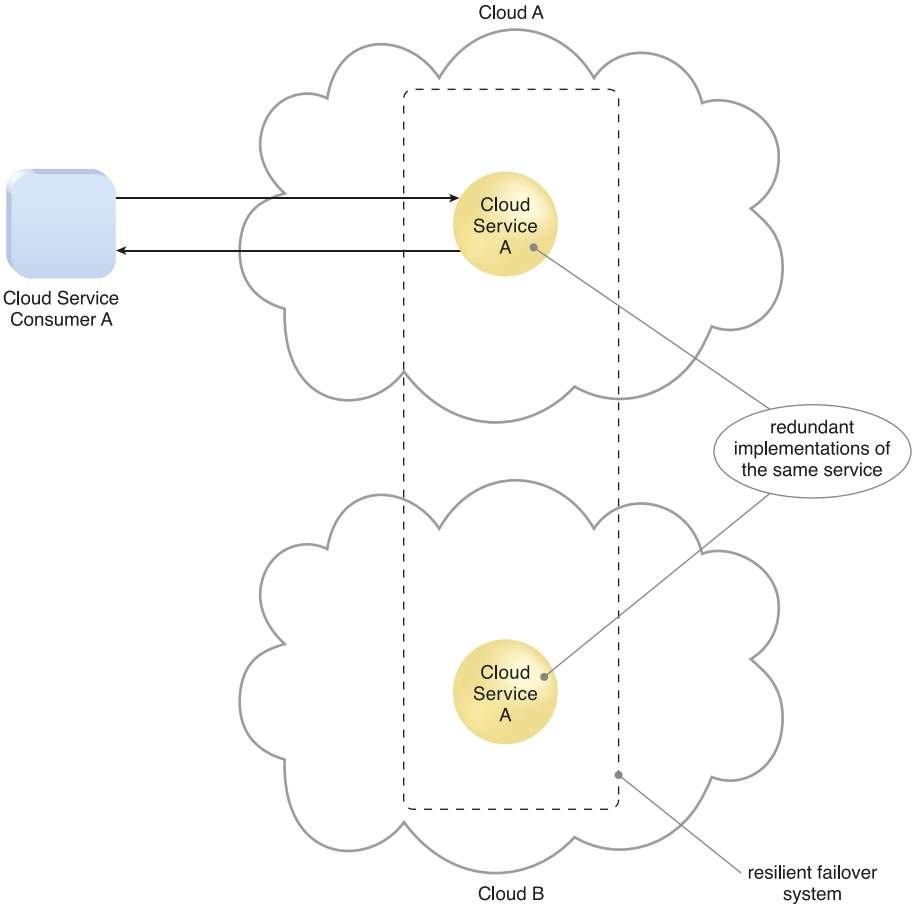


Figure 4.11

A resilient system in which Cloud B hosts a redundant implementation of Cloud Service A to provide failover in case Cloud Service A on Cloud A becomes unavailable.

NOTE

A cloud delivery model can be referred to as a cloud service delivery model because each model is classified as a different type of cloud service offering.

Infrastructure as a Service (IaaS)

The IaaS delivery model represents a self-contained IT environment comprised of infrastructure-centric IT resources that can be accessed and managed via cloud service-based interfaces and tools. This environment can include hardware, network, connectivity, operating systems, and other "raw" IT resources. In contrast to traditional hosting or outsourcing environments, with IaaS, IT resources are typically virtualized and packaged into bundles that simplify up-front runtime scaling and customization of the infrastructure.

The general purpose of an IaaS environment is to provide cloud consumers with a high level of control and responsibility over its configuration and utilization. The IT resources provided by IaaS are generally not preconfigured, placing the administrative responsibility directly upon the cloud consumer. This model is therefore used by cloud consumers that require a high level of control over the cloud-based environment they intend to create.

Sometimes cloud providers will contract IaaS offerings from other cloud providers to scale their own cloud environments. The types and brands of the IT resources provided by IaaS products offered by different cloud providers can vary. IT resources available through IaaS environments are generally offered as freshly initialized virtual instances. A central and primary IT resource within a typical IaaS environment is the virtual server. Virtual servers are leased by specifying server hardware requirements, such as processor capacity, memory, and local storage space, as shown in Figure 4.12.

Platform as a Service (PaaS)

The PaaS delivery model represents a predefined "ready-to-use" environment typically comprised of already deployed and configured IT resources. Specifically, PaaS relies on (and is primarily defined by) the usage of a ready-made environment that establishes a set of prepackaged products and tools used to support the entire delivery lifecycle of custom applications.

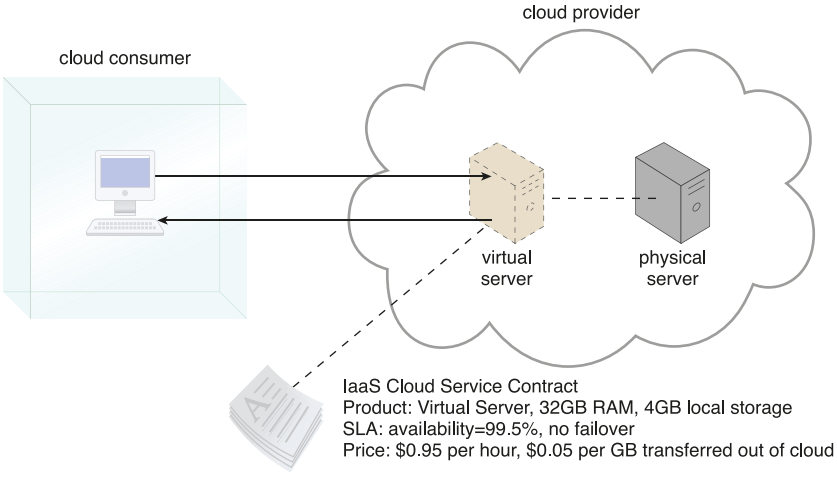


Figure 4.12
A cloud consumer is using a virtual server within an IaaS environment. Cloud consumers are provided with a range of contractual guarantees by the cloud provider, pertaining to characteristics such as capacity, performance, and availability.

Common reasons a cloud consumer would use and invest in a PaaS environment include:

- The cloud consumer wants to extend on-premises environments into the cloud for scalability and economic purposes.
- The cloud consumer uses the ready-made environment to entirely substitute an on-premises environment.
- The cloud consumer wants to become a cloud provider and deploys its own cloud services to be made available to other external cloud consumers.

By working within a ready-made platform, the cloud consumer is spared the administrative burden of setting up and maintaining the bare infrastructure IT resources provided via the IaaS model. Conversely, the cloud consumer is granted a lower level of control over the underlying IT resources that host and provision the platform (Figure 4.13).

PaaS products are available with different development stacks. For example, Google App Engine offers Java- and Python-based environments.

The ready-made environment is further described as a cloud computing mechanism in Chapter 8.

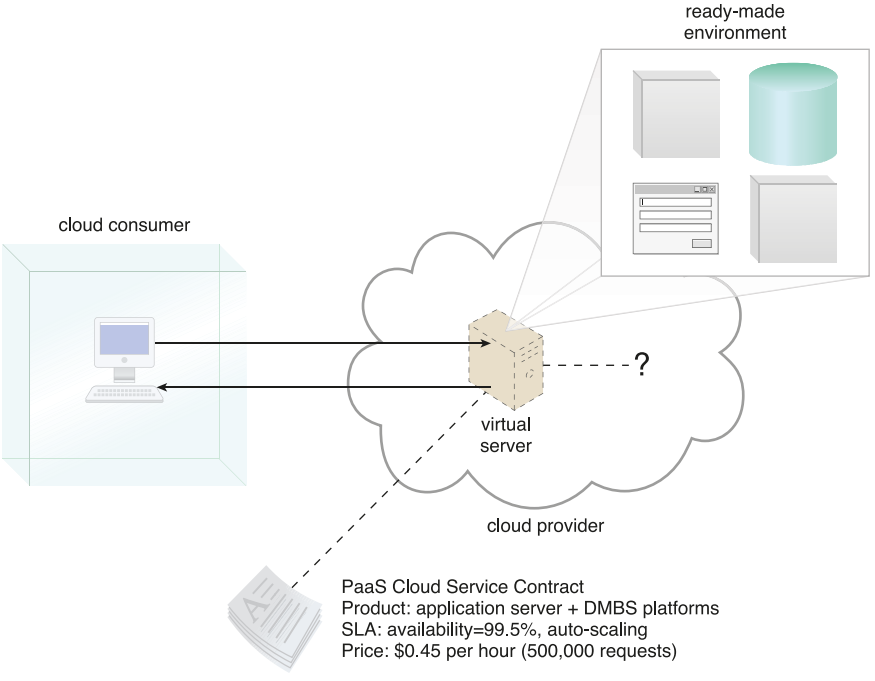


Figure 4.13

A cloud consumer is accessing a ready-made PaaS environment. The question mark indicates that the cloud consumer is intentionally shielded from the implementation details of the platform.

Software as a Service (SaaS)

A software program positioned as a shared cloud service and made available as a "product" or generic utility represents the typical profile of a SaaS offering. The SaaS delivery model is typically used to make a reusable cloud service widely available (often commercially) to a range of cloud consumers. An entire marketplace exists around SaaS products that can be leased and used for different purposes and via different terms (Figure 4.14).

A cloud consumer is generally granted very limited administrative control over a SaaS implementation. It is most often provisioned by the cloud provider, but it can be legally owned by whichever entity assumes the cloud service owner role. For example, an organization acting as a cloud consumer while using and working with a PaaS environment can build a cloud service that it decides to deploy in that same environment as a SaaS offering. The same organization then effectively assumes the cloud provider role as the SaaS-based cloud service is made available to other organizations that act as cloud consumers when using that cloud service.

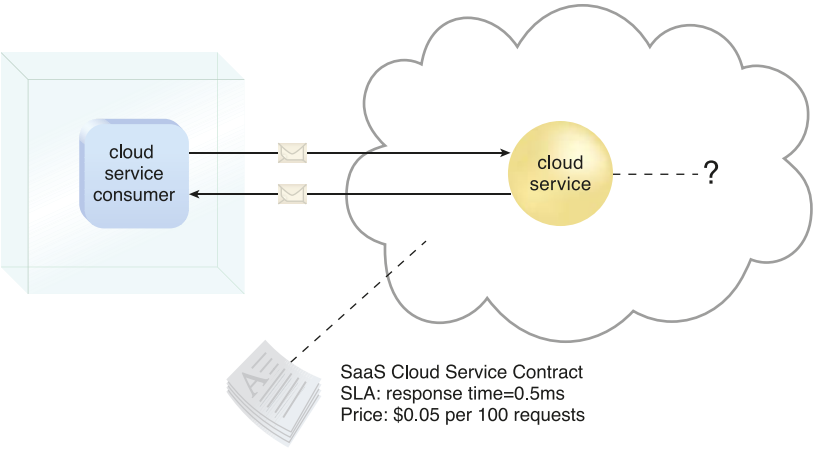


Figure 4.14
The cloud service consumer is given access to the cloud service contract, but not to any underlying IT resources or implementation details.

Comparing Cloud Delivery Models

Provided in this section are two tables that compare different aspects of cloud delivery model usage and implementation. Table 4.1 contrasts control levels and Table 4.2 compares typical responsibilities and usage.

Cloud Delivery Model	Typical Level of Control Granted to Cloud Consumer	Typical Functionality Made Available to Cloud Consumer
SaaS	usage and usage-related configuration	access to front-end user interface
PaaS	limited administrative	moderate level of administrative control over IT resources relevant to cloud consumer's usage of platform
IaaS	full administrative	full access to virtualized infrastructure-related IT resources and, possibly, to underlying physical IT resources

Table 4.1
A comparison of typical cloud delivery model control levels.

Cloud Delivery Model	Common Cloud Consumer Activities	Common Cloud Provider Activities
SaaS	uses and configures cloud service	implements, manages, and maintains cloud service monitors usage by cloud consumers
PaaS	develops, tests, deploys, and manages cloud services and cloud-based solutions	preconfigures platform and provisions underlying infrastructure, middleware, and other needed IT resources, as necessary monitors usage by cloud consumers
IaaS	sets up and configures bare infrastructure, and installs, manages, and monitors any needed software	provisions and manages the physical processing, storage, networking, and hosting required monitors usage by cloud consumers

Table 4.2
Typical activities carried out by cloud consumers and cloud providers in relation to the cloud delivery models.

Combining Cloud Delivery Models

The three base cloud delivery models comprise a natural provisioning hierarchy, allowing for opportunities for the combined application of the models to be explored. The upcoming sections briefly highlight considerations pertaining to two common combinations.

IaaS + PaaS

A PaaS environment will be built upon an underlying infrastructure comparable to the physical and virtual servers and other IT resources provided in an IaaS environment. Figure 4.15 shows how these two models can conceptually be combined into a simple layered architecture.

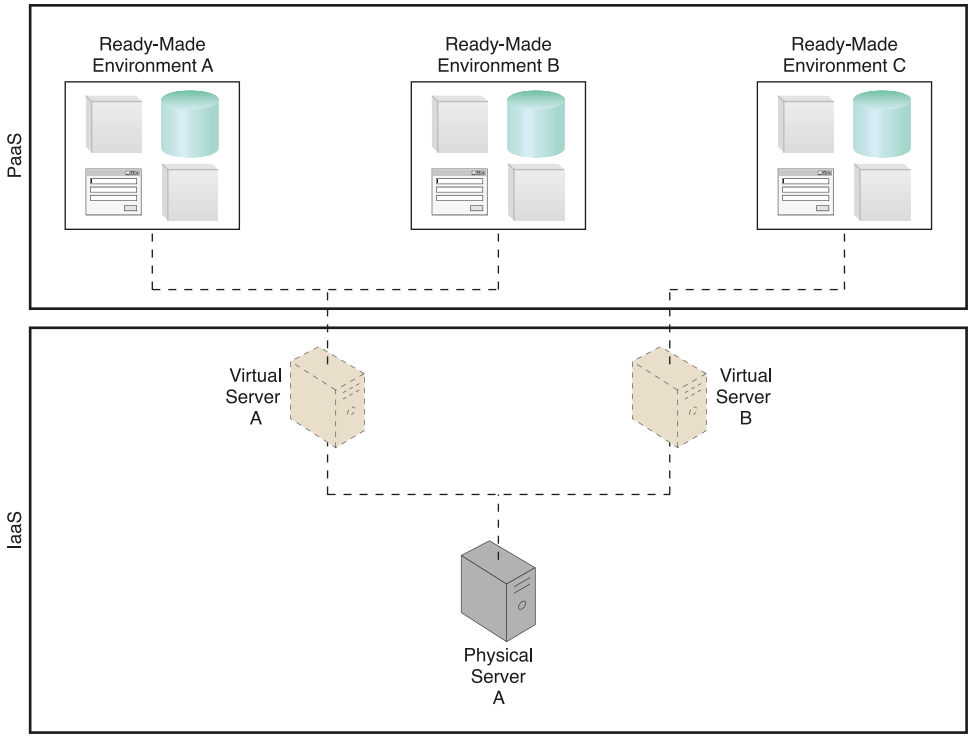


Figure 4.15
A PaaS environment based on the IT resources provided by an underlying IaaS environment.

A cloud provider would not normally need to provision an IaaS environment from its own cloud to make a PaaS environment available to cloud consumers. So how would the architectural view provided by Figure 4.16 be useful or applicable? Suppose that the cloud provider offering the PaaS environment chose to lease an IaaS environment from a *different* cloud provider.

The motivation for such an arrangement may be influenced by economics or maybe because the first cloud provider is close to exceeding its existing capacity by serving other cloud consumers. Or, perhaps a particular cloud consumer imposes a legal requirement for data to be physically stored in a specific region (different from where the first cloud provider's cloud resides), as illustrated in Figure 4.16.

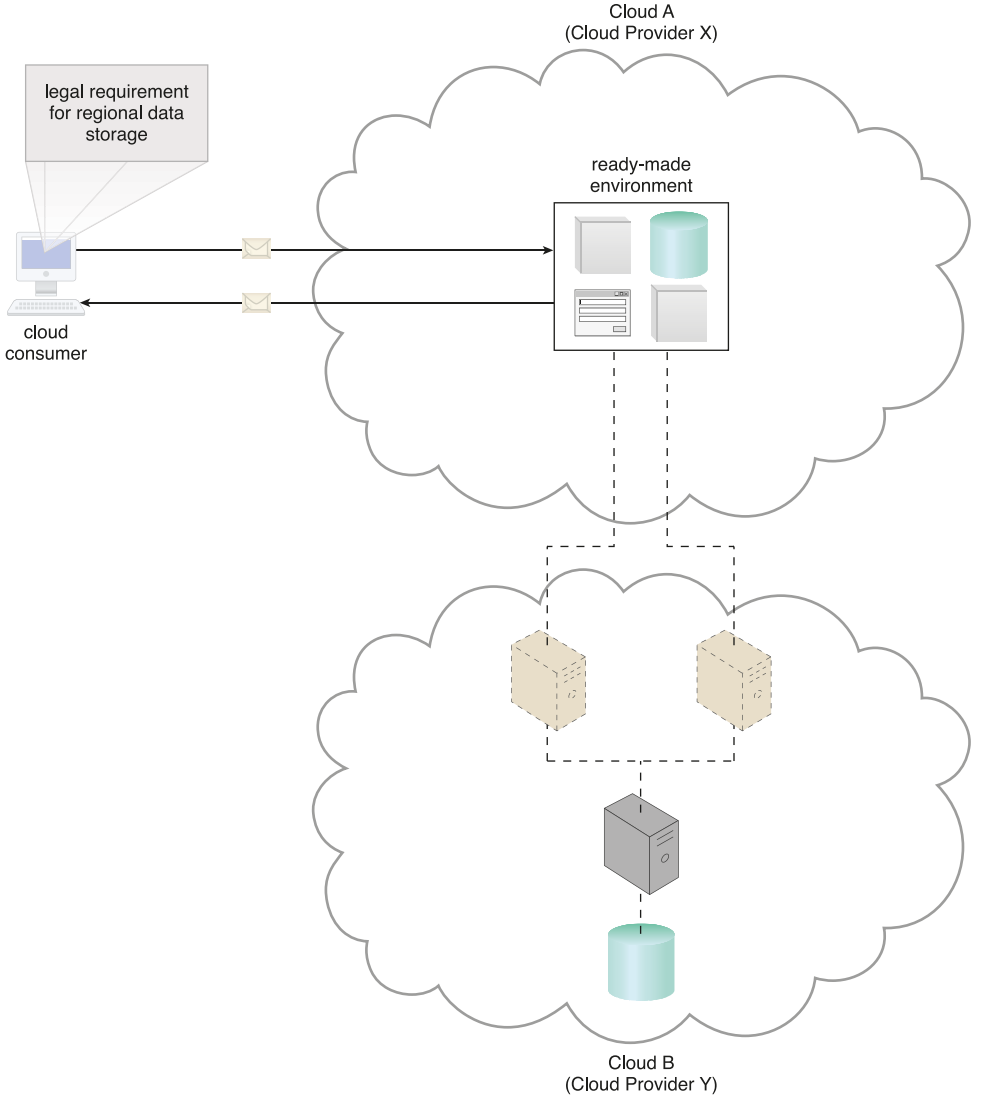


Figure 4.16

An example of a contract between Cloud Providers X and Y, in which services offered by Cloud Provider X are physically hosted on virtual servers belonging to Cloud Provider Y. Sensitive data that is legally required to stay in a specific region is physically kept in Cloud B, which is physically located in that region.

IaaS + PaaS + SaaS

All three cloud delivery models can be combined to establish layers of IT resources that build upon each other. For example, by adding on to the preceding layered architecture shown in Figure 4.16, the ready-made environment provided by the PaaS environment can be used by the cloud consumer organization to develop and deploy its own SaaS cloud services that it can then make available as commercial products (Figure 4.17).

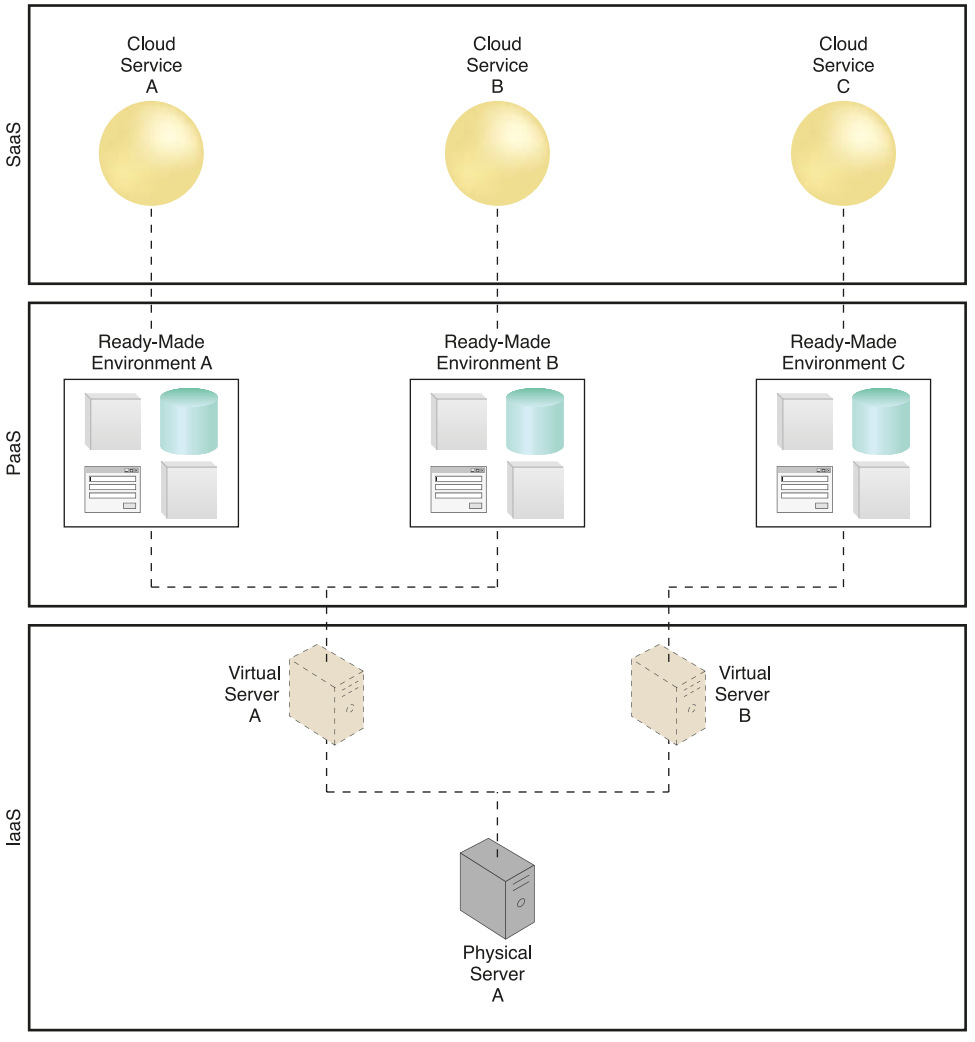


Figure 4.17

A simple layered view of an architecture comprised of IaaS and PaaS environments hosting three SaaS cloud service implementations.

Cloud Delivery Submodels

Many specialized variations of the cloud delivery models exist, each comprised of a distinct combination of IT resources. These *cloud delivery submodels* are also typically named using the "as a service" convention, and each can be mapped to one of the three basic cloud delivery models.

For example, the Database as a Service submodel (Figure 4.18) belongs to the PaaS model, since a database system is commonly a component of the ready-made environment that is part of a PaaS platform.

Similarly, Security as a Service is a submodel of SaaS that is used to provide access to features that can be used to secure cloud consumer IT assets.

Another example is the Storage as a Service submodel (Figure 4.19) of IaaS, which a cloud provider can use to deliver cloud storage–related services to cloud consumers.

Figure 4.18
The Database as a Service cloud delivery submodel is represented by a cloud provider to provide access to databases.

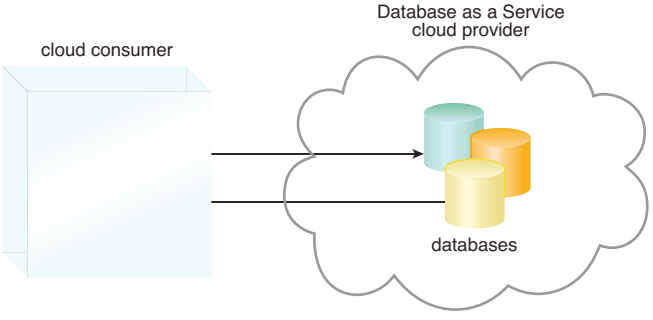
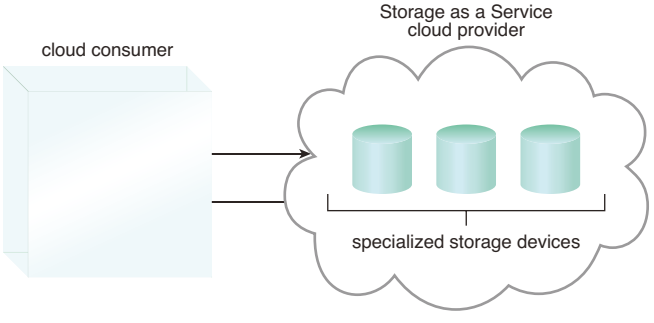


Figure 4.19
A Storage as a Service offering can provide different storage-related services, such as structured and unstructured data storage, file storage, object storage, and long-term archive storage.



Also considered a submodel of SaaS is the *cloud-native delivery submodel*, which allows for cloud-native applications to be built and deployed as collections of self-contained services packaged in lightweight containers.

Cloud-native applications (Figure 4.20) have no preference for any particular operating system or computer and work at a higher degree of abstraction. These types of applications run on infrastructure that is virtualized, shared, and elastic. They may align with the underlying infrastructure to dynamically grow and shrink in response to load fluctuations.

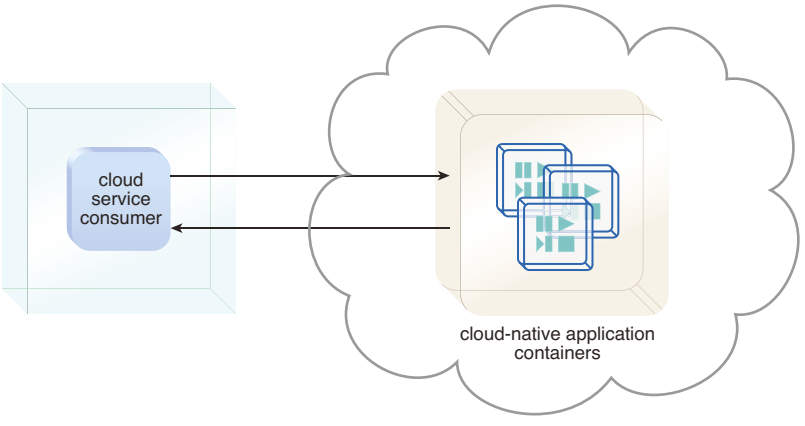


Figure 4.20
A cloud-native application deployed using multiple containers.

Other examples of common cloud delivery submodels include (but are not limited to) the following:

- Communication as a Service (a submodel of SaaS)
- Integration as a Service (a submodel of PaaS)
- Testing as a Service (a submodel of SaaS)
- Process as a Service (a submodel of SaaS)
- Desktop as a Service (a submodel of IaaS)

4.4 Cloud Deployment Models

A cloud deployment model represents a specific type of cloud environment, primarily distinguished by ownership, size, and access.

There are four common cloud deployment models:

- Public Cloud
- Private Cloud
- Multicloud
- Hybrid Cloud

The following sections describe each model.

Public Clouds

A *public cloud* is a publicly accessible cloud environment owned by a third-party cloud provider. The IT resources on public clouds are usually provisioned via the previously described cloud delivery models and are generally offered to cloud consumers at a cost or are commercialized via other avenues (such as advertisement).

The cloud provider is responsible for the creation and ongoing maintenance of the public cloud and its IT resources. Many of the scenarios and architectures explored in upcoming chapters involve public clouds and the relationship between the providers and consumers of IT resources via public clouds.

Figure 4.21 shows a partial view of the public cloud landscape, highlighting some of the primary vendors in the marketplace.

Private Clouds

A private cloud is owned by a single organization. Private clouds enable an organization to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations, or departments of the organization. When a private cloud exists as a controlled environment, the problems described in the *Risks and Challenges* section from Chapter 3 do not tend to apply.

The use of a private cloud can change how organizational and trust boundaries are defined and applied. The actual administration of a private cloud environment may be carried out by internal or outsourced staff.

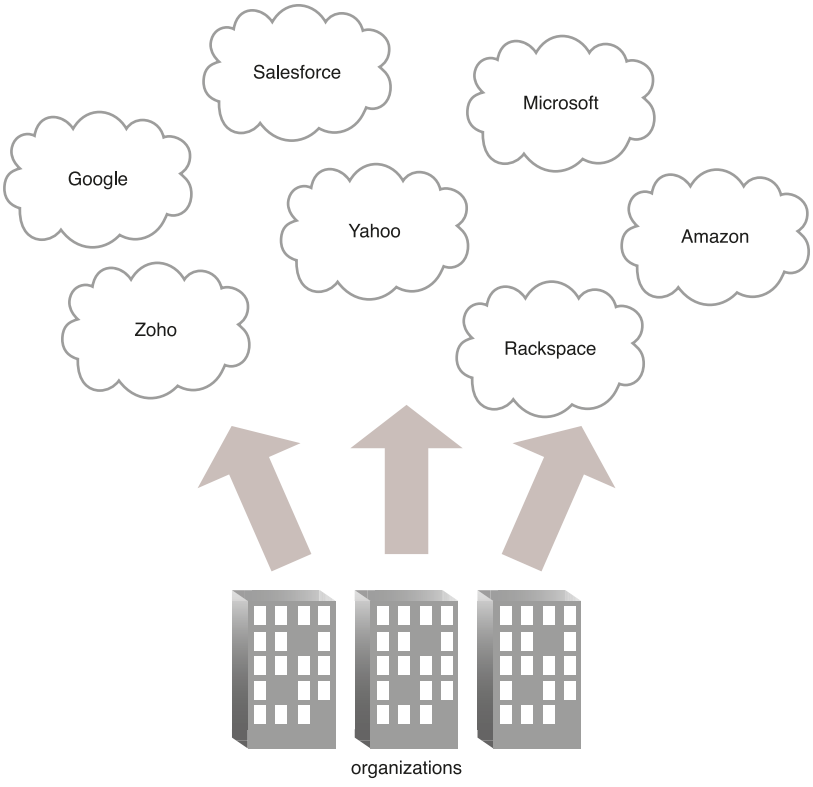


Figure 4.21
Organizations act as cloud consumers when accessing cloud services and IT resources made available by different cloud providers.

With a private cloud, the same organization is technically both the cloud consumer and cloud provider (Figure 4.22). To differentiate these roles:

- a separate organizational department typically assumes the responsibility for provisioning the cloud (and therefore assumes the cloud provider role)
- departments requiring access to the private cloud assume the cloud consumer role

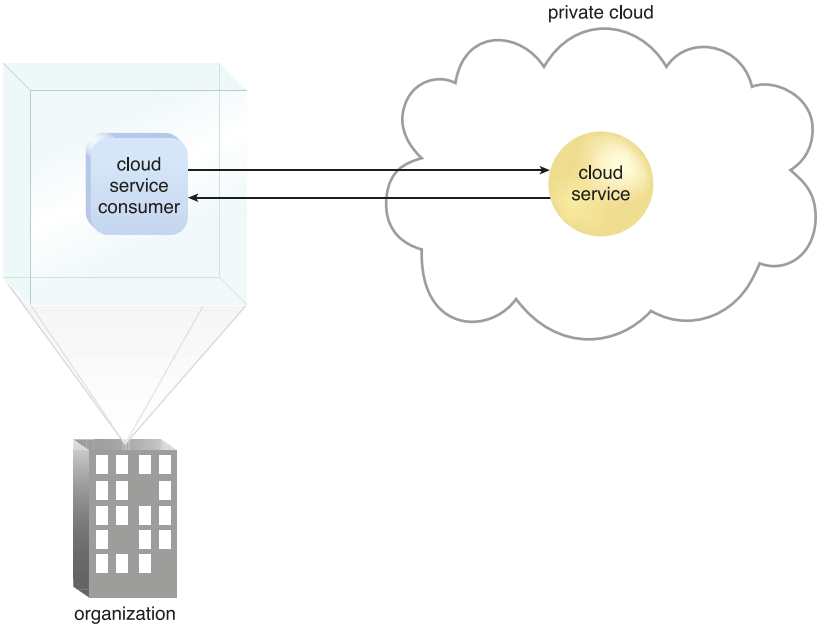


Figure 4.22

A cloud service consumer in the organization's on-premises environment accesses a cloud service hosted on the same organization's private cloud via a virtual private network.

It is important to use the terms "on premises" and "cloud-based" correctly within the context of a private cloud. Even though the private cloud may physically reside on the organization's premises, IT resources it hosts are still considered "cloud-based" as long as they are made remotely accessible to cloud consumers. IT resources hosted outside of the private cloud by the departments acting as cloud consumers are therefore considered "on premises" in relation to the private cloud-based IT resources.

Multiclouds

With a multicloud deployment model, a cloud consumer organization can use cloud services and IT resources from different public clouds provided by multiple cloud providers, as shown in Figure 4.23.

For example, this deployment model can be used to improve redundancy and system backups, to improve mobility by reducing vendor lock-in, or to leverage best-of-breed cloud services from different cloud vendors.

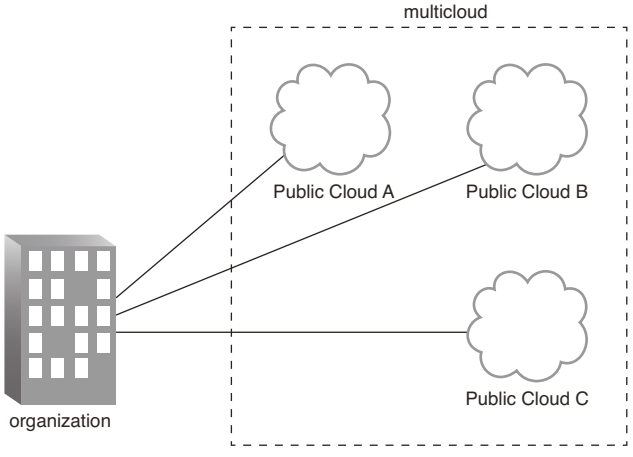


Figure 4.23
An organization uses the multicloud model to utilize cloud-based IT resources from different cloud providers.

Hybrid Clouds

A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models. For example, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services to a public cloud. The result of this combination is a hybrid deployment model (Figure 4.24).

Hybrid deployment architectures can be complex and challenging to create and maintain due to the potential disparity in cloud environments and the fact that management responsibilities are typically split between the private cloud provider organization and the public cloud provider.

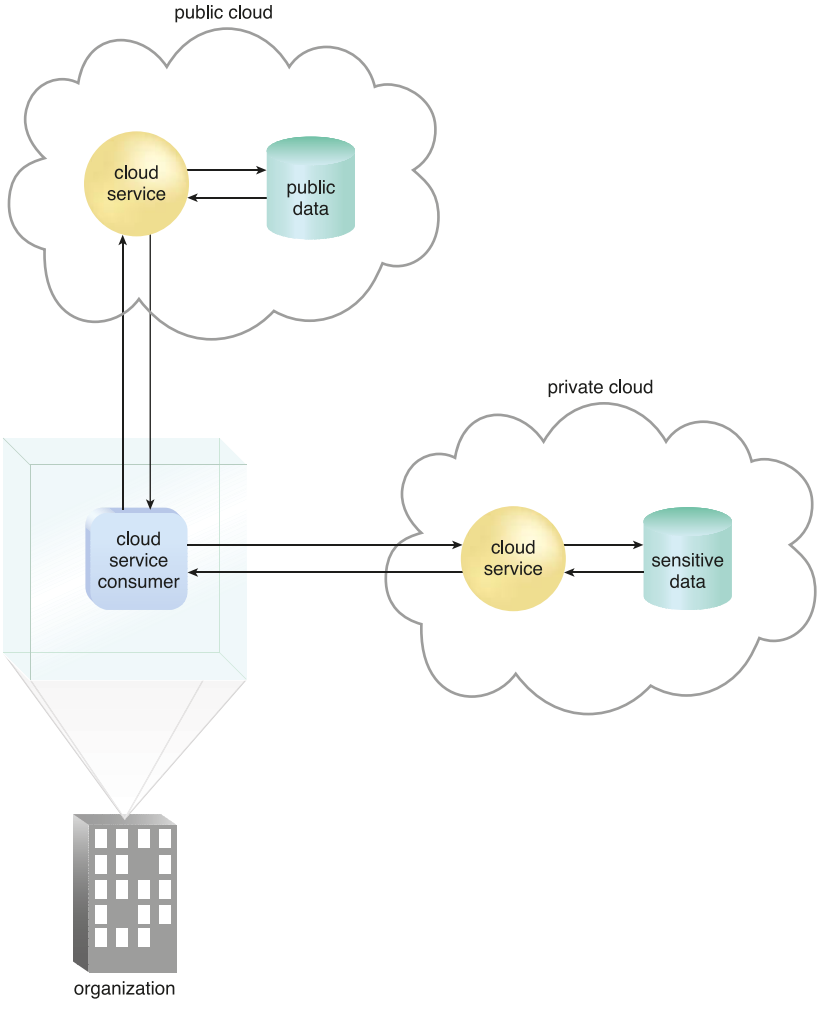


Figure 4.24
An organization using a hybrid cloud architecture that utilizes both a private cloud and a public cloud.

Chapter 5

Cloud-Enabling Technology

Modern-day clouds are underpinned by a set of primary technology components that collectively enable key features and characteristics associated with contemporary cloud computing.

Most existed and matured prior to the advent of cloud computing, although cloud computing advancements helped further evolve select areas of these cloud-enabling technologies.

5.1 Networks and Internet Architecture

All clouds must be connected to a network. This inevitable requirement forms an inherent dependency on internetworking.

Internetworks, or the internet, allow for the remote provisioning of IT resources and are directly supportive of ubiquitous network access. Cloud consumers have the option of accessing the cloud using only private and dedicated network links in LANs, although most clouds are internet-enabled. The potential of cloud platforms therefore generally grows in parallel with advancements in internet connectivity and service quality.

Internet Service Providers (ISPs)

Established and deployed by ISPs, the internet's largest backbone networks are strategically interconnected by core routers that connect the world's multinational networks. As shown in Figure 5.1, an ISP network interconnects to other ISP networks and various organizations.

The concept of the internet was based on a decentralized provisioning and management model. ISPs can freely deploy, operate, and manage their networks in addition to selecting partner ISPs for interconnection. No centralized entity comprehensively governs the internet, although bodies like the Internet Corporation for Assigned Names and Numbers (ICANN) supervise and coordinate internet communications.

Governmental and regulatory laws dictate the service provisioning conditions for organizations and ISPs both within and outside of national borders. Certain realms of the internet still require the demarcation of national jurisdiction and legal boundaries.

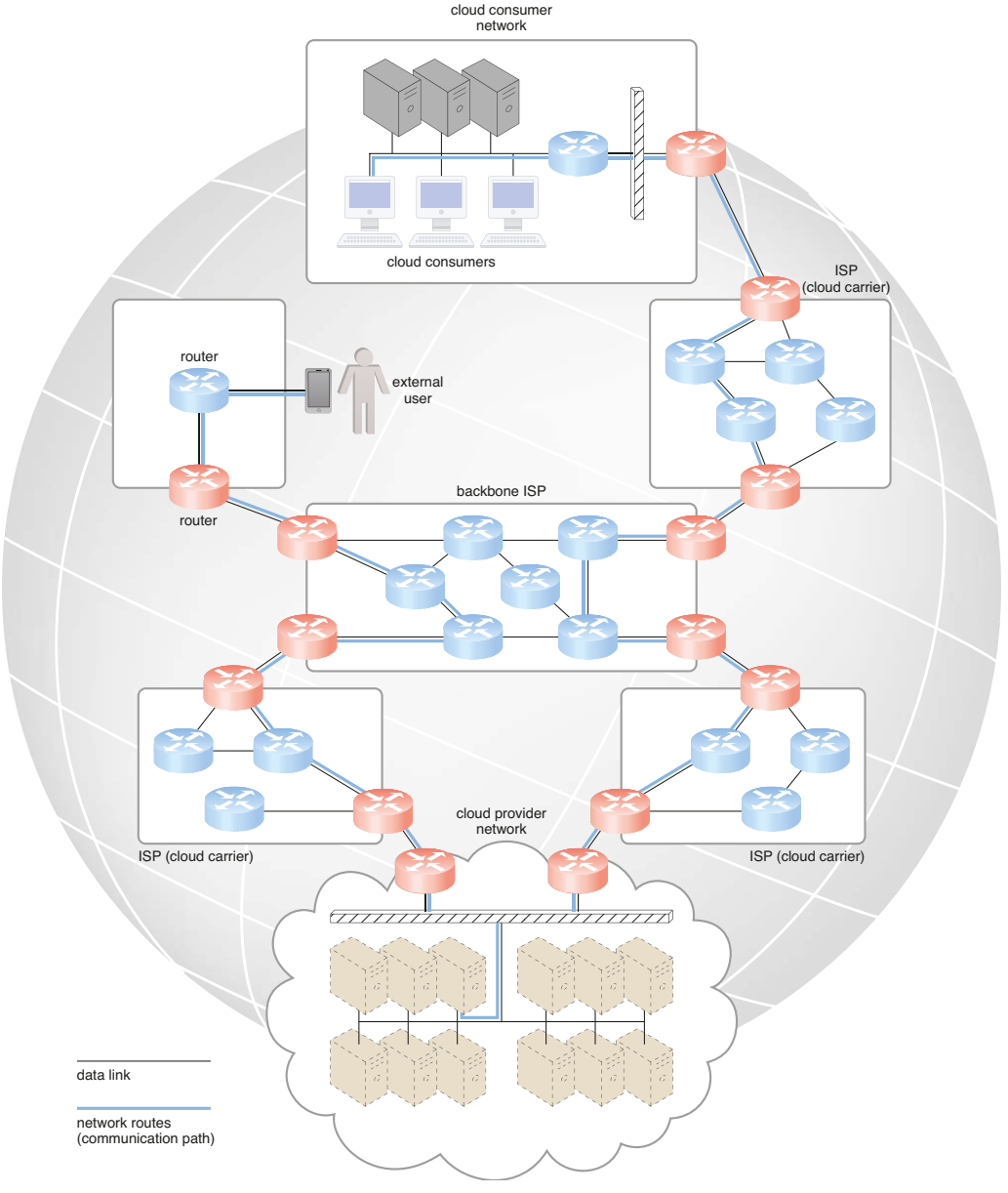


Figure 5.1

Messages travel over dynamic network routes in this ISP internetworking configuration.

The internet's topology has become a dynamic and complex aggregate of ISPs that are highly interconnected via its core protocols. Smaller branches extend from these major nodes of interconnection, branching outwards through smaller networks until they eventually reach every internet-enabled electronic device.

Worldwide connectivity is enabled through a hierarchical topology composed of Tiers 1, 2, and 3 (Figure 5.2). The core Tier 1 is made up of large-scale, international cloud providers that oversee massive interconnected global networks, which are connected to Tier 2's large regional providers. The interconnected ISPs of Tier 2 connect with Tier 1 providers, as well as the local ISPs of Tier 3. Cloud consumers and cloud providers can connect directly using a Tier 1 provider, since any operational ISP can enable internet connection.

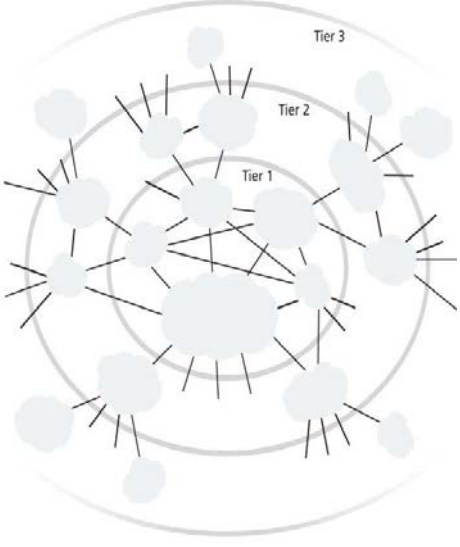


Figure 5.2
An abstraction of the internetworking structure of the internet.

The communication links and routers of the internet and ISP networks are IT resources that are distributed among countless traffic generation paths. Two fundamental components used to construct the internetworking architecture are *connectionless packet switching* (datagram networks) and *router-based interconnectivity*.

Connectionless Packet Switching (Datagram Networks)

End-to-end (sender–receiver pair) data flows are divided into packets of a limited size that are received and processed through network switches and routers, then queued

and forwarded from one intermediary node to the next. Each packet carries the necessary location information, such as the Internet Protocol (IP) or Media Access Control (MAC) address, to be processed and routed at every source, intermediary, and destination node.

Router-Based Interconnectivity

A router is a device that is connected to multiple networks through which it forwards packets. Even when successive packets are part of the same data flow, routers process and forward each packet individually while maintaining the network topology information that locates the next node on the communication path between the source and destination nodes. Routers manage network traffic and gauge the most efficient hop for packet delivery, since they are privy to both the packet source and the packet destination.

The basic mechanics of internetworking are illustrated in Figure 5.3, in which a message is coalesced from an incoming group of disordered packets. The depicted router receives and forwards packets from multiple data flows.

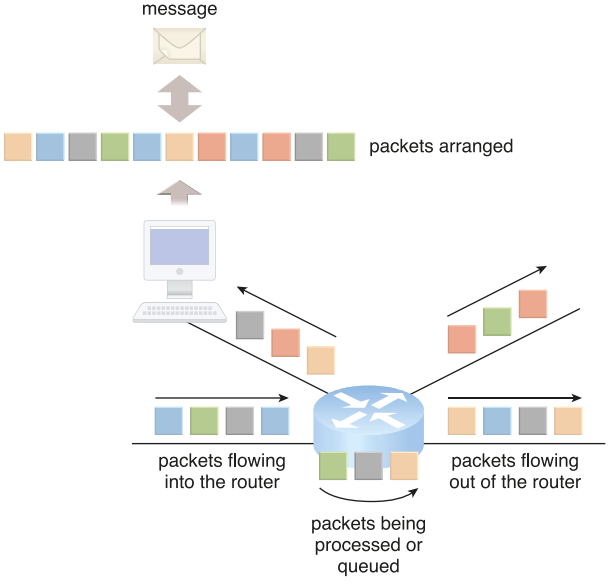


Figure 5.3

Packets traveling through the internet are directed by a router that arranges them into a message.

The communication path that connects a cloud consumer with its cloud provider may involve multiple ISP networks. The internet's mesh structure connects internet hosts (endpoint systems) using multiple alternative network routes that are determined at runtime. Communication can therefore be sustained even during simultaneous network failures, although using multiple network paths can cause routing fluctuations and latency.

This applies to ISPs that implement the internet's internetworking layer and interact with other network technologies, as follows:

Physical Network

IP packets are transmitted through underlying physical networks that connect adjacent nodes, such as Ethernet, ATM network, and the 3G mobile HSDPA. Physical networks comprise a data link layer that controls data transfer between neighboring nodes, and a physical layer that transmits data bits through both wired and wireless media.

Transport Layer Protocol

Transport layer protocols, such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), use the IP to provide standardized, end-to-end communication support that facilitates the navigation of data packets across the internet.

Application Layer Protocol

Protocols such as HTTP, SMTP for email, BitTorrent for P2P, and SIP for IP telephony use transport layer protocols to standardize and enable specific data packet transferring methods over the internet. Many other protocols also fulfill application-centric requirements and use either TCP/IP or UDP as their primary method of data transferring across the internet and LANs.

Figure 5.4 presents the Internet Reference Model and the protocol stack.

Technical and Business Considerations

Connectivity Issues

In traditional, on-premises deployment models, enterprise applications and various IT solutions are commonly hosted on centralized servers and storage devices residing in the organization's own data center. End-user devices, such as smartphones and laptops, access the data center through the corporate network, which provides uninterrupted internet connectivity.

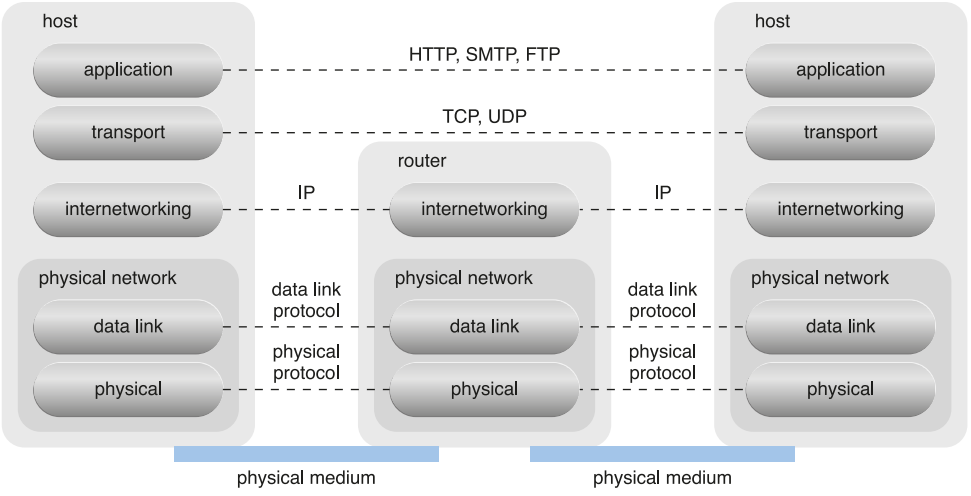


Figure 5.4

A generic view of the internet reference model and protocol stack.

TCP/IP facilitates both internet access and on-premises data exchange over LANs (Figure 5.5). Although not commonly referred to as a cloud model, this configuration has been implemented numerous times for medium and large on-premises networks.

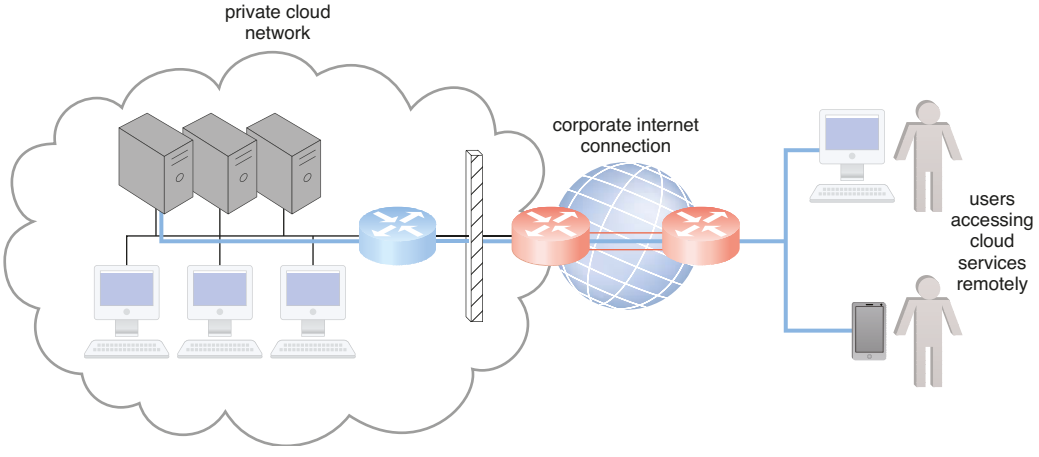


Figure 5.5

The internetworking architecture of a private cloud. The physical IT resources that constitute the cloud are located and managed within the organization.

Organizations using this deployment model can directly access the network traffic to and from the internet and usually have complete control over their corporate networks and can safeguard them using firewalls and monitoring software. These organizations also assume the responsibility of deploying, operating, and maintaining their IT resources and internet connectivity.

End-user devices that are connected to the network through the internet can be granted continuous access to centralized servers and applications in the cloud (Figure 5.6).

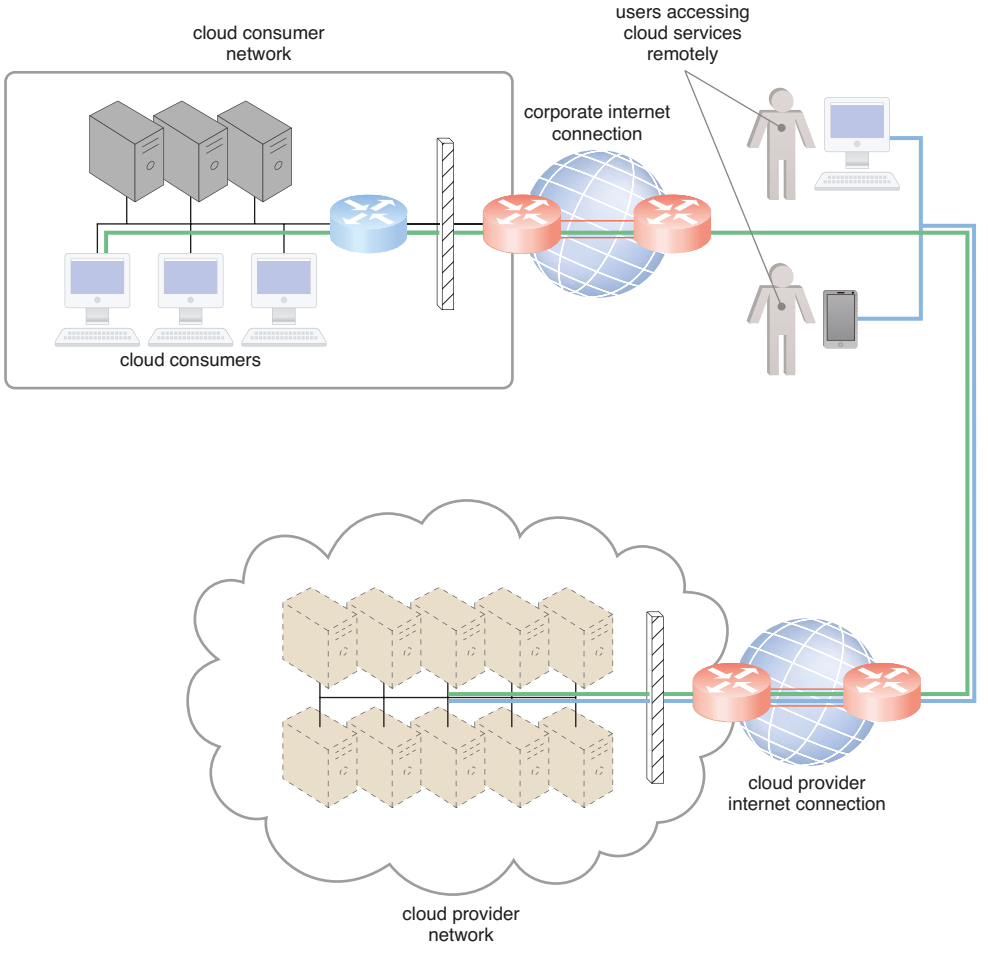


Figure 5.6

The internetworking architecture of an internet-based cloud deployment model. The internet is the connecting agent between non-proximate cloud consumers, roaming end users, and the cloud provider's own network.

A salient cloud feature that applies to end-user functionality is how centralized IT resources can be accessed using the same network protocols regardless of whether they reside inside or outside of a corporate network. Whether IT resources are on premises or internet-based dictates how internal versus external end users access services, even if the end users themselves are not concerned with the physical location of cloud-based IT resources (Table 5.1).

On-Premises IT Resources	Cloud-Based IT Resources
internal end-user devices access corporate IT services through the corporate network	internal end-user devices access corporate IT services through an internet connection
internal users access corporate IT services through the corporate internet connection while roaming in external networks	internal users access corporate IT services while roaming in external networks through the cloud provider's internet connection
external users access corporate IT services through the corporate internet connection	external users access corporate IT services through the cloud provider's internet connection

Table 5.1
A comparison of on-premises and cloud-based internetworking.

Cloud providers can easily configure cloud-based IT resources to be accessible for both external and internal users through an internet connection (as previously shown in Figure 5.6). This internetworking architecture benefits internal users that require ubiquitous access to corporate IT solutions, as well as cloud consumers that need to provide internet-based services to external users. Major cloud providers offer internet connectivity that is superior to the connectivity of individual organizations, resulting in additional network usage charges as part of their pricing model.

Network Bandwidth and Latency Issues

In addition to being affected by the bandwidth of the data link that connects networks to ISPs, end-to-end bandwidth is determined by the transmission capacity of the shared data links that connect intermediary nodes. ISPs need to use broadband network technology to implement the core network required to guarantee end-to-end connectivity. This type of bandwidth is constantly increasing, as web acceleration technologies, such as dynamic caching, compression, and pre-fetching, continue to improve end-user connectivity.

Also referred to as time delay, *latency* is the amount of time it takes a packet to travel from one data node to another. Latency increases with every intermediary node on the data packet's path. Transmission queues in the network infrastructure can result in heavy load conditions that also increase network latency. Networks are dependent on traffic conditions in shared nodes, making internet latency highly variable and often unpredictable.

Packet networks with "best effort" quality of service (QoS) typically transmit packets on a first-come/first-serve basis. Data flows that use congested network paths suffer service-level degradation in the form of bandwidth reduction, latency increase, or packet loss when traffic is not prioritized.

The nature of packet switching allows data packets to choose routes dynamically as they travel through the internet's network infrastructure. End-to-end QoS can be impacted as a result of this dynamic selecting, since the travel speed of data packets is susceptible to conditions like network congestion and is therefore nonuniform.

IT solutions need to be assessed against business requirements that are affected by network bandwidth and latency, which are inherent to cloud interconnection. Bandwidth is critical for applications that require substantial amounts of data to be transferred to and from the cloud, while latency is critical for applications with a business requirement of swift response times.

Wireless and Cellular

Cloud-based solutions that need to be accessible anywhere from any device, especially those that are targeted towards mobile clients and consumers, need to be accessible via wireless and cellular communication links. For example, mobile edge computing (MEC), an enabling technology for the Internet of Vehicles (IoV), offers prospective solutions for sharing processing capabilities across vehicles as well as other readily available resources.

The autonomous vehicular edge (AVE) is a distributed vehicular edge computing technology that enables sharing of nearby cars' available resources via vehicle-to-vehicle (V2V) communications. AVE is a principle that can be applied to a broader online solution known as the hybrid vehicular edge cloud (HVC), which enables effective sharing of all obtainable computing resources, including roadside units (RSUs) and the cloud, via multiaccess networks.

These are all examples of how wireless and cellular networks can be adapted or evolved to constitute valid internetworking components of cloud-based solutions by

overcoming many of the natural bandwidth and latency restrictions of wireless and cellular technologies.

Cloud Carrier and Cloud Provider Selection

The service levels of internet connections between cloud consumers and cloud providers are determined by their ISPs, which are usually different and therefore include multiple ISP networks in their paths. QoS management across multiple ISPs is difficult to achieve in practice, requiring collaboration of the cloud carriers on both sides to ensure that their end-to-end service levels are sufficient for business requirements.

Cloud consumers and cloud providers may need to use multiple cloud carriers to achieve the necessary level of connectivity and reliability for their cloud applications, resulting in additional costs. Cloud adoption can therefore be easier for applications with more relaxed latency and bandwidth requirements.

5.2 Cloud Data Center Technology

Grouping IT resources in close proximity with one another, rather than having them geographically dispersed, allows for power sharing, higher efficiency in shared IT resource usage, and improved accessibility for IT personnel. These are the advantages that naturally popularized the data center concept. Modern data centers exist as specialized IT infrastructure used to house centralized IT resources, such as servers, databases, networking and telecommunication devices, and software systems. Data centers for cloud providers often require additional technologies.

Data centers are typically comprised of the following technologies and components.

Virtualization

Data centers consist of both physical and virtualized IT resources. The physical IT resource layer refers to the facility infrastructure that houses computing/networking systems and equipment, together with hardware systems and their operating systems (Figure 5.7). The resource abstraction and control of the virtualization layer is comprised of operational and management tools that are often based on virtualization platforms that abstract the physical computing and networking IT resources as virtualized components that are easier to allocate, operate, release, monitor, and control.

Virtualization components are discussed separately in the upcoming *Modern Virtualization* section.

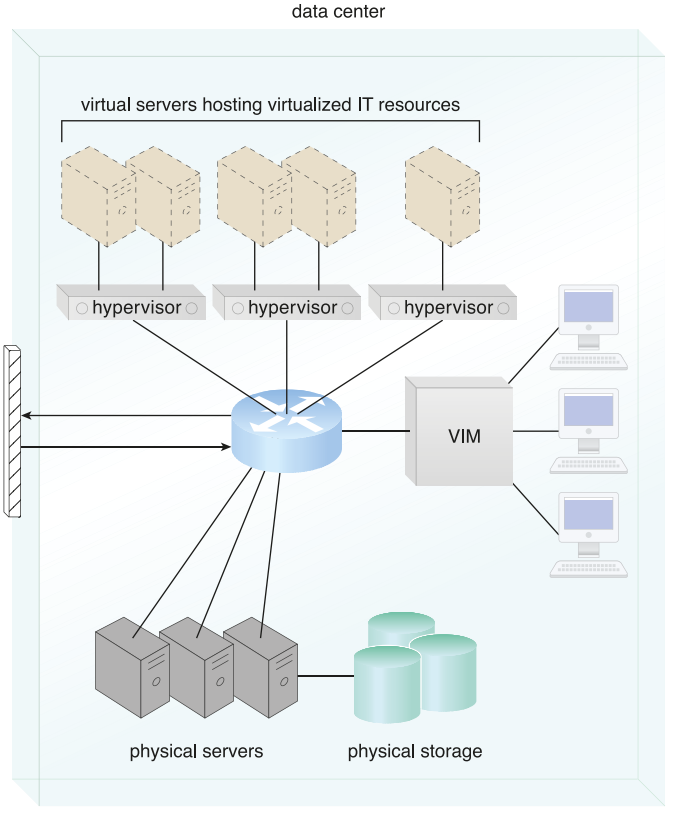


Figure 5.7

The common components of a data center working together to provide virtualized IT resources supported by physical IT resources.

Standardization and Modularity

Data centers are built upon standardized commodity hardware and designed with modular architectures, aggregating multiple identical building blocks of facility infrastructure and equipment to support scalability, growth, and speedy hardware replacements. Modularity and standardization are key requirements for reducing investment and operational costs, as they enable economies of scale for the procurement, acquisition, deployment, operation, and maintenance processes.

Common virtualization strategies and the constantly improving capacity and performance of physical devices both favor IT resource consolidation, since fewer physical components are needed to support complex configurations. Consolidated IT resources can serve different systems and be shared among different cloud consumers.

Autonomic Computing

Autonomic computing is the ability of a system to be self-managing, which means that it is expected to react to external input without the need for human intervention. Using autonomic computing, clouds can be equipped to manage certain tasks themselves, without human involvement.

The common features of self-management can include:

- *Self-configuration*, by which cloud services can configure themselves automatically in response to established policies, avoiding manual intervention from cloud resource administrators. This feature also involves the automatic configuration of new cloud resources when provisioned.
- *Self-optimization*, by which cloud resources continuously strive to improve their performance indicators by modifying their configuration parameters at runtime, such as scaling up or out dynamically.
- *Self-healing*, by which cloud services can recover from hardware or software failure, having detected and diagnosed issues automatically beforehand.
- *Self-protecting*, by which cloud computing platforms are able to defend themselves from malicious attacks or cascading failure conditions. This is possible due to their ability to predict potential problem situations based on the analysis of logs and diagnostics, in which data science technologies are typically involved.

Remote Operation and Management

Most of the operational and administrative tasks of IT resources in data centers are commanded through the network's remote consoles and management systems. Technical personnel are not required—and many times not allowed—to visit the dedicated rooms that house servers, except to perform highly specific tasks, such as equipment handling and cabling or hardware-level installation and maintenance.

High Availability

Since any form of data center outage significantly impacts business continuity for the organizations that use their services, data centers are designed to operate with increasingly higher levels of redundancy to sustain availability. Data centers usually have redundant, uninterruptible power supplies, cabling, and environmental control subsystems in anticipation of system failure, along with communication links and clustered hardware for load balancing.

Security-Aware Design, Operation, and Management

Requirements for security, such as physical and logical access controls and data recovery strategies, need to be thorough and comprehensive for data centers, since they are centralized structures that store and process business data.

Due to the sometimes-prohibitive nature of building and operating on-premises data centers, outsourcing data center-based IT resources has been a common industry practice for decades. However, the outsourcing models often required long-term consumer commitment and usually could not provide elasticity—issues that a typical cloud can address via inherent features, such as ubiquitous access, on-demand provisioning, rapid elasticity, and pay-per-use.

Facilities

Data center facilities are custom-designed locations that are outfitted with specialized computing, storage, and network equipment. These facilities have several functional layout areas, as well as various power supplies, cabling, and environmental control stations that regulate heating, ventilation, air conditioning, fire protection, and other related subsystems. The site and layout of a given data center facility are typically demarcated into segregated spaces.

Computing Hardware

Much of the heavy processing in data centers is often executed by standardized commodity servers that have substantial computing power and storage capacity. Several computing hardware technologies are integrated into these modular servers, such as:

- rackmount form factor server design composed of standardized racks with interconnects for power, network, and internal cooling
- support for different hardware processing architectures, such as x86-32bits, x86-64, and RISC
- a power-efficient multicore CPU architecture that houses hundreds of processing cores in a space as small as a single unit of standardized racks
- redundant and hot-swappable components, such as hard disks, power supplies, network interfaces, and storage controller cards

Computing architectures such as blade server technologies use rack-embedded physical interconnections (blade enclosures), fabrics (switches), and shared power supply units and cooling fans. The interconnections enhance intercomponent networking and management while optimizing physical space and power. These systems typically support individual server hot-swapping, scaling, replacement, and maintenance, which benefits the deployment of fault-tolerant systems that are based on computer clusters.

Contemporary computing hardware platforms generally support industry-standard and proprietary operational and management software systems that configure, monitor, and control hardware IT resources from remote management consoles. With a properly established management console, a single operator can oversee hundreds to thousands of physical servers, virtual servers, and other IT resources.

Storage Hardware

Data centers have specialized storage systems that maintain enormous amounts of digital information to fulfill considerable storage capacity needs. These storage systems are containers housing numerous hard disks that are organized into arrays.

Storage systems usually involve the following technologies:

- *Hard Disk Arrays* – These arrays inherently divide and replicate data among multiple physical drives, and increase performance and redundancy by including spare disks. This technology is often implemented using redundant arrays of independent disks (RAID) schemes, which are typically realized through hardware disk array controllers.
- *I/O Caching* – This is generally performed through hard disk array controllers, which enhance disk access times and performance by data caching.
- *Hot-Swappable Hard Disks* – These can be safely removed from arrays without requiring prior powering down.
- *Storage Virtualization* – This is realized through the use of virtualized hard disks and storage sharing.
- *Fast Data Replication Mechanisms* – These include *snapshotting*, which is saving a virtual machine's memory into a hypervisor-readable file for future reloading, and *volume cloning*, which is copying virtual or physical hard disk volumes and partitions.

Storage systems encompass tertiary redundancies, such as robotized tape libraries, which are used as backup and recovery systems that typically rely on removable media. This type of system can exist as a networked IT resource or direct-attached storage (DAS), in which a storage system is directly connected to the computing IT resource using a host bus adapter (HBA). In the former case, the storage system is connected to one or more IT resources through a network.

Networked storage devices usually fall into one of the following categories:

- *Storage Area Network (SAN)* – Physical data storage media are connected through a dedicated network and provide block-level data storage access using industry standard protocols, such as the Small Computer System Interface (SCSI).
- *Network-Attached Storage (NAS)* – Hard drive arrays are contained and managed by this dedicated device, which connects through a network and facilitates access to data using file-centric data access protocols like the Network File System (NFS) or Server Message Block (SMB).

NAS, SAN, and other more advanced storage system options provide fault tolerance in many components through controller redundancy, cooling redundancy, and hard disk arrays that use RAID storage technology.

Network Hardware

Data centers require extensive network hardware to enable multiple levels of connectivity. For a simplified version of networking infrastructure, the data center is broken down into five network subsystems, followed by a summary of the most common elements used for their implementation.

Carrier and External Networks Interconnection

A subsystem related to the internetworking infrastructure, this interconnection is usually comprised of backbone routers that provide routing between external WAN connections and the data center's LAN, as well as perimeter network security devices such as firewalls and VPN gateways.

Web-Tier Load Balancing and Acceleration

This subsystem comprises web acceleration devices, such as XML preprocessors, encryption/decryption appliances, and layer 7 switching devices that perform content-aware routing.

LAN Fabric

The LAN fabric constitutes the internal LAN and provides high-performance and redundant connectivity for all the data center's network-enabled IT resources. It is often implemented with multiple network switches that facilitate network communications and operate at speeds of up to 10 gigabits per second. These advanced network switches can also perform several virtualization functions, such as LAN segregation into VLANs, link aggregation, controlled routing between networks, load balancing, and failover.

SAN Fabric

Related to the implementation of storage area networks (SANs) that provide connectivity between servers and storage systems, the SAN fabric is usually implemented with Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), and InfiniBand network switches.

NAS Gateways

This subsystem supplies attachment points for NAS-based storage devices and implements protocol conversion hardware that facilitates data transmission between SAN and NAS devices.

Data center network technologies have operational requirements for scalability and high availability that are fulfilled by employing redundant and/or fault-tolerant configurations. These five network subsystems improve data center redundancy and reliability to ensure that they have enough IT resources to maintain a certain level of service even in the face of multiple failures.

Ultra-high-speed network optical links can be used to aggregate individual gigabit-per-second channels into single optical fibers using multiplexing technologies like dense wavelength-division multiplexing (DWDM). Spread over multiple locations and used to interconnect server farms, storage systems, and replicated data centers, optical links improve transfer speeds and resiliency.

Serverless Environments

A serverless environment consists of technologies that automatically provide runtime resources for applications that can be deployed without the need to set up the underlying resources required for them to run. The deployed logic still runs on servers, whether

physical, virtual, containerized, or otherwise, but service administrators do not need be concerned with capacity planning, management, resiliency, or elasticity configurations, which are aspects taken care of by the serverless environment.

Serverless technologies include automation, virtualization, infrastructure and software deployment and management, Infrastructure as Code, and Continuous Deployment, all encompassed in a highly customized cloud service that allows developers to simply upload their code and an accompanying description of its runtime requirements in a language specific to each cloud provider. The serverless environment then takes over from there.

A serverless environment is most commonly provided and operated by a public cloud provider that relies on either container engines or virtual machines to isolate the runtime of one application from another. The runtime details are hidden from the cloud consumer, and the cloud provider is responsible for managing the low-level infrastructure, including operating systems, virtual machines, and containers.

Resources required by programs deployed using these serverless technologies are billed by cloud providers only for the time that the programs actually run. When the program is not running, it does not generate any cost. This can be considered one of the most important advantages of serverless technologies, along with the ease of use provided to development teams by the automation of the entire deployment process all the way into production.

NoSQL Clustering

NoSQL (short for "Not only SQL") refers to technologies used to develop next-generation non-relational databases that are highly scalable and fault tolerant. These technologies achieve high levels of scalability and fault tolerance because they are designed as clusters of servers that act as a single database or storage entity. This is known as NoSQL clustering.

A cluster is a centrally managed group of nodes connected together via a network to process tasks in parallel, where each node is responsible for a subtask of a larger problem (Figure 5.8). Clusters enable distributed data processing. Ideally, a cluster comprises low-cost commodity nodes that collectively provide increased processing capacity with inherent redundancy and fault tolerance—features made possible by the fact that the cluster consists of physically separate nodes.

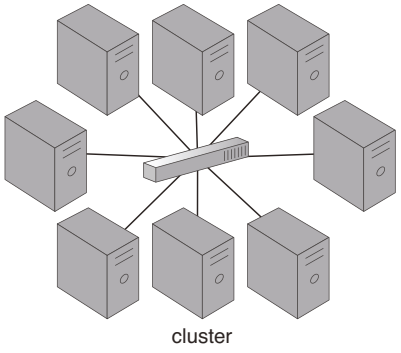


Figure 5.8
A cluster can be used as a deployment environment for various types of cloud-based solutions, including NoSQL databases.

Clusters are highly scalable, supporting horizontal scaling with linear performance gains. They provide an ideal deployment environment for a processing engine, as large datasets can be divided into smaller datasets and then processed in parallel in a distributed manner.

Clusters are a fundamental resource provided by cloud computing platforms. Clustering technology is used to provide big data platform-related services, advanced container management environments, development of applications that scale automatically, and deployment environments (such as PaaS), among others.

NoSQL clustering provides storage devices that are scalable, available, fault-tolerant, and very fast for read/write operations. However, these devices do not provide the same transaction and consistency support as exhibited by relational database management systems (RDBMSs).

Following are some of the principal features of NoSQL storage devices:

- *Schemaless Data Model* – Data can exist in its raw form.
- *Scale Out Rather Than Scale Up* – Additional nodes are added as required, rather than replacing an existing node with a better, higher-performance node.
- *Highly Available* – NoSQL storage devices are built on cluster-based technologies that provide fault tolerance out of the box.
- *Lower Operational Costs* – The devices are built on open-source platforms with no licensing costs and can be deployed on commodity hardware.

- *Eventual Consistency* – Reads across multiple nodes may not be consistent immediately after a write. However, all nodes will eventually be in a consistent state.
- *BASE, not ACID* – BASE compliance requires a database to maintain high availability in the event of network or node failure, while not requiring the database to be in a consistent state whenever an update occurs. The database can be in a soft or inconsistent state until it eventually attains consistency.
- *API-Driven Data Access* – Data access is generally supported via API-based queries, including RESTful APIs. Some implementations may also provide SQL-like query capability.
- *Auto Sharding and Replication* – To support horizontal scaling and provide high availability, a NoSQL storage device automatically employs sharding and replication techniques whereby the dataset is partitioned horizontally and then copied over to multiple nodes.
- *Integrated Caching* – This feature eliminates the need for a third-party distributed caching layer, such as Memcached.
- *Distributed Query Support* – NoSQL storage devices maintain consistent query behavior across multiple shards.
- *Polyglot Persistence* – The use of NoSQL device storage does not mandate retiring traditional RDBMSs. Both types of storage can be used at the same time, thereby supporting polyglot persistence, which is an approach for persisting data using different types of storage technologies. This is helpful for developing systems requiring structured as well as semi-structured or unstructured data.
- *Aggregate-Focused* – Unlike relational databases that are most effective with fully normalized data, NoSQL storage devices store denormalized aggregated data (an entity containing merged, often nested, data for an object), thereby eliminating the need for joins and mapping between application objects and the data stored in the database.

Other Considerations

IT hardware is subject to rapid technological obsolescence, with lifecycles that typically last between five and seven years. The ongoing need to replace equipment frequently results in a mix of hardware whose heterogeneity can complicate the entire data center's operations and management, although this can be partially mitigated through virtualization.

Security is another major issue when considering the role of the data center and the vast quantities of data contained within its doors. Even with extensive security precautions in place, housing data exclusively at one data center facility means much more can be compromised by a successful security incursion than if data was distributed across individual unlinked components.

5.3 Modern Virtualization

Modern virtualization technology is a foundation of contemporary cloud platforms. It provides a variety of virtualization types and technology layers, which are introduced in this section.

Hardware Independence

The installation of an operating system's configuration and application software in a unique IT hardware platform results in many software–hardware dependencies. In a nonvirtualized environment, the operating system is configured for specific hardware models and requires reconfiguration if these IT resources need to be modified.

Virtualization is a conversion process that translates unique IT hardware into emulated and standardized software-based copies. Through hardware independence, virtual servers can easily be moved to another virtualization host, automatically resolving multiple hardware–software incompatibility issues. As a result, cloning and manipulating virtual IT resources is much easier than duplicating physical hardware. The architectural models explored in Part III of this book provide numerous examples of this.

Server Consolidation

The coordination function that is provided by the virtualization software allows multiple virtual servers to be simultaneously created in the same virtualization host. Virtualization technology enables different virtual servers to share one physical server. This process is called *server consolidation* and is commonly used to increase hardware utilization, load balancing, and optimization of available IT resources. The resulting flexibility is such that different virtual servers can run different guest operating systems on the same host.

This fundamental capability directly supports common cloud features, such as on-demand usage, resource pooling, elasticity, scalability, and resiliency.

Resource Replication

Virtual servers are created as virtual disk images that contain binary file copies of hard disk content. These virtual disk images are accessible to the host's operating system, meaning simple file operations, such as copy, move, and paste, can be used to replicate, migrate, and back up the virtual server. This ease of manipulation and replication is one of the most salient features of virtualization technology, as it enables:

- The creation of standardized virtual machine images commonly configured to include virtual hardware capabilities, guest operating systems, and additional application software, for prepackaging in virtual disk images in support of instantaneous deployment.
- Increased agility in the migration and deployment of a virtual machine's new instances by being able to rapidly scale out and up.
- The ability to roll back, which is the instantaneous creation of VM snapshots by saving the state of the virtual server's memory and hard disk image to a host-based file. (Operators can easily revert to these snapshots and restore the virtual machine to its prior state.)
- The support of business continuity with efficient backup and restoration procedures, as well as the creation of multiple instances of critical IT resources and applications.

Operating System–Based Virtualization

Operating system–based virtualization is the installation of virtualization software in a preexisting operating system, which is called the *host operating system* (Figure 5.9). For example, a user whose workstation is installed with a specific version of Windows wants to generate virtual servers and installs virtualization software into the host operating system like any other program. This user needs to use this application to generate and operate one or more virtual servers. The user needs to use virtualization software to enable direct access to any of the generated virtual servers. Since the host operating system can provide hardware devices with the necessary support, operating system virtualization can rectify hardware compatibility issues even if the hardware driver is not available to the virtualization software.

Hardware independence that is enabled by virtualization allows hardware IT resources to be more flexibly used. For example, consider a scenario in which the host operating system has the software necessary for controlling five network adapters that are

available to the physical computer. The virtualization software can make the five network adapters available to the virtual server, even if the virtualized operating system is incapable of physically housing five network adapters.

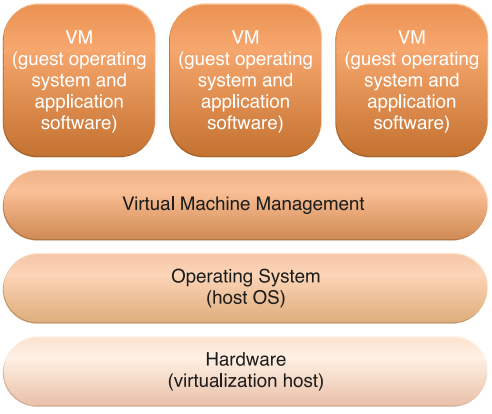


Figure 5.9
The different logical layers of operating system–based virtualization, in which the VM is first installed into a full host operating system and subsequently used to generate virtual machines.

Virtualization software translates hardware IT resources that require unique software for operation into virtualized IT resources that are compatible with a range of operating systems. Since the host operating system is a complete operating system in itself, many operating system–based services that are available as administration tools can be used to manage the physical host.

Examples of such services include:

- Backup and Recovery
- Integration to Directory Services
- Security Management

Operating system–based virtualization can introduce demands and issues related to performance overhead, such as:

- The host operating system consumes CPU, memory, and other hardware IT resources.
- Hardware-related calls from guest operating systems need to traverse several layers to and from the hardware, which decreases overall performance.
- Licenses are usually required for host operating systems, in addition to individual licenses for each of their guest operating systems.

A concern with operating system–based virtualization is the processing overhead required to run the virtualization software and host operating systems. Implementing a virtualization layer will negatively affect overall system performance. Estimating, monitoring, and managing the resulting impact can be challenging because it requires expertise in system workloads, software and hardware environments, and sophisticated monitoring tools.

Hardware-Based Virtualization

This option represents the installation of virtualization software directly on the physical host hardware so as to bypass the host operating system, which is presumably engaged with operating system–based virtualization (Figure 5.10). Allowing the virtual servers to interact with hardware without requiring intermediary action from the host operating system generally makes hardware-based virtualization more efficient.

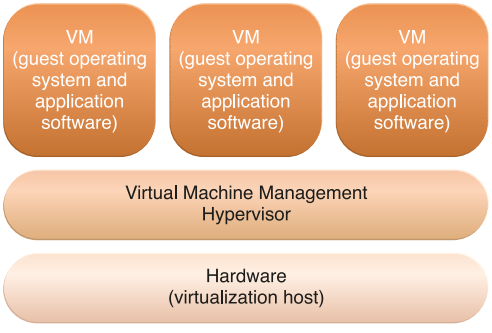


Figure 5.10
The different logical layers of hardware-based virtualization, which does not require another host operating system.

Virtualization software is typically referred to as a *hypervisor* for this type of processing. A hypervisor has a simple user interface that requires a negligible amount of storage space. It exists as a thin layer of software that handles hardware management functions to establish a virtualization management layer. Device drivers and system services are optimized for the provisioning of virtual servers, although many standard operating system functions are not implemented. This type of virtualization system is essentially used to optimize performance overhead inherent to the coordination that enables multiple virtual servers to interact with the same hardware platform.

One of the main issues of hardware-based virtualization concerns compatibility with hardware devices. The virtualization layer is designed to communicate directly with the host hardware, meaning all the associated device drivers and support software

need to be compatible with the hypervisor. Hardware device drivers may not be as available to hypervisor platforms as they are to operating systems. Host management and administration features may further not include the range of advanced functions that are common to operating systems.

Containers and Application-Based Virtualization

Application virtualization is a method of creating and using applications without operating system dependency. For many types of applications and services, containers provide a portable, compatible, and highly manageable deployment environment that allows independent and autonomous software programs and systems to run on almost any platform, in accordance with the definition of application virtualization.

Software running in containers can be deployed almost anywhere, always providing the same functionality independently of the runtime environment in which it is deployed. Containers are suitable for application-based virtualization because applications running in containers can run on any platform, regardless of the underlying operating system or hardware architecture, as long as a compatible containerization engine is running on that platform, as depicted in Figure 5.11.

Containerization has become a fundamental infrastructure technology in contemporary cloud environments and is covered in detail in Chapter 6.

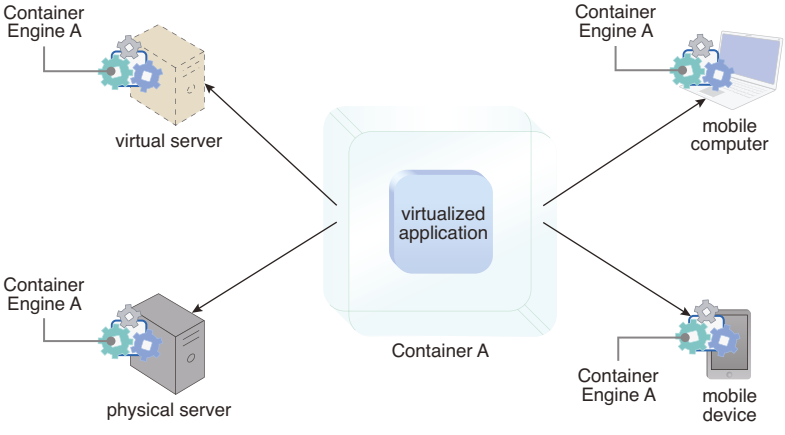


Figure 5.11

A virtualized application running in a container can be deployed anywhere its corresponding containerization engine is installed, regardless of underlying hardware or operating system architectures.

Virtualization Management

Many administrative tasks can be performed more easily using virtual servers as opposed to using their physical counterparts. Modern virtualization software provides several advanced management functions that can automate administration tasks and reduce the overall operational burden on virtualized IT resources.

Virtualized IT resource management is often supported by *virtualization infrastructure management (VIM)* tools that collectively manage virtual IT resources and rely on a centralized management module, otherwise known as a controller, that runs on a dedicated computer. VIMs are commonly encompassed by the resource management system mechanism described in Chapter 12.

Other Considerations

- *Performance Overhead* – Virtualization may not be ideal for complex systems that have high workloads with little use for resource sharing and replication. A poorly formulated virtualization plan can result in excessive performance overhead. A common strategy used to rectify the overhead issue is a technique called para-virtualization, which presents a software interface to the virtual machines that is not identical to that of the underlying hardware. The software interface has instead been modified to reduce the guest operating system's processing overhead, which is more difficult to manage. A major drawback of this approach is the need to adapt the guest operating system to the para-virtualization API, which can impair the use of standard guest operating systems while decreasing solution portability.
- *Special Hardware Compatibility* – Many hardware vendors that distribute specialized hardware may not have device driver versions that are compatible with virtualization software. Conversely, the software itself may be incompatible with recently released hardware versions. These types of incompatibility issues can be resolved by using established commodity hardware platforms and mature virtualization software products. Container engines are not affected by this consideration because they run on top of the host operating system, which abstracts any potential hardware compatibility, making containerization a highly portable type of virtualization technology.
- *Portability* – The programmatic and management interfaces that establish administration environments for a virtualization program to operate with various virtualization solutions can introduce portability gaps due to incompatibilities. Initiatives

such as the Open Virtualization Format (OVF) for the standardization of virtual disk image formats are dedicated to alleviating this concern. Furthermore, containerization provides an alternative type of virtualization technology with very high levels of portability.

5.4 Multitenant Technology

The multitenant application design was created to enable multiple users (tenants) to access the same application logic simultaneously. Each tenant has its own view of the application that it uses, administers, and customizes as a dedicated instance of the software while remaining unaware of other tenants that are using the same application.

Multitenant applications ensure that tenants do not have access to data and configuration information that is not their own. Tenants can individually customize features of the application, such as:

- *User Interface* – Tenants can define a specialized "look and feel" for their application interface.
- *Business Process* – Tenants can customize the rules, logic, and workflows of the business processes that are implemented in the application.
- *Data Model* – Tenants can extend the data schema of the application to include, exclude, or rename fields in the application data structures.
- *Access Control* – Tenants can independently control the access rights for users and groups.

Multitenant application architecture is often significantly more complex than that of single-tenant applications. Multitenant applications need to support the sharing of various artifacts by multiple users (including portals, data schemas, middleware, and databases), while maintaining security levels that segregate individual tenant operational environments.

Common characteristics of multitenant applications include:

- *Usage Isolation* – The usage behavior of one tenant does not affect the application availability and performance of other tenants.
- *Data Security* – Tenants cannot access data that belongs to other tenants.
- *Recovery* – Backup and restore procedures are separately executed for the data of each tenant.

- *Application Upgrades* – Tenants are not negatively affected by the synchronous upgrading of shared software artifacts.
- *Scalability* – The application can scale to accommodate increases in usage by existing tenants and/or increases in the number of tenants.
- *Metered Usage* – Tenants are charged only for the application processing and features that are actually consumed.
- *Data Tier Isolation* – Tenants can have individual databases, tables, and/or schemas isolated from other tenants. Alternatively, databases, tables, and/or schemas can be designed to be intentionally shared by tenants.

A multitenant application that is being concurrently used by two different tenants is illustrated in Figure 5.12. This type of application is typical with SaaS implementations.

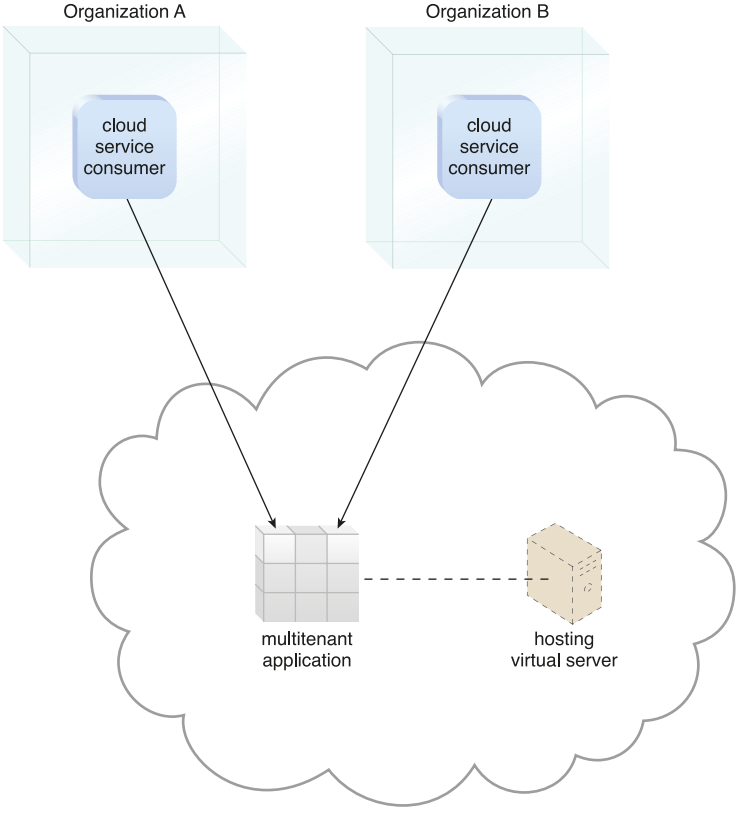


Figure 5.12
A multitenant application that is serving multiple cloud service consumers simultaneously.

MULTITENANCY VS. VIRTUALIZATION

Multitenancy is sometimes mistaken for virtualization because the concept of multiple tenants is similar to the concept of virtualized instances.

The differences lie in what is multiplied within a physical server acting as a host:

- With virtualization: Multiple virtual copies of the server environment can be hosted by a single physical server. Each copy can be provided to different users, can be configured independently, and can contain its own operating systems and applications.
- With multitenancy: A physical or virtual server hosting an application is designed to allow usage by multiple different users. Each user feels as though they have exclusive usage of the application.

5.5 Service Technology and Service APIs

The field of service technology is a keystone foundation of cloud computing that formed the basis of the "as a service" cloud delivery models. Several prominent service technologies that are used to realize and build upon cloud-based environments are described in this section.

ABOUT WEB-BASED SERVICES

Reliant on the use of standardized protocols, *web-based services* are self-contained units of logic that support interoperable machine-to-machine interaction over a network. These services are generally designed to communicate via nonproprietary technologies in accordance with industry standards and conventions. Because their sole function is to process data between computers, these services expose APIs and do not have user interfaces. Web services and REST services represent two common forms of web-based services.

REST Services

REST services are designed according to a set of constraints that shape the service architecture to emulate the properties of the World Wide Web, resulting in service implementations that rely on the use of core web technologies.

The six REST design constraints are:

- Client-Server
- Stateless
- Cache
- Interface/Uniform Contract
- Layered System
- Code-On-Demand

REST services do not have individual technical interfaces, but instead share a common technical interface that is known as the uniform contract, which is typically established via the use of HTTP methods.

NOTE

To learn more about REST services, read *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST* from the *Pearson Digital Enterprise Series from Thomas Erl*.

Web Services

Also commonly prefixed with "SOAP-based," web services represent an established and common medium for sophisticated, web-based service logic. Along with XML, the core technologies behind web services are represented by the following industry standards:

- *Web Service Description Language (WSDL)* – This markup language is used to create a WSDL definition that defines the application programming interface (API) of a web service, including its individual operations (functions) and each operation's input and output messages.
- *XML Schema Definition Language (XML Schema)* – Messages exchanged by web services must be expressed using XML. XML schemas are created to define the data structure of the XML-based input and output messages exchanged by web services. XML schemas can be directly linked to or embedded within WSDL definitions.
- *SOAP* – Formerly known as the Simple Object Access Protocol, this standard defines a common messaging format used for request and response messages exchanged by web services. SOAP messages are comprised of body and header

sections. The former houses the main message content, and the latter is used to contain metadata that can be processed at runtime.

- *Universal Description, Discovery, and Integration (UDDI)* – This standard regulates service registries in which WSDL definitions can be published as part of a service catalog for discovery purposes.

These four technologies collectively form the first generation of web service technologies (Figure 5.13). A comprehensive set of second-generation web service technologies (commonly referred to as WS-*) has been developed to address various additional functional areas, such as security, reliability, transactions, routing, and business process automation.

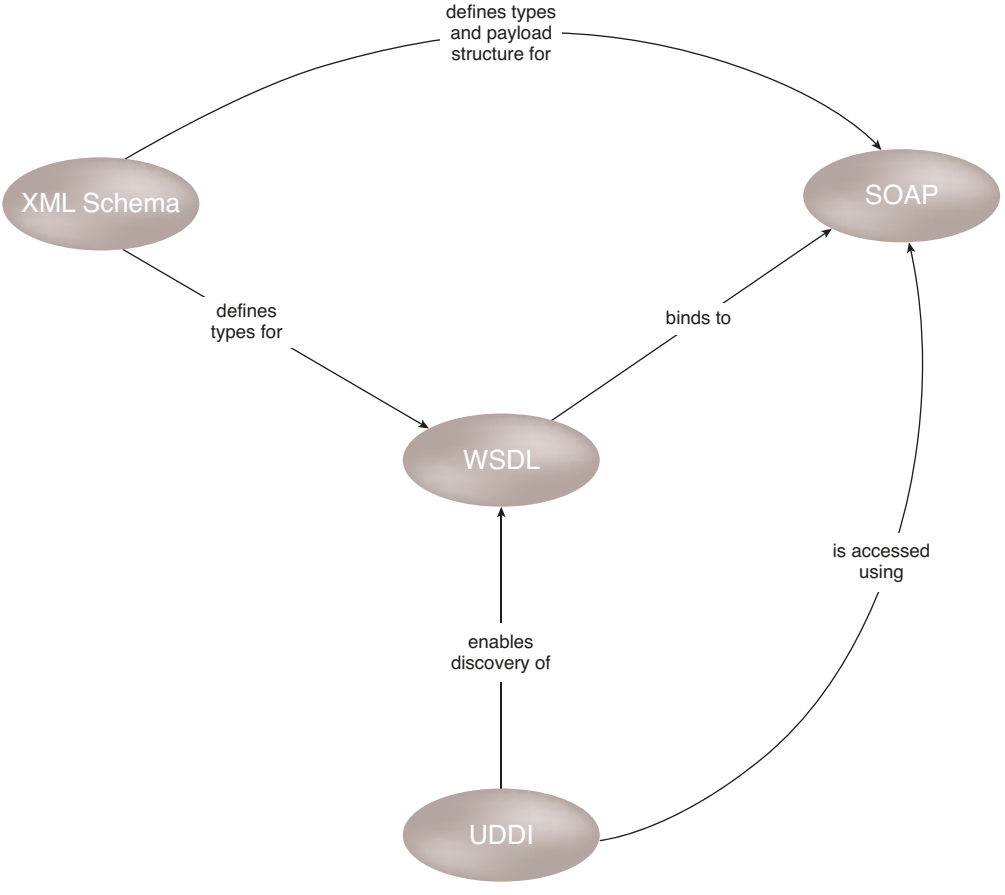


Figure 5.13

An overview of how first-generation web service technologies commonly relate to each other.

NOTE

To learn more about web service technologies, read *Web Service Contract Design & Versioning for SOA* from the *Pearson Digital Enterprise Series from Thomas Erl.* This title covers first- and second-generation web service standards in technical detail.

Service Agents

Service agents are event-driven programs designed to intercept messages at runtime. There are active and passive service agents, both of which are common in cloud environments. Active service agents perform an action upon intercepting and reading the contents of a message. This action typically requires making changes to the message contents (most commonly message header data and less commonly the body content) or changes to the message path itself. Passive service agents, on the other hand, do not change message contents. Instead, they read the message and may then capture certain parts of its contents, usually for monitoring, logging, or reporting purposes.

Cloud-based environments rely heavily on the use of system-level and custom service agents to perform much of the runtime monitoring and measuring required to ensure that features, such as elastic scaling and pay-for-use billing, can be carried out instantaneously.

Several of the mechanisms described in Part II of this book exist as, or rely on the use of, service agents.

Service Middleware

Falling under the umbrella of service technology is the large market of middleware platforms that evolved from messaging-oriented middleware (MOM) platforms, used primarily to facilitate integration, to sophisticated service middleware platforms designed to accommodate complex service compositions.

The two most common types of middleware platforms relevant to services computing are the enterprise service bus (ESB) and the orchestration platform. The ESB encompasses a range of intermediary processing features, including service brokerage, routing, and message queuing. Orchestration environments are designed to host and execute workflow logic that drives the runtime composition of services.

Both forms of service middleware can be deployed and operated within cloud-based environments.

Web-Based RPC

Cloud providers commonly deliver access to the resources that they offer via RESTful services. The interaction between RESTful services and their service consumers requires considerable amounts of bandwidth in conversations that require multiple messages to be exchanged through the network.

Traditional RPC frameworks can overcome some of the performance challenges presented by RESTful architectures, but they are bound to communication via TCP/IP, which is a limitation incompatible with web-based application requirements. To address both sets of limitations, a set of contemporary protocols was developed that leverage the performance benefits of RPC, while supporting web-based communication. These include:

- gRPC (originally developed by Google)
- GraphQL (originally developed by Facebook)
- Falcor (originally developed by Netflix)

Each of these protocols was developed by an organization in response to a need to overcome limitations with more established protocols.

5.6 CASE STUDY EXAMPLE

DTGOV has assembled cloud-aware infrastructures in each of its data centers, which are comprised of the following components:

- Tier-3 facility infrastructure, which provides redundant configurations for all the central subsystems in the data center facility layer.
- Redundant connections with utility service providers that have installed local capacity for power generation and water supply that activates in the event of general failure.
- An internetwork that supplies an ultra-high bandwidth interconnection between the three data centers through dedicated links.
- Redundant internet connections in each data center to multiple ISPs and the *.GOV* extranet, which interconnects DTGOV with its main government clients.
- Standardized hardware of higher aggregated capacity that is abstracted by a cloud-aware virtualization platform.

Physical servers are organized on server racks, each of which has two redundant top-of-rack router switches (layer 3) that are connected to each physical server. These router switches are interconnected to LAN core-switches that have been configured as a cluster. The core-switches connect to routers that supply internetworking capabilities and firewalls that provide network access control capabilities. Figure 5.14 illustrates the physical layout of the server network connections inside of the data center.

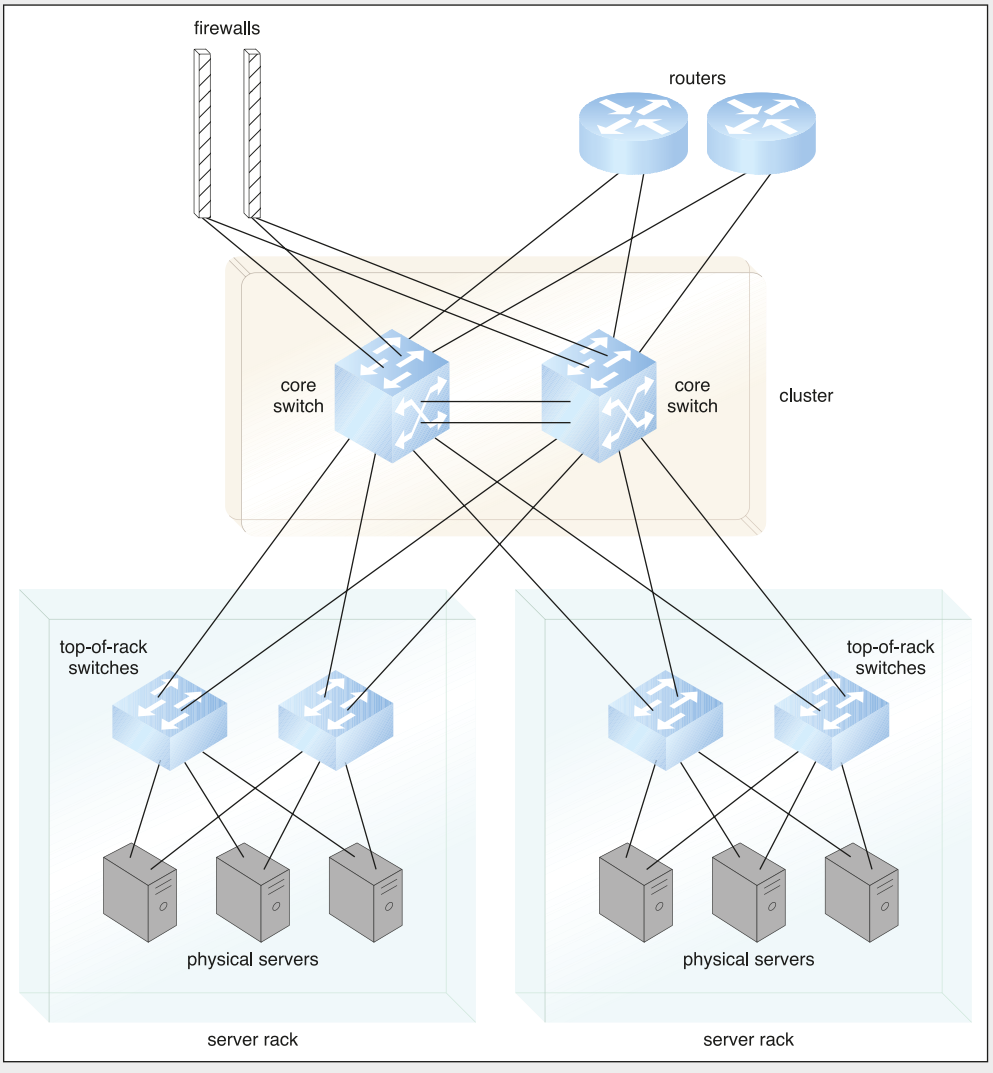


Figure 5.14
A view of the server network connections inside the DTGOV data center.

A separate network that connects the storage systems and servers is installed with clustered storage area network (SAN) switches and similar redundant connections to various devices (Figure 5.15).

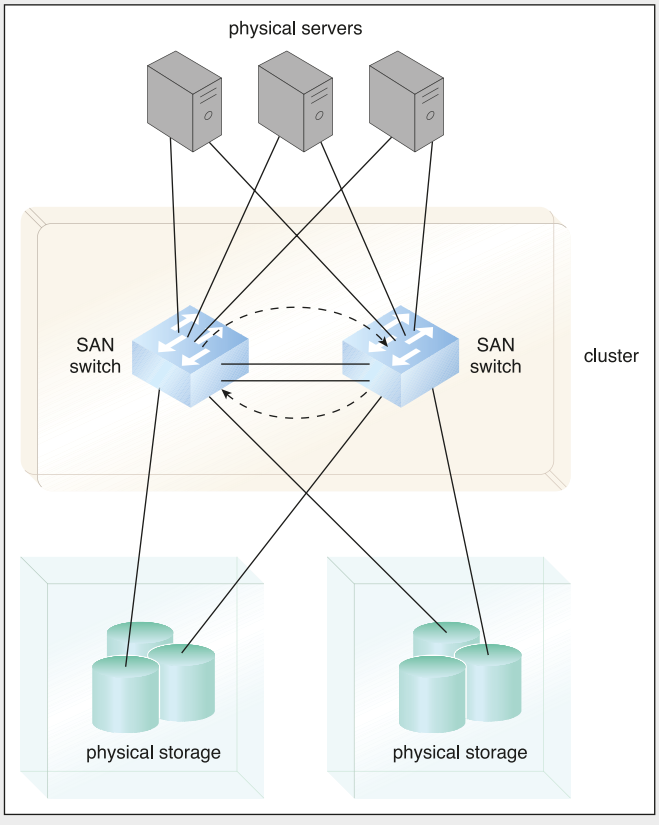


Figure 5.15
A view of the storage system network connections inside the DTGOV data center.

Figure 5.16 illustrates an internetworking architecture that is established between every data center pair within the DTGOV corporate infrastructure.

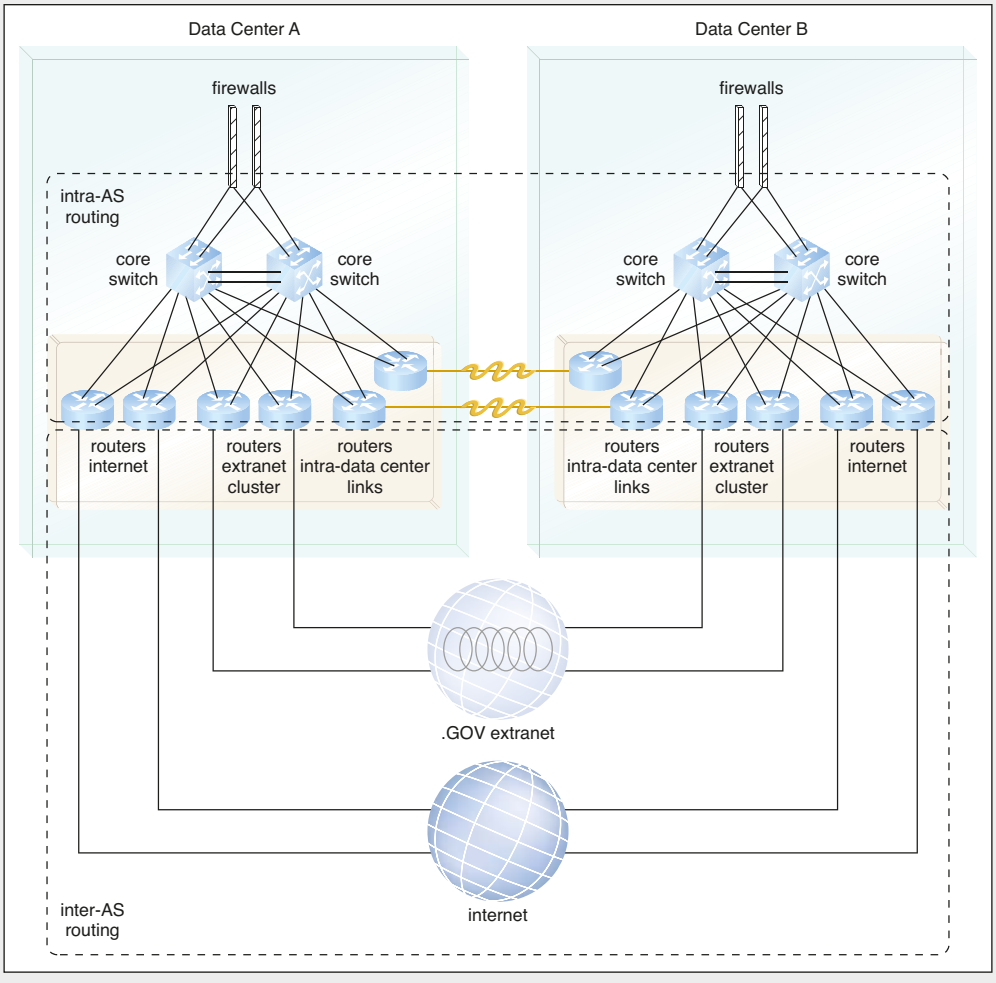


Figure 5.16

The internetworking setup between two data centers that is similarly implemented between every pair of DTGOV data centers. The DTGOV internetwork is designed to be an autonomous system (AS) on the internet, meaning the links interconnecting the data centers with the LANs define the intra-AS routing domain. The interconnections to external ISPs are controlled through inter-AS routing technology, which shapes internet traffic and enables flexible configurations for load balancing and failover.

As shown in Figures 5.15 and 5.16, combining interconnected physical IT resources with virtualized IT resources on the physical layer enables the dynamic and well-managed configuration and allocation of virtual IT resources.

Chapter 6

Understanding Containerization

Containerization is a virtualization technology used to deploy and run applications and services without the need to deploy a virtual server for each solution. This chapter covers fundamental topics pertaining to virtualization and then takes a close look at containerization technology and the utilization of containers.

NOTE

This chapter is supplemented with coverage of the Docker and Kubernetes containerization technologies provided in Appendix B.

6.1 Origins and Influences

A Brief History

The concept of containers has been present since the 1970s, when it originally referred to a capability used in Unix systems to better segregate application code. Early containers provided an isolated environment in which services and applications could operate without interfering with other processes, resulting in an environment similar to a sandbox for testing apps, services, and other processes.

Containers gained widespread usage decades later due to a slew of Linux distributions that released new deployment and management tools. Containers running on Linux systems were turned into an operating system–level virtualization technique specially designed to enable several isolated Linux environments to run on a single Linux host. However, while running containers on a Linux platform broadened their usefulness, there remained key obstacles to solve, including unified administration, true portability, compatibility, and control of scale.

The introduction of Apache Mesos, Google Borg, and Facebook Tupperware—all of which provided varied degrees of container orchestration and cluster management capabilities—marked a significant advancement in the use of containers on Linux systems. These systems enabled the instant creation of hundreds of containers, as well as automatic failover and other mission-critical functionality necessary for container

management at scale. After Docker containers were introduced, containerization began becoming part of the IT mainstream. Docker's prominence led to the innovation of sophisticated containerization platforms, including Marathon, Kubernetes, and Docker Swarm.

Containerization and Cloud Computing

Cloud computing helped popularize virtualization technology, and further advances in cloud computing technology helped realize contemporary containerization technology. Containerization is now a fundamental part of cloud computing infrastructure.

The use of containers can help support the primary business drivers behind cloud computing.

The simplified and flexible deployment architecture established by containerization can directly support the primary Cost Reduction and Business Agility business drivers behind cloud computing (as introduced in Chapter 3) and can further enable cloud-based solutions to better respond to fluctuating usage requirements.

6.2 Fundamental Virtualization and Containerization

This section covers the fundamental terms and concepts associated with operating systems and virtualization technology. It then proceeds to explain the basic components of containerization and concludes with a comparison of virtualization and containerization.

Operating System Basics

An *operating system* is software installed on a computer that provides a range of programs, tools, libraries, and other resources used to manage a computer, as well as programs used to host and support the ongoing operations of applications installed on the operating system. The installation of an operating system can also include various consumer applications.

The operating system programs used to support the execution and active operation of applications are collectively referred to as the *runtime* (Figure 6.1). Applications themselves may introduce their own runtime software that operates on top of an operating system runtime environment.

Figure 6.1
The symbol used to represent a runtime.

Virtualization Basics

To best understand containerization, it is important to first establish some basics about virtualization. As has already been explained in Chapter 3, *virtualization* is the technology that enables a physical IT resource to provide multiple virtual images of itself so that its underlying processing capabilities can be shared by multiple solutions.

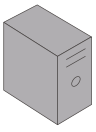


Figure 6.2
The symbol used to represent a physical server.

Physical Servers

The physical IT resource most commonly virtualized is the *physical server* (Figure 6.2). A physical server provides an operating system environment that can host applications, services, and other software programs.

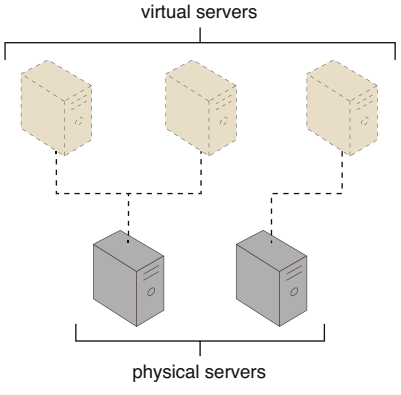
Figure 6.3
The symbol used to represent a virtual server.

Virtual Servers

When utilizing virtualization technology, the operating system hosting environment provided by the physical server can be abstracted into one or more *virtual servers* (Figure 6.3).

Each virtual server can then provide a fresh and dedicated copy (or *image*) of the operating system hosting environment, which can be further referred to as a *guest* operating system. Each virtual server can provide its virtualized operating system environment to a different set of consumer applications or services that do not require any knowledge of how the underlying physical server exists or operates (Figure 6.4). As consumer usage demands fluctuate, the physical server can be scaled accordingly.

Figure 6.4
Three virtual servers that exist on two physical servers.



The administrator responsible for the physical server can retain administrative control of the physical server hardware and its operating system. The administrators responsible for the individual virtual servers are not given (nor require) access to the underlying physical server, but they can independently control their respective virtual operating system environments.

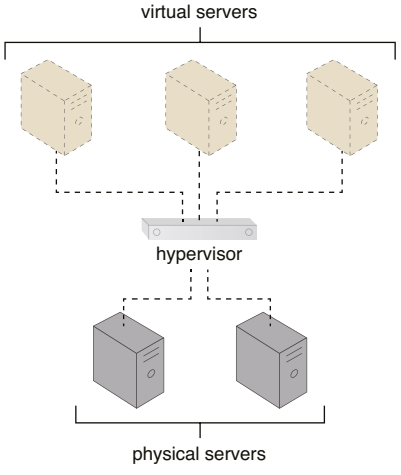
Figure 6.5
The symbol used to represent the hypervisor.

Hypervisors

The component responsible for creating and running multiple virtual servers from a physical server is the *hypervisor* (Figures 6.5 and 6.6).

Virtual servers perceive the emulated hardware presented to them by the hypervisor as real hardware. Each virtual server has its own operating system (also known as a guest operating system) that needs to be deployed inside the virtual server and managed and maintained as if it were deployed on a physical server.

Figure 6.6
Three virtual servers created and run by a hypervisor that exists on two physical servers.

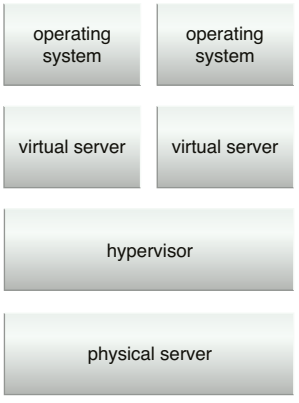


Virtualization Types

There are two types of virtualization environments that are primarily distinguished by whether the physical server has an operating system installed.

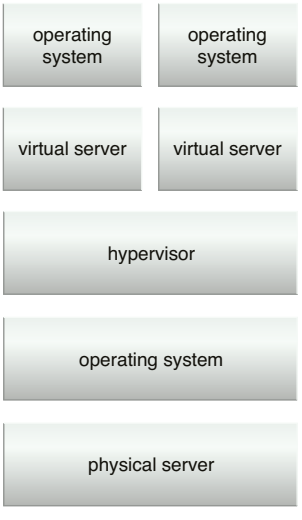
In a Type 1 virtualization environment, the physical server does not have an operating system installed. Instead, only the hypervisor is installed on the physical server, and it is responsible for creating the virtual servers and providing them with virtualized operating system environments (Figure 6.7).

Figure 6.7
The physical server hosts only the hypervisor that creates virtual servers, each with its own operating system.



In a Type 2 virtualization environment, the physical server has an operating system installed and may also have a hypervisor installed. In this case, the physical server is accessible via its operating system, and the hypervisor remains responsible for creating the virtual servers and providing them with their virtualized operating system environments (Figure 6.8).

Figure 6.8
The physical server hosts its own operating system as well as a hypervisor that creates virtual servers with their own operating system environments.



Containerization Basics

Containers

A *container* (Figure 6.9) is a virtualized hosting environment that can be optimized to provide only the resources required for the software programs it hosts.

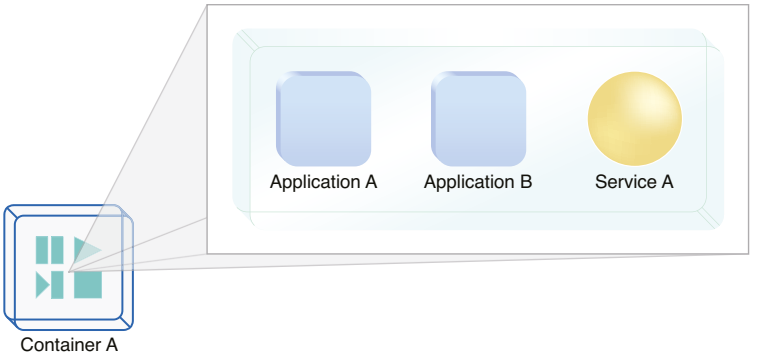


Figure 6.9
The symbol on the left is the container icon. The symbol on the right is also used to represent a container and to show its contents.

Containers have various features and characteristics that are explored in more detail in the upcoming *Understanding Containers* section.

Container Images

A *container image* (Figure 6.10) is similar to a predefined template that is used to create deployed containers.

Figure 6.10
The symbol used to represent a container image.

The definition and usage of container images is integral to how containerization platforms operate. Further details are provided in the upcoming *Understanding Container Images* section.

Container Engines

The *container engine* (Figure 6.11), also referred to as the *containerization engine*, is responsible for creating containers based on predefined container images. The container engine is deployed in a physical or virtual server's operating system from where it can abstract the resources required for a given container.

Figure 6.11
The symbol used to represent the container engine.

The container engine is a core part of a containerization platform and is responsible for many of its primary processing tasks. Its implementation is organized into two "planes," as follows:

- *Management Plane* – the GUI and command-line tools made available to enable human administrators to configure and maintain the container engine environment
- *Control Plane* – all remaining container engine functions and features that the container engine carries out automatically and in response to settings and commands issued via the management plane

A given container engine can create multiple containers (Figure 6.12).

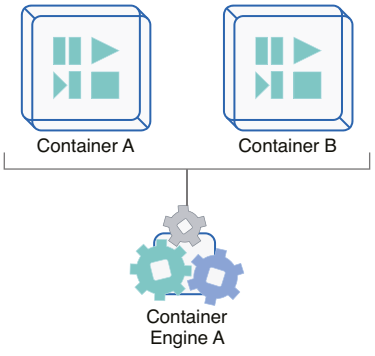


Figure 6.12
A container engine creating two different containers.

Pods

A *pod*, also known as a *logical pod container*, is a special type of system container that can be used to host a single container or a group of containers (Figure 6.13) that have shared storage and/or network resources, and also share the same configuration that determines how the containers are to be run.

How pods relate to container deployment is further explored in the *Containers and Pods* subsection of the upcoming *Understanding Containers* section.

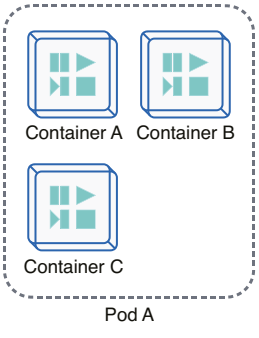


Figure 6.13
A pod is depicted as a perforated outline showing the containers it is hosting.

Hosts

A *host* is the environment in which a container is deployed. A host can be referred to as a *server* or a *node*. The host provides the operating system from which the container abstracts the resources it needs to support the programs it is hosting. Multiple containers can be deployed and run on a single host (Figure 6.14).

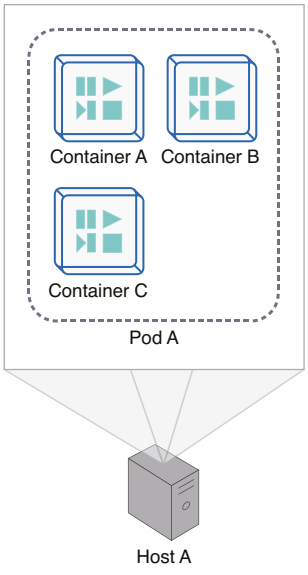


Figure 6.14
Three containers in a single pod reside on Host A, which is a physical server.

Different combinations of containers and pods can be deployed on different hosts (Figure 6.15). However, a single pod cannot span more than one host.

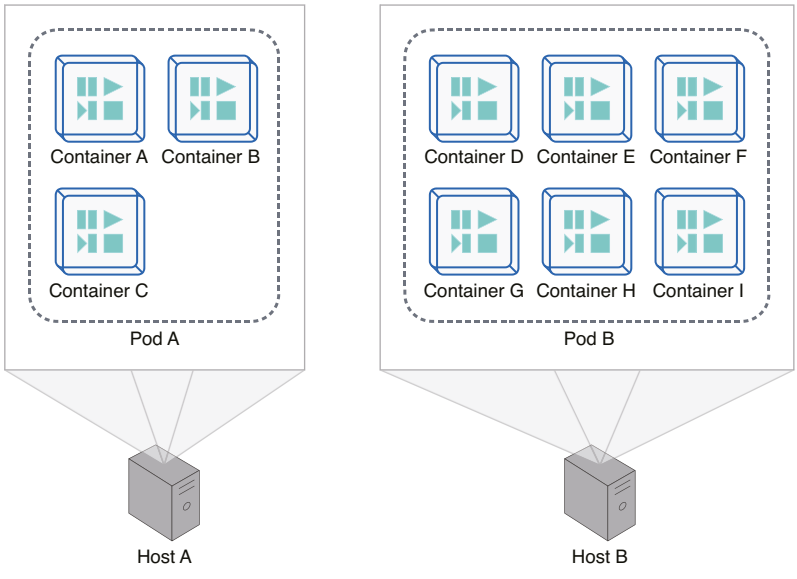


Figure 6.15
Host A has three active containers in a pod, whereas there are six containers in a pod operating on Host B.

Containers also operate on a host without a pod when the container engine deployed does not support pods (Figure 6.16).

Hosts commonly exist as physical servers, but a host can also be a virtual server. When a container is deployed on a virtual server, it is considered a form of *nested virtualization* because one virtualized system is deployed on another.

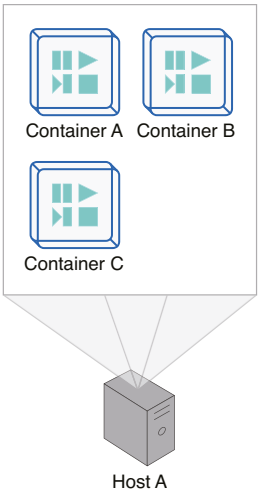


Figure 6.16
Three containers are deployed on Host A without the involvement of a pod.

Host Clusters

Host servers can be combined into "clusters" that can collectively establish a pool of readily available processing resources with increased computing capacity. Both virtual and physical hosts can be clustered (Figures 6.17 and 6.18). Within clustering environments, host servers are commonly referred to as *nodes*.

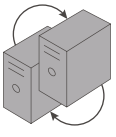


Figure 6.17
The symbol used to represent a physical host cluster.

Common types of host clusters include:

- *Load-Balanced Cluster* – This type of host cluster specializes in distributing workloads among hosts to increase resource capacity while preserving the centralization of resource management. It usually implements a load balancer that is embedded within a cluster management platform or set up as a separate resource.
- *High Availability (HA) Cluster* – This type of cluster maintains system availability in the event of multiple host failures. It typically provides redundant implementations of most or all of the clustered resources and implements a failover system that monitors failure conditions and automatically redirects workloads away from failed host environments.
- *Scaling Cluster* – This type of cluster is used to support both vertical and horizontal scaling.

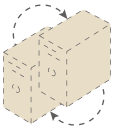


Figure 6.18
The symbol used to represent a virtual host cluster.

Containerization platforms utilize all the aforementioned types of host cluster models in support of high-performance and resiliency requirements, as well as in relation to optimized deployment capabilities.

Host Networks and Overlay Networks

Each host has its own container engine that is responsible for generating container images and deploying and running containers on that host. Related containers within a host can communicate with each other using a local *host network*. Related containers and container engines on different hosts can communicate with each other via an *overlay network*. Both of these types of networks are considered *container networks* (Figure 6.19).

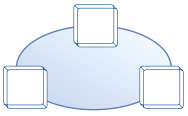


Figure 6.19
The symbol used to represent a container network.

Container networks can be configured by administrators to support various scalability and resiliency capabilities and to control which hosted programs can access resources outside of the container network, as further explored in the *Container Networks* subsection in the upcoming *Understanding Containers* section.

Virtualization and Containerization

The primary distinction between a virtual server and a container is that a virtual server provides a virtual version of a physical server's entire operating system, whereas a container only provides the subset of the operating system resources actually required by the software program (or programs) it is hosting. As a result, a container consumes less space and performs more efficiently than a virtual server.

Containerization on Physical Servers

When deploying containers on a physical server, the containerization platform requires no virtualization environment since virtual servers are not required. The underlying physical server has an operating system installed, and the containerization platform can create containers that each only abstract the subset of the operating system relevant to the software programs it hosts (Figure 6.20).

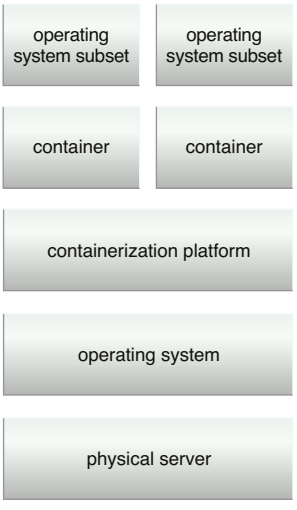


Figure 6.20
A physical server with an operating system hosts a containerization platform that creates containers, each with an environment that has only a subset of the underlying operating system.

Containerization on Virtual Servers

When deploying containers on one or more virtual servers, the containerization platform can be implemented on a Type 1 virtualization environment (Figure 6.21) or a Type 2 virtualization environment with a hypervisor (Figure 6.22). Both types of virtualization environments allow for the creation of virtual servers that can host containerization engines.

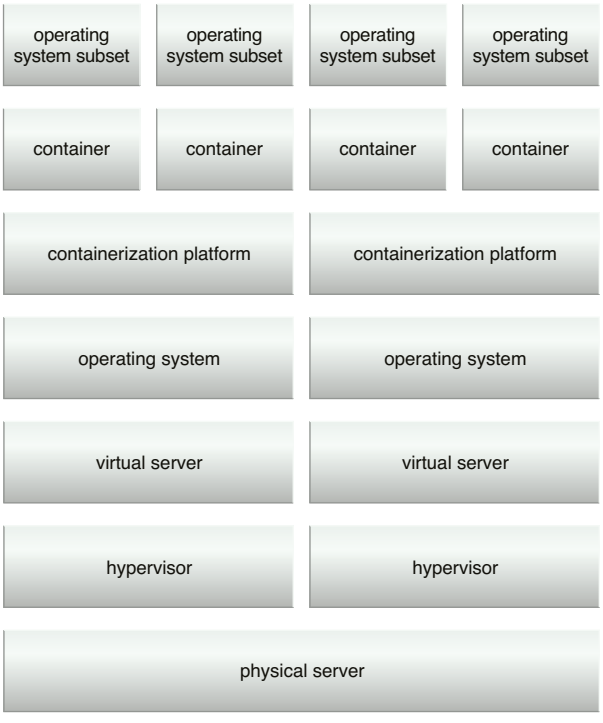


Figure 6.21
A physical server with no operating system hosts a hypervisor that creates virtual servers with operating systems, each of which hosts a containerization platform that can create containers that only have an operating system subset.

The motivation behind deploying containers on virtual servers is often related to security vulnerabilities that exist when the physical server has an operating system installed. As a result, the Type 1 virtualization environment is more common in most production environments. Type 2 virtualization is typically used in development environments when containerized solutions are being built and tested.

Type 2 virtualization can also be used for smaller solutions or for smaller organizations when the underlying physical server needs an operating system in order to host additional programs and systems alongside the containerization platform.

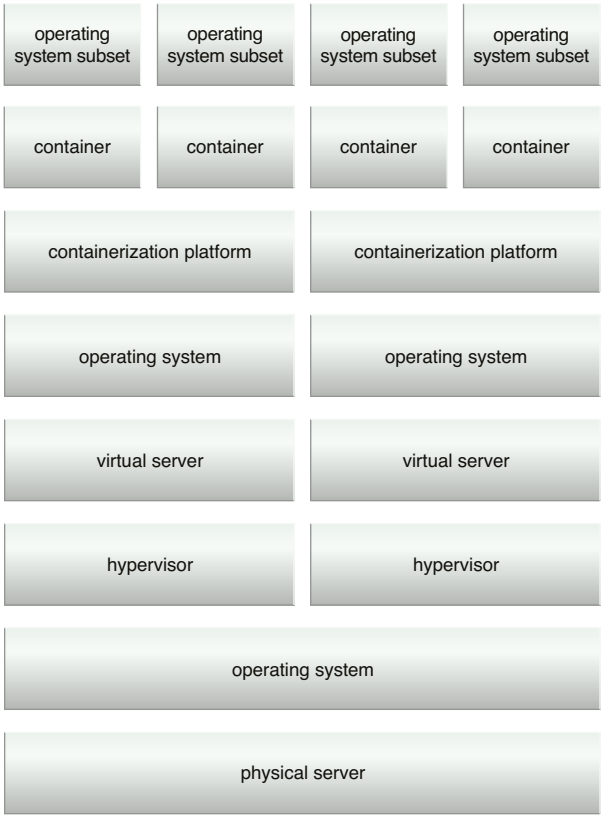


Figure 6.22
A physical server with an operating system hosts a hypervisor that creates virtual server environments with their own operating systems. Each virtual server hosts a containerization platform that creates containers that host a subset of the operating system.

The next two sections highlight key benefits and challenges of utilizing containerization technology with an emphasis on how containers compare to virtual servers.

Containerization Benefits

The following section highlights the key benefits of utilizing containerization technology. Many of these benefits are described in relation to how containers compare to virtual servers.

- *Solution Optimization* – Being able to customize an isolated environment for a solution that minimizes its footprint allows the solution to perform more optimally while demanding only the infrastructure resources it actually requires.
- *Enhanced Scalability* – The reduced CPU, memory, and storage usage footprint of containers allows them to be more effectively and rapidly scaled in response to usage demands.

- *Enhanced Resiliency* – Using special features of container environments, resiliency can be natively provided to ensure that new solution instances are automatically generated in response to failure conditions.
- *Enhanced Deployment Speed* – Containers can be created and deployed faster than virtual servers, which supports rapid deployment and facilitates DevOps approaches, such as continuous integration (CI).
- *Version Support* – Containers allow versions of software code and its dependencies to be tracked. Some platforms allow developers to maintain and track versions of a solution, inspect differences between different versions, and roll back to previous versions, when required.
- *Enhanced Portability* – A containerized solution can be more easily moved across server hosting environments, without the need to change the solution software within the container.

Containerization Risks and Challenges

The following are common risks and challenges of using containerization:

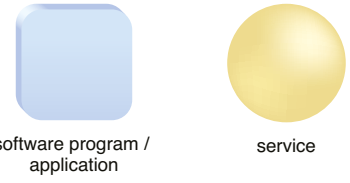
- *Lack of Isolation from Host Operating System* – When multiple containers are deployed on the same physical server, they end up sharing the same host operating system. This means that if the underlying physical server fails or is compromised, all containers running on the server will likely be impacted.
- *Containerization Attack Threat* – Whereas the administrator of a virtual server cannot access or modify the operating system of the underlying physical server, the administrator of a container can, because the operating system's kernel is shared among all containers running on the same physical server. This introduces a significant security vulnerability when containerization platforms are deployed without the involvement of virtual servers.
- *Increased Complexity* – The addition of containerization technology adds new layers and design considerations that can increase both the complexity and the cost of the underlying solution infrastructure. This can introduce effort and risk, as well as an increased learning curve for those responsible for building the solution and its underlying infrastructure environment. The use of containerization may also negatively impact the performance of a solution due to the additional layer of processing it introduces.

- *Increased Administrative Overhead* – Because a given container provides a given version of a solution with only the operating system resources it requires, ongoing administrative effort may be needed to maintain the creation of subsequent container versions that may be needed to accommodate the changing needs of future solution versions. In a virtual server environment, this is less of a concern because the entire operating system is always provided to a solution and its subsequent versions.

6.3 Understanding Containers

While a container can contain any type of software program, it is most commonly used to host applications or services that comprise or are part of a greater automation solution (Figure 6.23).

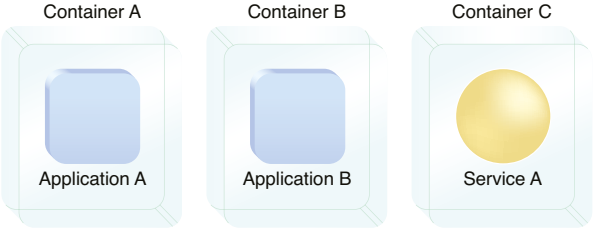
Figure 6.23
The symbol on the left is used to represent a software program that is an application or an application component. The symbol on the right represents a software program that is designed as a service.



Container Hosting

A single container can host a single software program and multiple containers can co-exist, side by side, in the same environment (Figure 6.24). When multiple containers reside in the same underlying environment, they are securely isolated from each other so that each container can operate independently.

Figure 6.24
Three different containers host three different software programs.



A single container can also be used to host multiple related or different software programs (Figure 6.25).

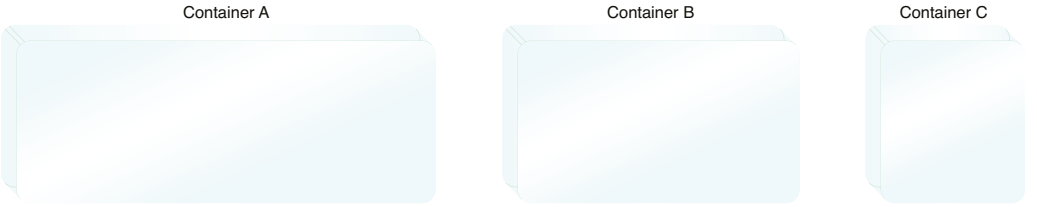


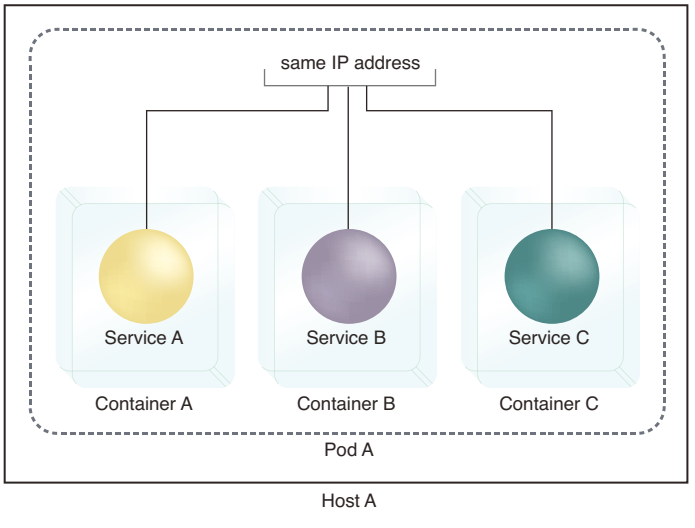
Figure 6.25
Each of the three containers hosts one or more different software programs.

Containers are dynamically generated based on predefined container images, as explained shortly.

Containers and Pods

Grouping individual containers in a pod allows related software programs to be kept together, such as when they are part of the same overall distributed solution (or namespace) and when they need to run under a single IP address (as explained later in the *Container Network Addresses* section) (Figure 6.26). Containers inside a pod can find and discover each other via the host on which the pod is deployed and can communicate with each other using standard interprocess communication methods, such as shared memory. As explained shortly, containers in a pod can also share a file system, dataset, or data storage device.

Figure 6.26
A single pod deployed on a virtual server allows the hosted services to share the same IP address. The pod can also be deployed directly on a physical server.



The pod establishes this environment, while ensuring that hosted programs are still isolated from each other. Pods further provide special containerization capabilities associated with container chains, orchestration, and scaling. The usage of pods is therefore often required by the containerization platform, which is why single pods are frequently used to host single containers.

An administrator creates and configures a pod, and the containers are then added (Figures 6.27 and 6.28).

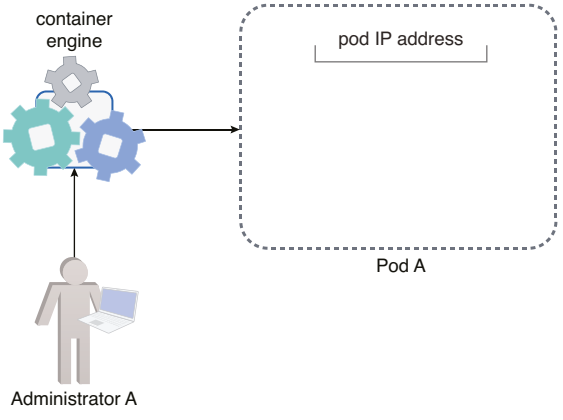


Figure 6.27
An empty Pod A is created by Administrator A.

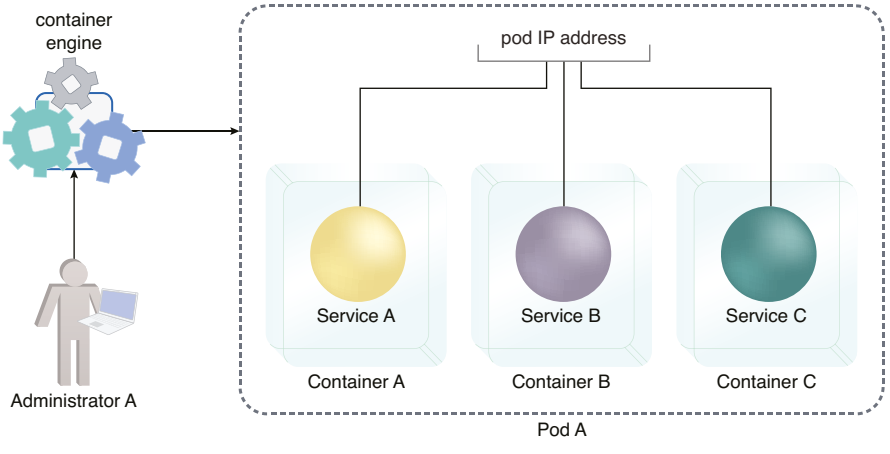


Figure 6.28
Administrator A instructs the container engine to add Containers A, B, and C to Pod A.

A common feature of pods is the ability for the pod to provide common storage to the containers residing in it. The storage usually exists as a file system that is referred to as a *volume*. This form of common storage can be attractive as it offers high-speed access to stored content. Types of files stored in a volume can include log files, media files, and configuration files.

The administrator can configure a pod to enable access to resident containers (Figure 6.29).

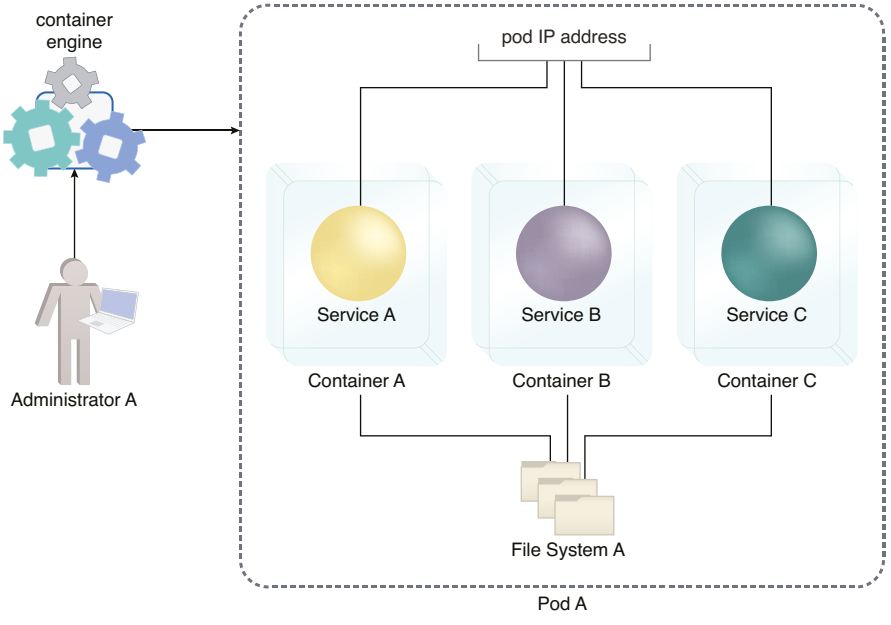


Figure 6.29

The administrator allocates file system storage that is made available to the containers deployed inside the pod.

When deploying a pod hosted on a virtual server, the additional virtualization layer may add runtime processing latency. Depending on the hosted application or service requirements, this can cause performance issues. In some deployment scenarios, the performance may be impacted by other virtual servers hosted on the same host. If the applications or services deployed in the pod are latency sensitive, they can be especially negatively impacted if the pod resides on a virtual server. Performance and latency can be measured before it is determined where to best deploy the pod.

Container Instances and Clusters

Multiple *instances* of the same container with the same software program can be generated (Figure 6.30). This is usually required when concurrent usage of the hosted software program by multiple consumer programs is necessary. Instances of containers are commonly referred to as *replicas*.

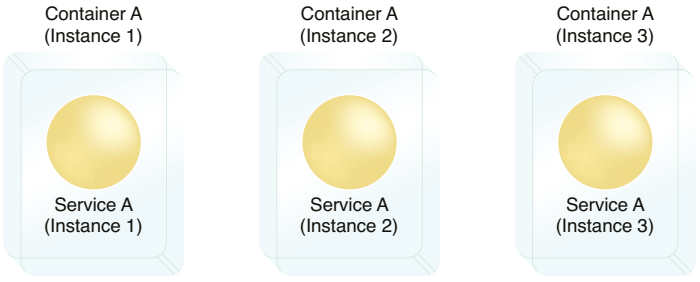


Figure 6.30
Three instances of Container A and its hosted Service A are generated. This allows each instance of Service A to interact with a different consumer program.

Container clusters (Figure 6.31) are pools of container instances that are instantiated in advance of their actual usage. Container clusters can be manually created or automatically generated. They are loaded into memory, where they sit idle, waiting to be invoked. They can be scheduled so that they only reside in memory during predetermined time periods, such as anticipated peak usage times.

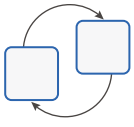


Figure 6.31
The symbol used to represent a container cluster.

Container clusters are primarily created in support of high-performance requirements, often for service-based solutions, to ensure that the containerized service instances can be rapidly provisioned in response to usage demands. Container cluster enviornments can provide auto-scaling capabilities, enabling them to dynamically adjust the size of a cluster based on demand.

Container Package Management

Container package management refers to the process of managing software packages and dependencies within containerized applications. It enables an application and its dependencies to be grouped into a single portable unit called a *package*, which can be deployed on any system that supports the containerization technology.

A *container package manager* is a tool that makes containerized application packaging and distribution easier. It allows container images and their dependencies to be grouped into a single, distributable package that can be deployed and managed across multiple container orchestrators (a mechanism described in the following section).

Figure 6.32
The symbol used to represent a package.

Container package managers typically include a set of command-line tools for creating, tagging, and submitting container images to a container registry, as well as for creating and managing container images and their dependencies. They frequently allow the use of templates or configuration files to define the contents of the package and its dependencies, as well as a method to version and manage the package over time.

Figure 6.33
The symbol used to represent a package repository.

A container package manager is used to coordinate the initial deployment of containers based on predefined workflow logic. The deployment workflow logic is defined in a *package*, also known as a *container deployment file* (Figure 6.32). Typically, host clusters are required to provide a pool of hosts in support of the deployment requirements.

The container deployment file is retrieved from a *package repository* (Figure 6.33).

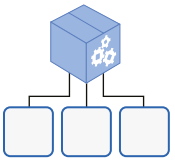


Figure 6.34
The symbol used to represent a container package manager.

The container deployment file is then provided to the container package manager (Figure 6.34).

Before the container package manager carries out the deployment workflow, a special *deployment optimizer* program (Figure 6.35) studies the contents of the package and then assesses available hosts in the cluster to determine the optimal destination for the containers to be deployed.

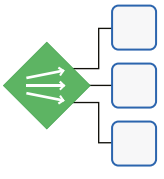


Figure 6.35
The symbol used to represent a deployment optimizer.

Besides the processing capacity of a candidate host, some of the other factors that the deployment optimizer may consider include:

- hardware and software policy limitations
- affinity and anti-affinity specifications
- data locality
- inter-workload interference

Once it has chosen a suitable destination host, the deployment optimizer instructs the container package manager as to where the containers should go. A deployment optimizer can further monitor already deployed containers to ensure their current hosts remain suitable.

NOTE

Within the context of containerization, deployment optimization is often referred to as "scheduling." Furthermore, container package manager and deployment optimizer programs are often limited to deploying containers residing in pods.

Typically, a package represents the containers that comprise an entire solution. Container package managers are therefore created for a set of related containers. In this sense, the package repository can provide a means of application version management.

Examples of what is defined in a package include:

- which host a given container will be deployed on
- which pod a given container will be deployed in
- what sequence a set of containers are deployed in

The administrator authors a package, stores it in the package repository, and then assigns it to the container package manager when it is time for the containers to be deployed (Figure 6.36).

After the deployment, packages are still usually kept in the package repository, as they are often reusable. For example, if a set of containers should need to be ported to a new host, the same container deployment file could be revised with the new host information and then reused.

Docker Compose and Helm are some popular container package managers. These tools make it easier for developers to deploy and manage containerized applications on a variety of container orchestrators (described in the following section) by simplifying the packaging and distribution of containerized applications.

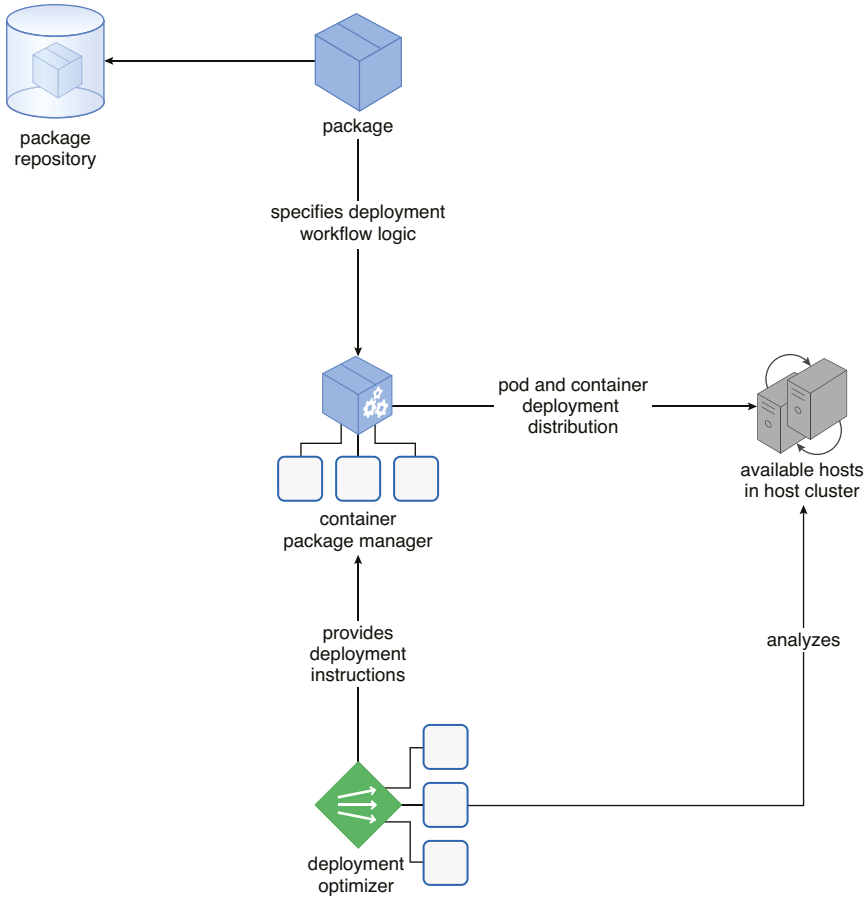


Figure 6.36

The container package manager coordinates the deployment of containers, as per the deployment workflow logic provided in the package and the host deployment instructions it receives from the deployment optimizer.

Container Orchestration

The process of automating the deployment, scaling, and management of containerized applications in a distributed computing environment is known as *container orchestration*. It entails the use of a *container orchestrator*, also referred to as a *container orchestration tool* or *container orchestration platform*.

A container orchestrator performs a wide range of operations in a distributed computing environment. These are some of the key operations performed by a container orchestrator:

- *Container Deployment* – A container orchestrator deploys containers across multiple nodes in a cluster, ensuring that the containers are properly configured and networked.
- *Load Balancing* – The orchestrator distributes traffic across multiple containers running the same application, helping to ensure high availability and scalability.
- *Scaling* – The orchestrator automatically scales up, down, in, or out the number of containers running an application based on demand, helping to ensure optimal resource utilization and cost efficiency.
- *Health Monitoring* – The orchestrator monitors the health of containers and can automatically restart failed containers or replace them with healthy ones.
- *Service Discovery* – The orchestrator maintains a service registry, allowing applications to discover and communicate with each other across the network.
- *Storage Orchestration* – The orchestrator manages the persistent storage needs of containers, ensuring that data is stored and retrieved correctly.
- *Network Orchestration* – The orchestrator manages the networking needs of containers, providing each container with a unique IP address and routing network traffic between containers.
- *Configuration Management* – The orchestrator manages the configuration of containers and can automatically apply changes to running containers.

A container orchestrator typically consists of several components that work together. Some of the key components of a container orchestrator are:

- *Container Runtime* – Responsible for running and managing containers on each node in the cluster.
- *API Server* – Provides a central interface for interacting with the orchestrator. It accepts API requests from clients and communicates with the other components of the orchestrator to perform the requested actions.
- *Scheduler* – Responsible for deciding which node in the cluster to deploy a new container to, based on factors such as resource availability and workload balancing.

- *Controller Manager* – Responsible for managing various controllers that automate different aspects of the containerized application lifecycle, such as scaling, replication, and health monitoring.
- *Distributed Key-Value Store* – Used by the orchestrator to store configuration data, service discovery information, and other metadata.
- *Networking* – A component that provides the necessary network infrastructure to allow containers to communicate with each other across the cluster, including routing and load balancing.
- *Storage* – A component that manages the persistent storage needs of containers, including providing access to shared storage resources and ensuring data integrity.

The basic steps involved in container orchestration are:

1. *Create a Container Image* – Developers create a container image that includes their application code and all of its dependencies.
2. *Push the Image to a Container Registry* – The container image is pushed to a container registry, which is a central remote repository of container images.
3. *Define the Application Deployment* – Using a container orchestrator, developers define how the containerized application should be deployed, including the number of replicas, the network configuration, and any storage requirements.
4. *Deploy the Application* – The container orchestrator deploys the application across multiple nodes in a cluster, ensuring that the desired number of replicas are running and that the application is accessible to users.
5. *Monitor and Manage the Application* – The container orchestrator monitors the health of the application, automatically scaling it up or down as needed, and rolling out updates and patches without causing downtime. It also provides logging and monitoring capabilities to identify and troubleshoot any issues that arise.
6. *Manage Multiple Applications* – The container orchestrator can manage multiple containerized applications simultaneously, ensuring that they are deployed, scaled, and managed according to their individual requirements.

Container Package Manager vs. Container Orchestrator

A container package manager and a container orchestrator serve different functions. The following are the key differences:

- *Function* – A container package manager is responsible for managing container images and their dependencies, whereas a container orchestrator is responsible for automating the deployment, scaling, and management of containerized applications in a distributed computing environment.
- *Scope* – Container package managers focus specifically on managing container images and their dependencies, whereas container orchestrators manage the entire containerized application, from deployment to scaling to management.
- *Level of Abstraction* – Container package managers operate at a lower level of abstraction than container orchestrators. Package managers deal with individual container images and their dependencies, whereas orchestrators provide a high-level view of the entire containerized application.
- *Toolset* – Container package managers typically provide a more limited set of tools focused on managing container images and their dependencies. Container orchestrators, on the other hand, provide a range of tools and APIs for managing containers, networks, storage, and other infrastructure resources.

Container Networks

Containerization platforms generally provide virtual *container networks* to enable communication among containers that need to connect with each other. A container network is required to enable various containerization platform and system capabilities in support of providing:

- container availability
- container scalability
- container resiliency

The container network typically exists as a virtual network (Figure 6.37) that can be independently managed, configured, and encrypted.

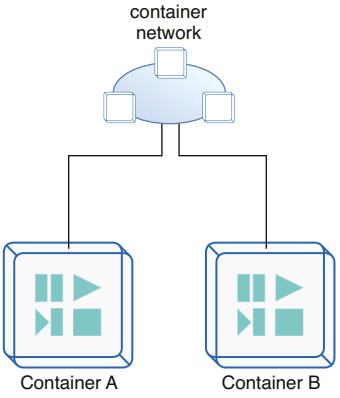


Figure 6.37
A container network allows containers to communicate with each other independently from the communication among the software programs they host.

As previously described in the *Fundamental Virtualization and Containerization* section, there are two primary types of container networks:

- Host Network
- Overlay Network

Whereas the host network is managed by a single container engine to support communication among containers on the same host, the overlay network enables container engines deployed on different servers to enable communication between containers on different hosts.

For example, if a distributed solution encompasses two services, each in its own container, then a container network is established for the two containers that are part of that solution. If the containers are on the same host, then a host network is created (Figure 6.38). If one container is on one host and the other is on a different host, then an overlay network is created (Figure 6.39).

Container Network Scope

The scope of a container network is usually equal to the scope of a given solution. This is because the scope of a solution will encompass only those containers hosting software programs that are part of that solution. Therefore, when multiple solutions are hosted, multiple container networks will be required.

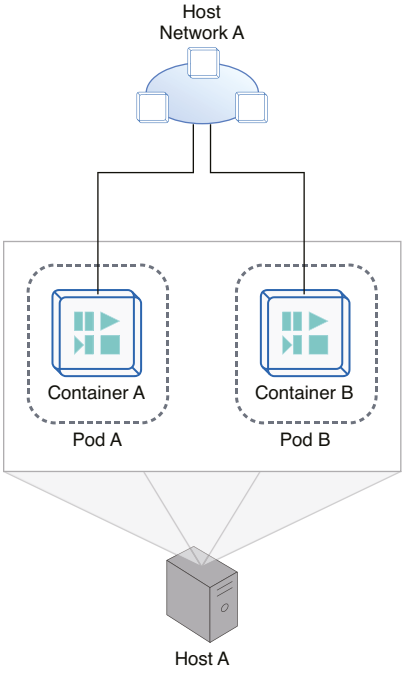


Figure 6.38

Containers A and B reside in separate pods on the same host and can communicate with each other via Host Network A.

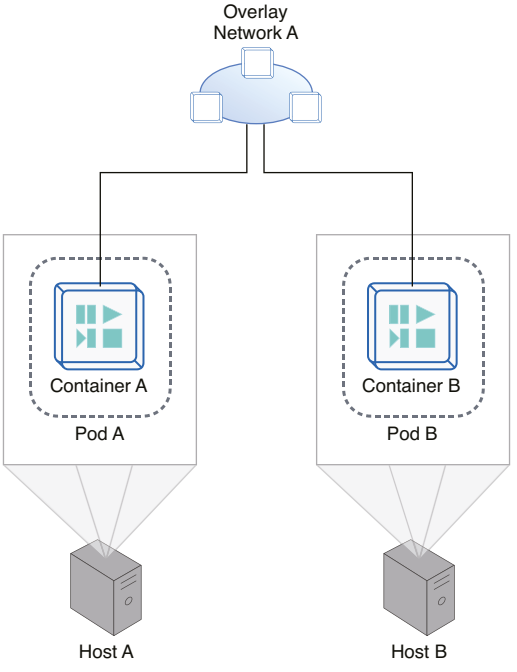


Figure 6.39

Containers A and B reside on different hosts and can communicate with each other via Overlay Network A.

Some solutions share software programs, such as reusable utility services or a shared database. If the reusable software program is in a container, then that container can participate in multiple container networks (Figure 6.40).

NOTE

The container network(s) that a given container will join can be specified by the administrator in the container image's build file and also in the container deployment package. If no network is specified, the container engine may automatically assign a container to a "default" host network. Build files are covered later in this chapter in the *Container Build Files* subsection of the upcoming *Understanding Container Images* section.

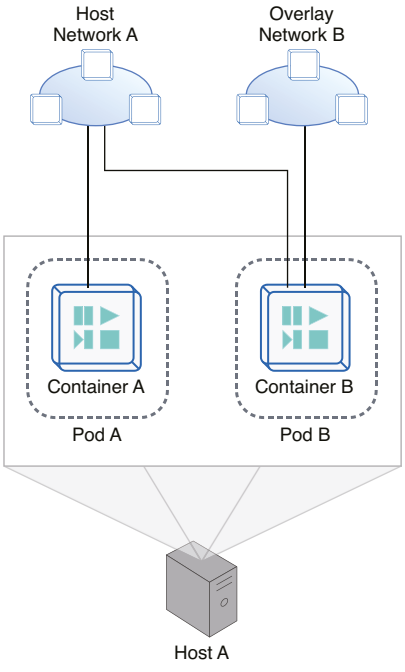


Figure 6.40
Containers A and B reside on the same host and can communicate via Host Network A. Container B is further part of Overlay Network B, through which it can communicate with other containers.

Usually, container networks by default limit communication of the containerized solution software programs so that they can only communicate with each other. However, a solution may need to be able to access software programs or IT resources that are not containerized and therefore reside outside of the container network. In this case, the container network needs to be configured to allow the solution to communicate outside of the container network boundary (Figure 6.41).

Container Network Addresses

Each deployed container receives a *network address* that enables it to participate in a container network. A network address usually exists as an IP address. If a container needs to participate in multiple container networks, it will require a separate network address for each container network. For example, if a container hosting a software program is being reused across two container networks (such as in the example shown in Figure 6.40), then that container will need two network addresses.

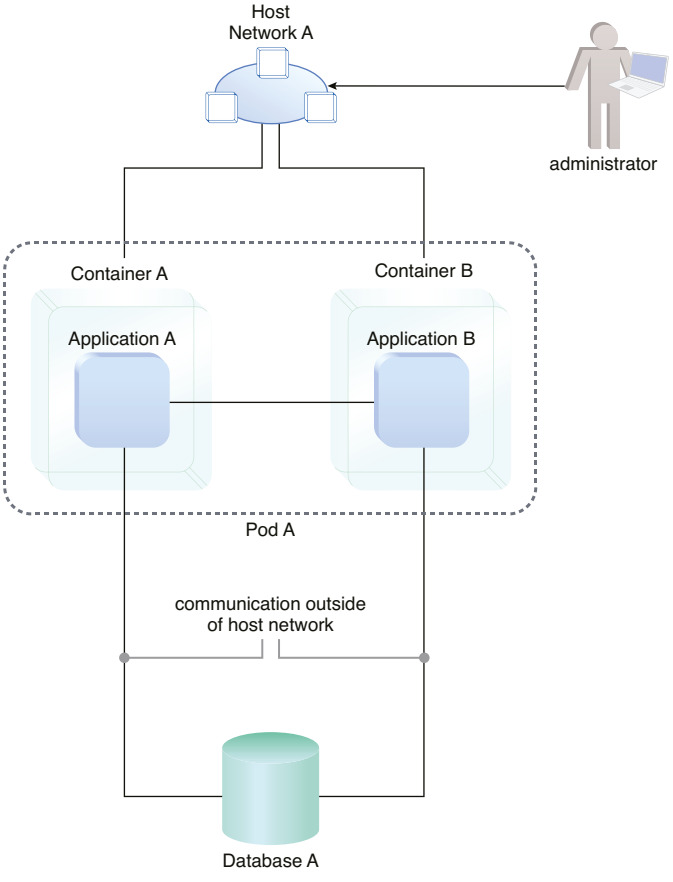


Figure 6.41

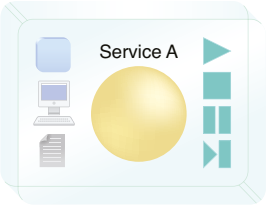
Containers A and B communicate with each other via Host Network A. Containerized Applications A and B need to communicate with each other, as well as with Database A, which resides outside of Host Network A. The administrator enables this by explicitly configuring Host Network A to allow for the required external access.

Network addresses are usually assigned by the container engine subsequent to container deployment. They can also be manually assigned by the administrator in a deployment package. Containers residing in the same pod share the same network address and are individually identified through different network ports.

Rich Containers

Different types of containerization platforms can vary in the range of features supported by a given container. Containers that are more feature-rich are referred to as *rich containers* (Figure 6.42).

Figure 6.42
Service A is deployed inside a rich container that provides extra features, including monitoring capabilities that can provide ongoing status and health information about the service.



The extent to which a container can be feature-rich is determined by the capabilities of the underlying container engine responsible for creating the container.

Examples of features provided by more advanced container engines include:

- Container resources can be limited to control and govern the maximum number of resources that a container can consume.
- Usage logs can be collected for auditing and regulatory purposes.
- Container restart criteria can be specified. For example, the container can be configured to automatically restart if a certain event or error occurs, but not if other types of events or errors occur.
- Container storage can be managed. This includes enabling an isolated file system to be shared by multiple containerized services.
- Sharing storage between the host and the containers running on the host. This may be required for regulatory and auditing purposes or to ensure that if the container is turned off, access to the data is still available.
- Support for the execution of service composition logic is available. This can be used for services deployed inside a container hosted together on the same host.

Some container engines further provide proxy features that allow them to act as a proxy for consumer requests to services they are hosting.

Other Common Container Characteristics

Provided here are some further common container characteristics:

- With each program hosted in a container, numerous supporting programs (such as databases, utility programs, and monitors) can also be deployed.
- Containers can be configured so that the amount of infrastructure resources each consumes is limited.
- The visibility of external programs and IT resources accessible to a container and its hosted programs can be limited.
- The programs hosted by a container normally share the same lifecycle as the container itself. This means that a hosted program will typically start, stop, pause, and resume in sync with the container.

6.4 Understanding Container Images

Container images are a central part of containerization platforms. They form the basis of ongoing container creation. The processing of container images is one of the primary responsibilities of the container engine.

Container Image Types and Roles

How container images are used, stored, and processed depends on what type they are or what role they assume.

There are two primary types of container images:

- *Base Container Images* – These container images act as templates for customized container images. In this book, this type of container image is qualified as a "base" container image. Base container images are also referred to as partial container images.
- *Customized Container Images* – These container images are created by the container engine, which then uses them to create actual, deployed containers. In this book, this type of container image may or may not be qualified as a "customized" container image. When a symbol is only labeled as a "container image," it is implied that it has been customized.

The reason that these types of container images can be considered roles is that a customized container image created from a base container image can itself become a base container image to be used as a template for future, different customized container images.

As illustrated in Figure 6.43, a container image classified as a base container image is published to the image registry, from where it can then be accessed by the container engine to form the basis of customized container images (as explained in further detail later in this section).

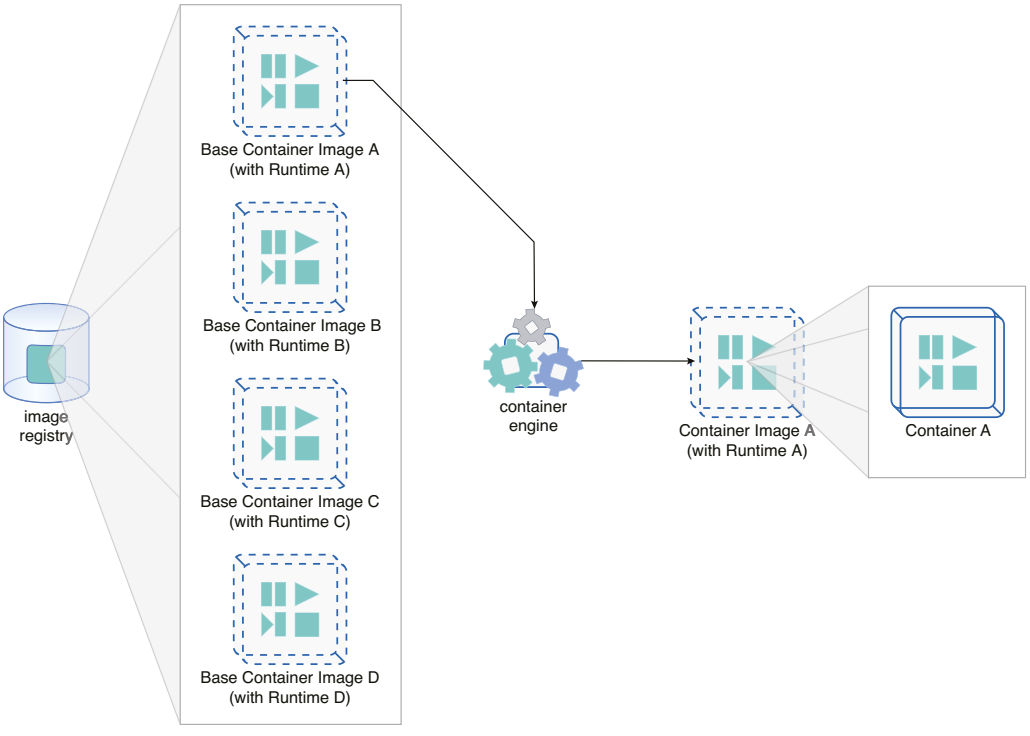


Figure 6.43

The container engine supports four different runtimes that can be deployed in containers. Application A requires the capabilities of Runtime A. Base Container Image A is used by the container engine to create the customized Container Image A that is then used to create and deploy the actual Container A for Application A.

Container Image Immutability

A key characteristic of container images is that, once created, they are *immutable*. This means that they cannot be altered (neither patched nor updated, nor any other type of alteration made). If a change to a container image is required, then a new or revised build file needs to be created and a new version of the container image needs to be generated, further resulting in the need for a new version of the deployed container.

The scope of a container's immutability relates to the contents of the build file. Administrative tools allow for settings on a container to be changed. These changes do not relate to a build file and can therefore be made without the need to create a new version of the container.

The container engine assigns each individual container image a unique auto-generated image key that is further kept where the container image is stored, either in the image registry or in the container engine's internal storage.

Container Image Abstraction

A base container image will typically provide a subset of the functions offered by the underlying host operating system. This is referred to as *operating system abstraction*, or simply *abstraction*. However, not all parts of the operating system are abstracted by the container image, as further explained in the next two subsections.

Operating System Kernel Abstraction

Each operating system has a *kernel*, which exists as a set of the most essential operating system functions. Kernels in different operating systems are comprised of very similar functions. For example, the kernel of a Windows operating system has similar functions to the kernel of a Linux operating system.

Common functions provided by a kernel can include:

- access to CPU resources
- access to processing
- access to host memory
- access to input/output devices in the host
- access to hardware storage
- access to device drivers

- access to server file systems
- access to power management

The kernel is *not* abstracted by the container image. Instead, it is encompassed by the container engine. As a result, container images do not need to copy the kernel, which helps to further reduce their footprint.

The container engine acts as somewhat of a liaison or intermediary, enabling containers to have full access to the entire set of kernel functions at runtime. Container engines are generally able to interact with kernels from different operating systems, which helps enable the portability of containers and their platforms across different hosting environments.

Operating System Abstraction Beyond the Kernel

The parts of an operating system outside of the kernel *can* be abstracted and included in a container image.

Common operating system functions and resources that exist outside of the kernel can include:

- programming language libraries and compilers
- various system libraries
- encryption platforms
- system monitors and monitoring functions
- configuration files and editors
- administrative functions and platforms
- administrative tools (for use by human administrators)
- localization programs

This form of abstraction represents the subset of the operating system that a given container image can capture to provide a customized and optimized hosting environment for the software programs that will be deployed in the container generated from that container image.

When abstracting these types of functions and resources, the container remains portable because the abstracted functions and resources are copied to and ported with the container.

Container Build Files

Figure 6.44
The symbol used to represent a container build file.

A *container build file* (or just the *build file*) (Figure 6.44) is a human-editable, machine-processable configuration file that specifies what belongs in (or what is abstracted by) a customized container image.

Specifically, the build file can identify:

- the base container image that will be used to form the basis of the customized container image
- the additional operating system resources to be added to (or abstracted by) the customized container image
- the container network(s) in which the deployed customized container will need to participate

The syntax and format in a given build file can vary, depending on the type of container engine being used.

Container Image Layers

A container image organizes its content into *layers*. Each layer corresponds to a container build file statement or instruction.

Examples of content in container image layers include:

- data files and folders
- configuration files
- databases and repositories
- executable files
- operating system program files and runtimes

Except for the very final layer, all the layers are read-only. The containerization platform uses a *union file system* as the basis of container image layering. The use of the union file system and layering is what makes the reusability of base container images possible.

A base container image is comprised of a number of layers that represent what it abstracts (Figure 6.45).

The customized container image that is derived from the base container image will add layers to what is provided by the base container image. In the customized container image, the entire base container image represents the bottom layer (Figure 6.46).

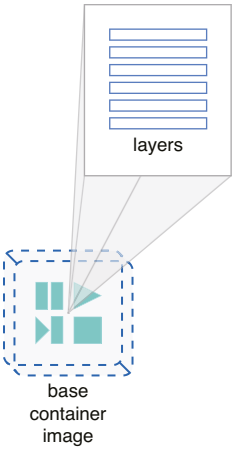


Figure 6.45
The base container image has its own set of layers.

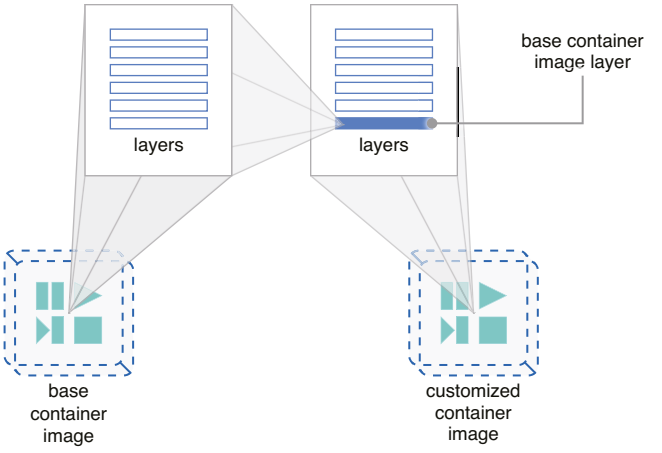
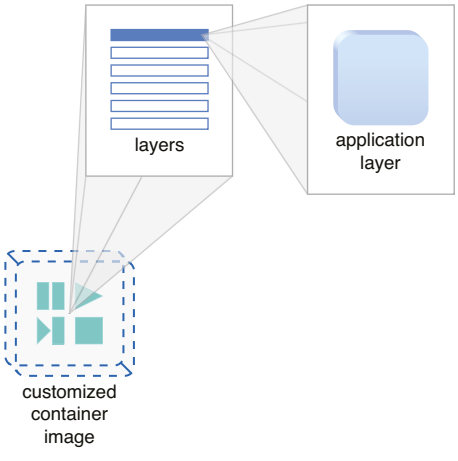


Figure 6.46
The bottom layer of the customized container image is comprised of the contents of the base container image.

A software program that will be hosted by the deployed container that will be generated from the customized container image can itself reside in a layer of the customized container image (Figure 6.47).

Figure 6.47
A layer within the customized container image is comprised of the software program that the deployed container will be responsible for hosting.



Because container images are immutable, if a layer within an image needs to be removed or added to, a new container image version needs to be created.

How Customized Container Images Are Created

The container engine uses the build file together with the base container image to generate the customized container image (Figure 6.48).

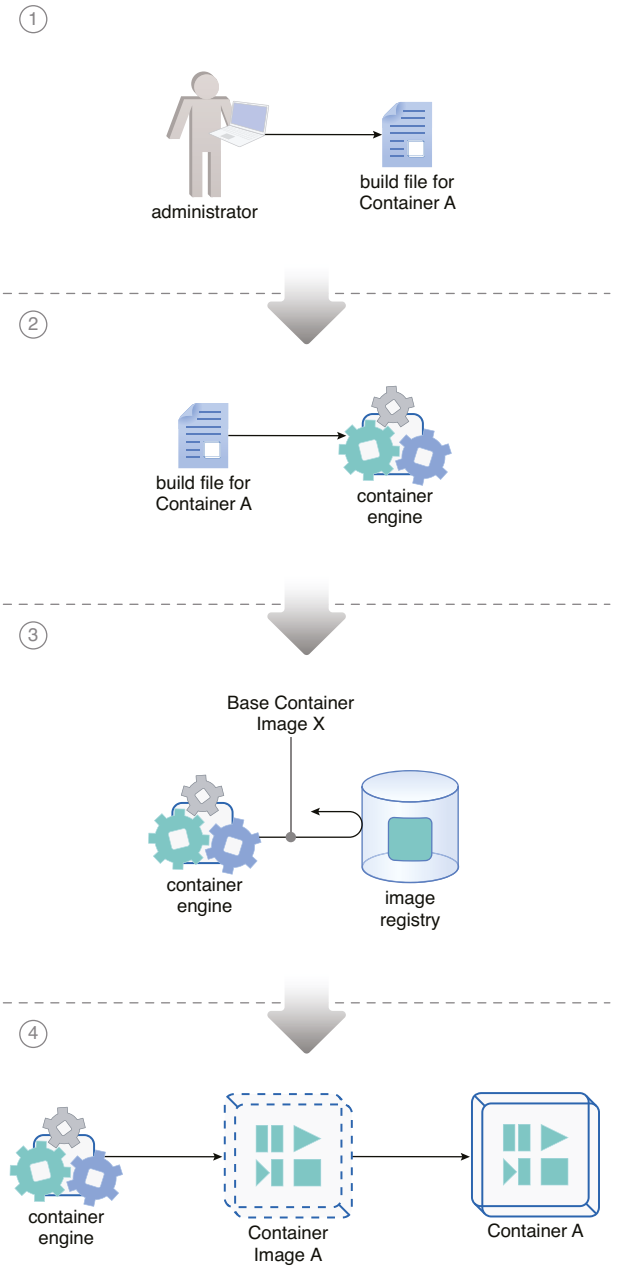


Figure 6.48
The administrator authors a build file for Container A (1). The administrator provides the build file to the container engine (2). The container engine retrieves the required base container image from the image registry (3). The container engine then uses the base container image and the information from the build file to create a new customized Container Image A, from which it then generates and deploys Container A (4).

Once the actual container implementation is created from the customized container image and deployed, there may no longer be a need to keep the build file because the container engine now has the customized container image that it can use to create more instances of the actual container in the future.

Note that the customized container image is not typically stored in the image registry. Instead, it is stored in the container engine's internal storage so that the engine can retain immediate access to it in support of efficiently and rapidly creating new container instances for scalability and resiliency purposes.

A customized container image can also be published to the image registry if the administrator determines that it may be useful as a base container image to be used for new types of customized container images.

6.5 Multi-Container Types

So far, the majority of containers shown have been hosting applications and services that, presumably, are responsible for processing primary business logic. However, for applications to function in distributed environments, additional types of secondary (or utility) processing are also required.

This section introduces the following set of basic *multi-container* types, each of which adds a container with a secondary component that abstracts utility-related processing:

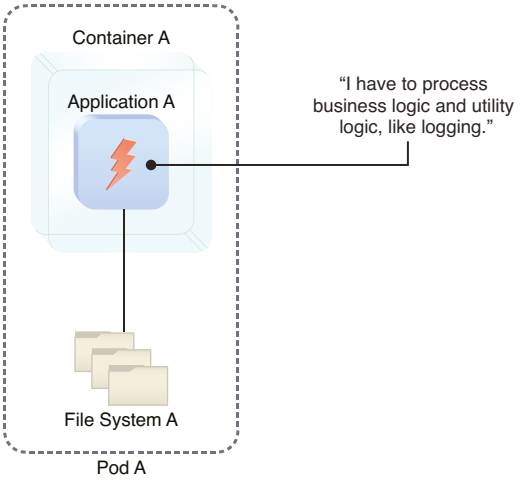
- Sidecar Container
- Adapter Container
- Ambassador Container

Sidecar Container

When an application responsible for processing primary business logic is also built to process generic utility logic, the ability for the application to process its business logic reliably and effectively can be compromised (Figure 6.49).

A secondary containerized application component (referred to as a *sidecar component*) is added to abstract the utility logic–related processing (Figure 6.50). The sidecar component is deployed in a separate container, usually within the same pod as the application. Depending on the nature of the utility processing, the application may or may not need to communicate with the sidecar component.

Figure 6.49
Application A is burdened with carrying out business logic and utility logic.



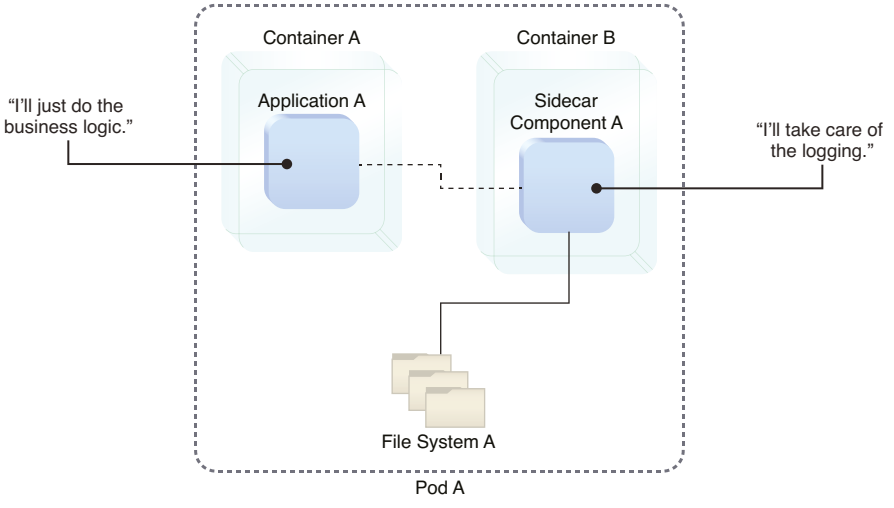


Figure 6.50
The utility logic is placed in Sidecar Component A that resides in the separate Container B, in the same pod. This enables Application A to focus exclusively on carrying out its business logic.

Adapter Container

When an application responsible for processing primary business logic is also built to carry out data conversion logic to accommodate external consumer applications, the ability for the application to process its business logic reliably and effectively can be compromised (Figure 6.51). Furthermore, by embedding this conversion logic into the application, it may become coupled to multiple different external consumer programs, which can become burdensome when those consumer programs change over time.

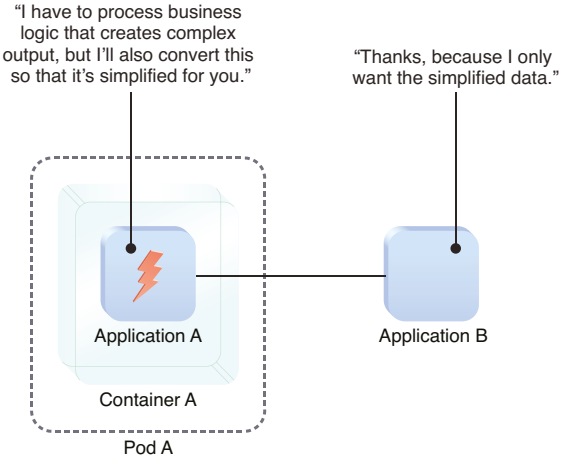


Figure 6.51
Application A is burdened with carrying out business logic and specific conversion logic required by Application B.

A secondary containerized application component (referred to as an *adapter component*) is added to abstract any necessary data conversion processing logic (Figure 6.52). The adapter component is deployed in a separate container, usually within the same pod as the application. A separate adapter component can be deployed for each consumer application that requires a different representation of the output data.

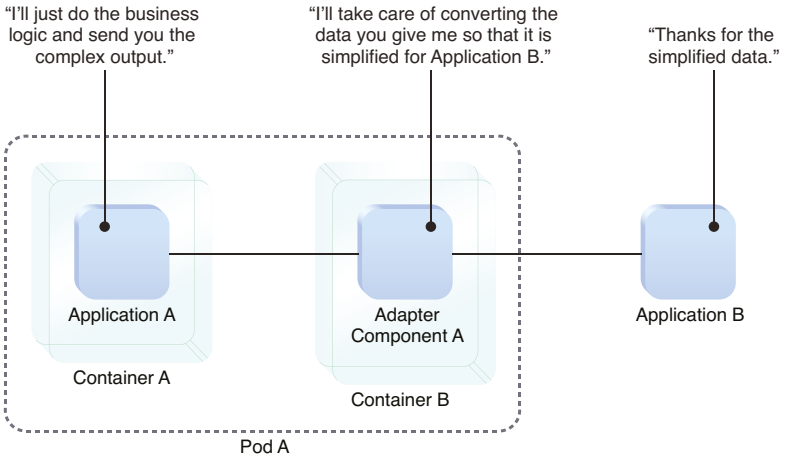


Figure 6.52

The conversion logic is placed in Adapter Component A that resides in the separate Container B, in the same pod. This enables Application A to focus exclusively on carrying out its business logic.

Ambassador Container

When an application responsible for processing primary business logic is also built to carry out external communication processing logic to connect with external consumer applications, the ability for the application to process its business logic reliably and effectively can be compromised (Figure 6.53). Also, by embedding this specific communication logic (such as logic related to protocols, messaging, and security) into the application, it may become coupled to multiple different external programs, which can become burdensome when the APIs of those programs change over time.

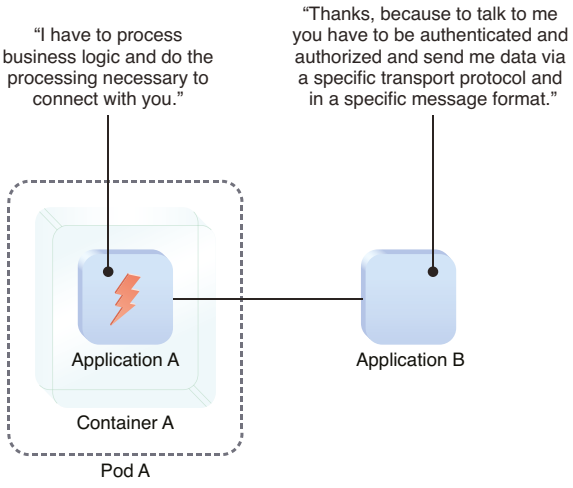


Figure 6.53
Application A is burdened with carrying out business logic and specific communications processing logic required to connect with Application B.

A secondary containerized application component (referred to as an *ambassador component*) is added to abstract any necessary communication processing logic (Figure 6.54). The ambassador component is deployed in a separate container, usually within the same pod as the application. A separate ambassador component can be deployed for each application that has a set of different communication requirements.

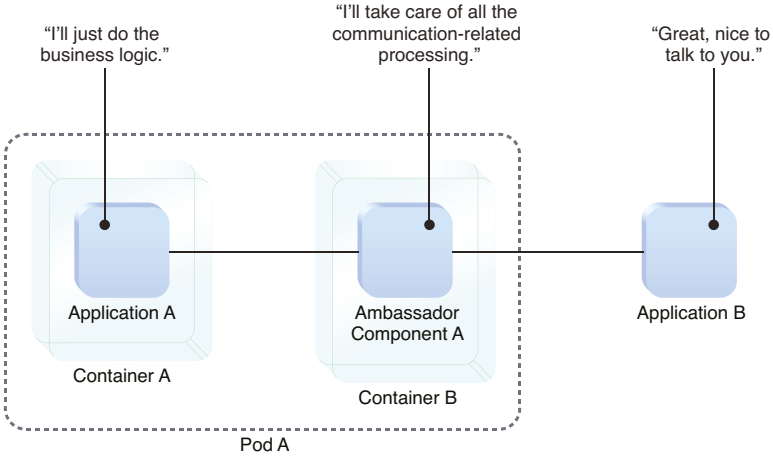


Figure 6.54
The communications logic is placed in Ambassador Component A that resides in the separate Container B, in the same pod. This enables Application A to focus exclusively on carrying out its business logic.

Using Multi-Containers Together

The three types of multi-containers can be used individually or together, as required. For example, depending on the nature of this business logic, an application may require some or all of the secondary containers to be deployed with it (Figure 6.55).

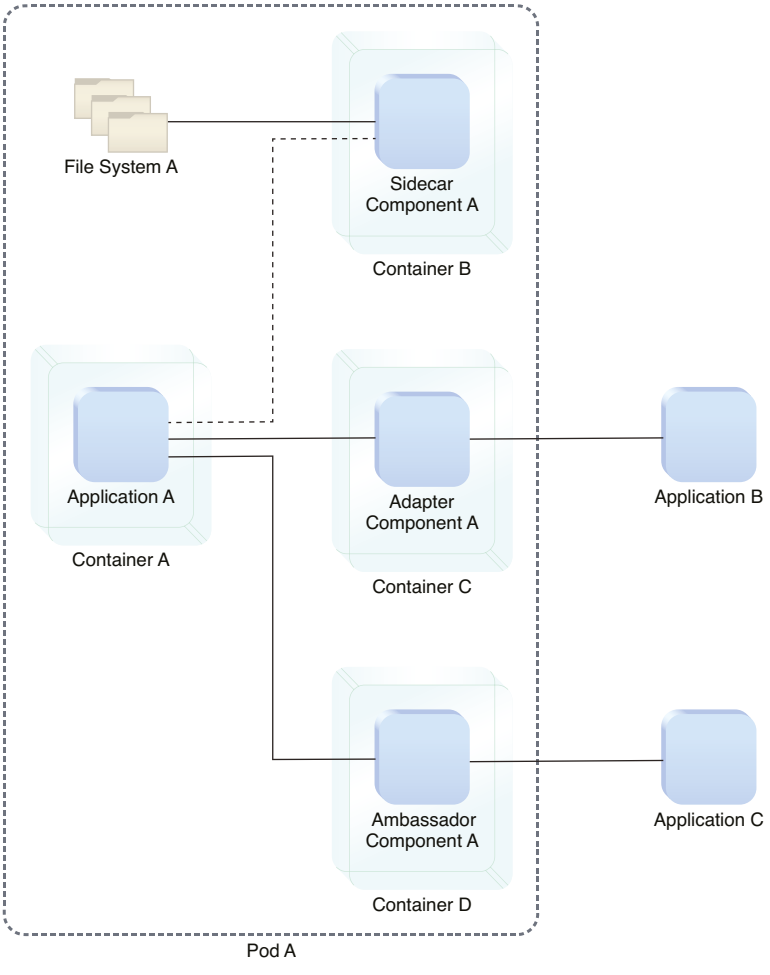


Figure 6.55
Application A is supported by three secondary containers.

6.6 CASE STUDY EXAMPLE

Innovartus Technologies Inc. has identified multiple benefits from using containerization technology in support of its technology and business strategies, including the following:

- Scalability can be greatly improved to accommodate increased and less predictable cloud consumer interaction.
- Service levels can also be improved to avoid the frequent outages that are currently occurring more frequently than usual.
- Cost-effectiveness can be improved by reducing the number of virtual servers required for the delivery of its virtual products, as these products can now be deployed in containers instead of virtual servers.

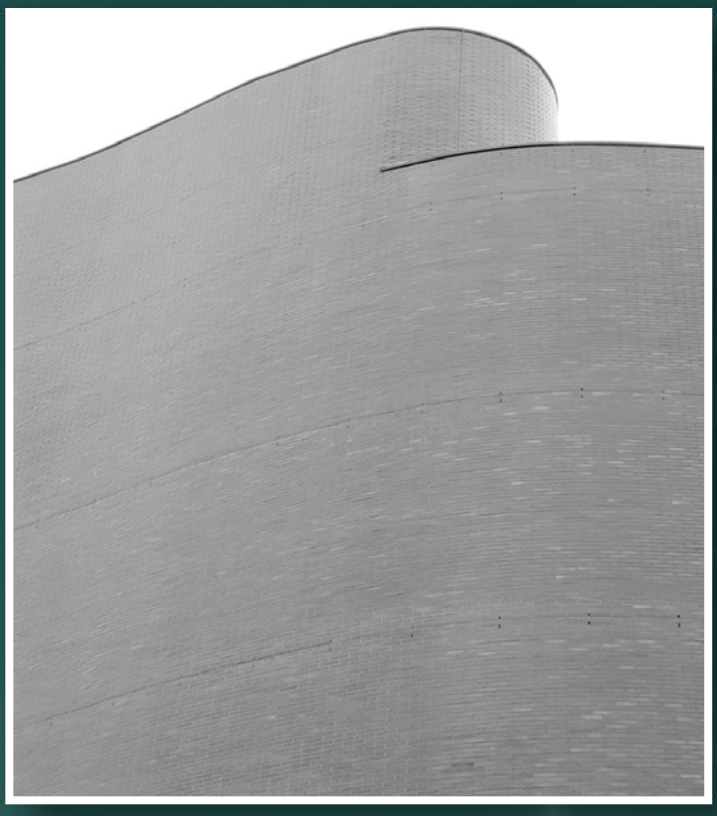
The virtual toys and educational entertainment products for children offered by Innovartus were designed as applications comprised of multiple independent services that work together to provide the necessary functionality. This allows for every individual service to be deployed in its individual container and scaled out dynamically in accordance with its performance and total capacity requirements.

Due to certain security-related requirements, three services that provide access to parents for the configuration of their child's virtual toys need to share the same IP address. Deploying them in separate containers within a single logical pod provides the ideal deployment solution for this requirement.

Monitoring the usage, performance, and security of the virtual toys and entertainment products is fundamental to its business strategy. However, to allow each service running in its own container to focus on the functionality it must deliver as part of a virtual toy or other entertainment product, sidecar containers can be used to separate utility-related functionality, such as writing logs or reporting data to performance and security monitoring logic, in components running in the sidecar container.

Two of the monitoring systems that the services need to send telemetry data to are deployed remotely. In this case, ambassador containers are used to allow services to delegate the communication with the remote system to the ambassador and focus on the core functionality they were designed to deliver.

Finally, adapter containers are used throughout the Innovartus architecture to allow the use of their products by users via different devices (such as smartphones, tablets, and computers) with the adapter containers running the logic necessary for every device to be accessed separately by both parents and children.



Chapter 7

Understanding Cloud Security and Cybersecurity

This chapter introduces terms and concepts that address basic information security within clouds, and then concludes by defining a set of threats and attacks common to public cloud environments. The cloud security and cybersecurity mechanisms covered in Chapters 10 and 11 establish the security controls used to counter these threats.

7.1 Basic Security Terminology

Information security is a complex ensemble of techniques, technologies, regulations, and behaviors that collaboratively protect the integrity of and access to computer systems and data. IT security measures aim to defend against threats and interference that arise from both malicious intent and unintentional user error.

The upcoming sections define fundamental security terms relevant to cloud computing and describe associated concepts.

Confidentiality

Confidentiality is the characteristic of something being made accessible only to authorized parties (Figure 7.1). Within cloud environments, confidentiality primarily pertains to restricting access to data in transit and storage.

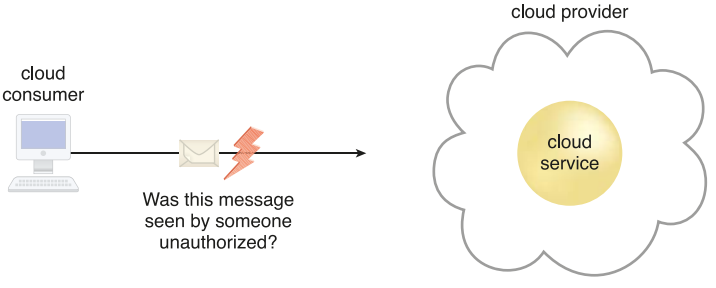


Figure 7.1
The message issued by the cloud consumer to the cloud service is considered confidential only if it is not accessed or read by an unauthorized party.

Integrity

Integrity is the characteristic of not having been altered by an unauthorized party (Figure 7.2). An important issue that concerns data integrity in the cloud is whether a cloud consumer can be guaranteed that the data it transmits to a cloud service matches the data received by that cloud service. Integrity can extend to how data is stored, processed, and retrieved by cloud services and cloud-based IT resources.

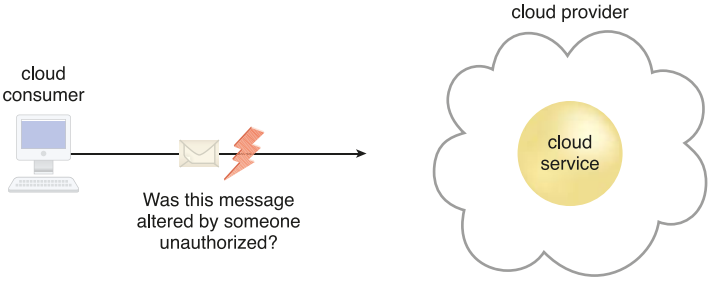


Figure 7.2
The message issued by the cloud consumer to the cloud service is considered to have integrity if it has not been altered.

Availability

Availability is the characteristic of being accessible and usable during a specified time period. In typical cloud environments, the availability of cloud services can be a responsibility that is shared by the cloud provider and the cloud carrier. The availability of a cloud-based solution that extends to cloud service consumers is further shared by the cloud consumer.

Figure 7.3 depicts a scenario that demonstrates how a collection of security technologies helps ensure the confidentiality and integrity of data exchange over the internet, as well as the availability of a central database containing private data.

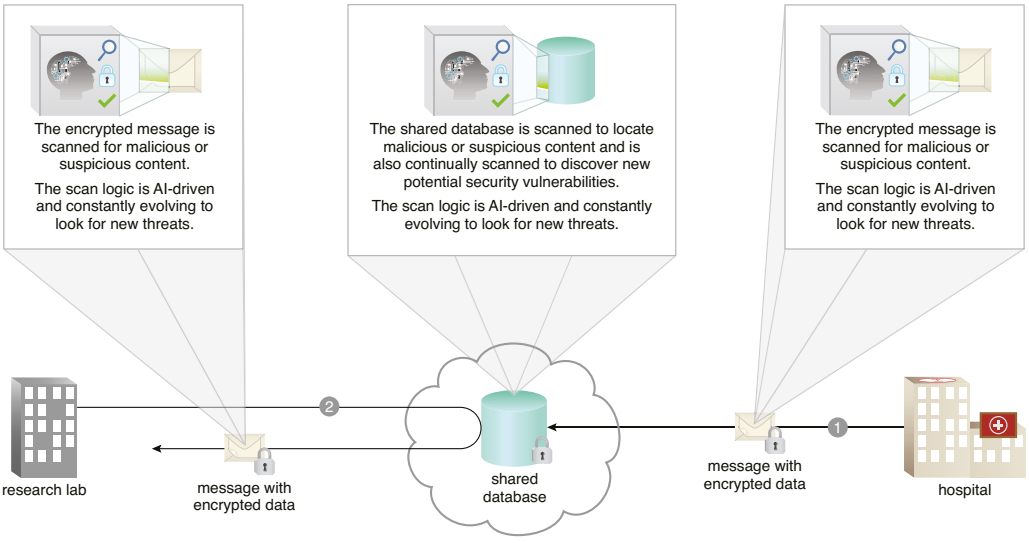


Figure 7.3

A hospital contributes confidential medical data to a database in a cloud (1) shared by a research institution that retrieves the data (2). Supporting cybersecurity technologies provide confidentiality via encryption, integrity via runtime scanning, and availability by ensuring the ongoing safety of the shared cloud-based database.

Authenticity

Authenticity is the characteristic of something having been provided by an authorized source. This concept encompasses non-repudiation, which is the inability of a party to deny or challenge the authentication of an interaction. Authentication in non-repudiable interactions provides proof that these interactions are uniquely linked to an authorized source. For example, a user may not be able to access a non-repudiable file after its receipt without also generating a record of this access.

Security Controls

Security controls are countermeasures used to prevent or respond to security threats and to reduce or avoid risk. Details on how to use security countermeasures are typically outlined in the security policy, which contains a set of rules and practices specifying how to implement a system, service, or security plan for maximum protection of sensitive and critical IT resources.

Security Mechanisms

Countermeasures are typically described in terms of *security mechanisms*, which are components comprising a defensive framework that protects IT resources, information, and services. Chapters 10 and 11 describe a series of cloud security and cybersecurity mechanisms.

Security Policies

A *security policy* establishes a set of security rules and regulations. Often, security policies will further define how these rules and regulations are implemented and enforced. For example, the positioning and usage of security controls and mechanisms can be determined by security policies.

7.2 Basic Threat Terminology

This section covers some fundamental topics that help establish the primary purpose and scope of cybersecurity practices and technologies, as well as some essential vocabulary.

Risk

Risk is the potential unwanted and unexpected loss that may result from a given action. Risks can pertain to various aspects of cybersecurity, including external threats, internal vulnerabilities, and responses carried out against threats, as well as risks associated with possible human error, technology malfunctions, and the overall quality of a cybersecurity environment.

Vulnerability

A *vulnerability*, within the context of cybersecurity, is a flaw, gap, or weakness in an IT environment or associated policies or processes that leaves an organization open to potentially successful security breaches. Vulnerabilities can be physical or digital. Attackers attempt to exploit vulnerabilities, while organizations attempt to eliminate or mitigate them.

Exploit

An *exploit* occurs when an attacker is able to take advantage of a vulnerability.

Zero-Day Vulnerability

A *zero-day vulnerability* is a vulnerability that an organization is either unaware of or for which it has not yet been able to provide a patch or fix. As a result, an attacker may be able to more easily exploit this vulnerability until the organization is able to address it.

Security Breach

A *security breach* is any incident that may result in unauthorized access to information or systems. It typically occurs when an attacker is able to bypass security mechanisms and controls.

Data Breach

A *data breach* is a type of security breach whereby an attacker is able to steal confidential information.

Data Leak

A *data leak* occurs when sensitive information is shared with unauthorized parties without an attack taking place. Data leaks can occur accidentally or intentionally and are usually carried out by humans.

Threat (or Cyber Threat)

A *threat* or a *cyber threat* is a known, potential attack that poses a danger and risk to an organization. The collection of threats relevant to a given organization is known as the *threat landscape* or *cyber threat landscape*.

Attack (or Cyber Attack)

When a threat is carried out by an attacker, it becomes an *attack* or a *cyber attack.*

Attacker and Intruder

Within the context of cloud security and cybersecurity, an *attacker* is an individual or organization that carries out cyber attacks.

There are different types of attackers:

- *Cyber Criminals* – Attackers that attempt to steal private information for the purpose of profit or other types of illegal activity.
- *Malicious Users* – Authorized users, such as rogue employees, who abuse trusted privileges to access a system with the intent to cause harm or carry out unauthorized actions.
- *Cyber Activists* – Attackers that carry out malicious activity to promote a political agenda, religious belief, or social ideology.
- *State-Sponsored Attackers* – Attackers who are hired by a government agency.

Any attacker that has successfully gained unauthorized access within an organizational boundary is known as an *intruder*.

Attack Vector and Surface

An *attack vector* is the path that an attacker takes to exploit vulnerabilities. Examples of attack vectors are email attachments, pop-up windows, chat rooms, and instant messages. Human error or ignorance is commonly planned for when creating a given attack vector. An *attack surface* is a collection of attack vectors from which an attacker can access a system or extract information.

7.3 Threat Agents

A *threat agent* is an entity that poses a threat because it is capable of carrying out an attack. Cloud security threats can originate either internally or externally, from humans or software programs. Corresponding threat agents are described in the upcoming sections. Figure 7.4 illustrates the role a threat agent assumes in relation to vulnerabilities, threats, and risks, and the safeguards established by security policies and security mechanisms.

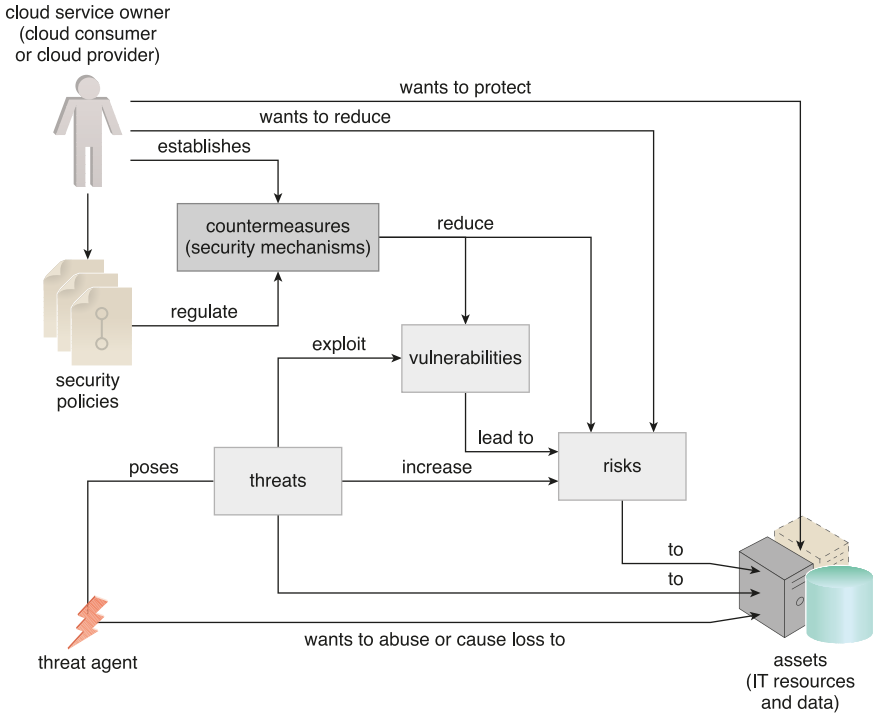


Figure 7.4

How security policies and security mechanisms are used to counter threats, vulnerabilities, and risks caused by threat agents.

Anonymous Attacker

An *anonymous attacker* is a non-trusted cloud service consumer without permissions in the cloud (Figure 7.5). It typically exists as an external software program that launches network-level attacks through public networks. When anonymous attackers have limited information about security policies and defenses, it can inhibit their ability to formulate effective attacks. Therefore, anonymous attackers often resort to committing acts such as bypassing user accounts or stealing user credentials, while using methods that either ensure anonymity or require substantial resources for prosecution.

Figure 7.5

The notation used for an anonymous attacker.

Malicious Service Agent

A *malicious service agent* is able to intercept and forward the network traffic that flows within a cloud (Figure 7.6). It typically exists as a service agent (or a program pretending to be a service agent) with compromised or malicious logic. It may also exist as an external program able to remotely intercept and potentially corrupt message contents.

Figure 7.6
The notation used for a malicious service agent.

Trusted Attacker

A *trusted attacker* shares IT resources in the same cloud environment as the cloud consumer and attempts to exploit legitimate credentials to target cloud providers and the cloud tenants with whom they share IT resources (Figure 7.7). Unlike anonymous attackers (which are non-trusted), trusted attackers usually launch their attacks from within a cloud's trust boundaries by abusing legitimate credentials or via the appropriation of sensitive and confidential information.

Figure 7.7
The notation that is used for a trusted attacker.

Trusted attackers (also known as *malicious tenants*) can use cloud-based IT resources for a wide range of exploitations, including the hacking of weak authentication processes, the breaking of encryption, the spamming of email accounts, or to launch common attacks, such as denial of service campaigns.

Malicious Insider

Malicious insiders are human threat agents acting on behalf of or in relation to the cloud provider. They are typically current or former employees or third parties with access to the cloud provider's premises. This type of threat agent carries tremendous damage potential, as the malicious insider may have administrative privileges for accessing cloud consumer IT resources.

NOTE

A notation used to represent a general form of human-driven attack is the workstation combined with a lightning bolt (Figure 7.8). This generic symbol does not imply a specific threat agent, only that an attack was initiated via a workstation.

Figure 7.8
The notation used for an attack originating from a workstation. The human symbol is optional.

7.4 Common Threats

This section introduces several common threats and vulnerabilities in cloud-based environments and describes the roles of the aforementioned threat agents.

Traffic Eavesdropping

Traffic eavesdropping occurs when data being transferred to or within a cloud (usually from the cloud consumer to the cloud provider) is passively intercepted by a malicious service agent for illegitimate information-gathering purposes (Figure 7.9). The aim of this attack is to directly compromise the confidentiality of the data and, possibly, the confidentiality of the relationship between the cloud consumer and the cloud provider. Because of the passive nature of the attack, it can more easily go undetected for extended periods of time.

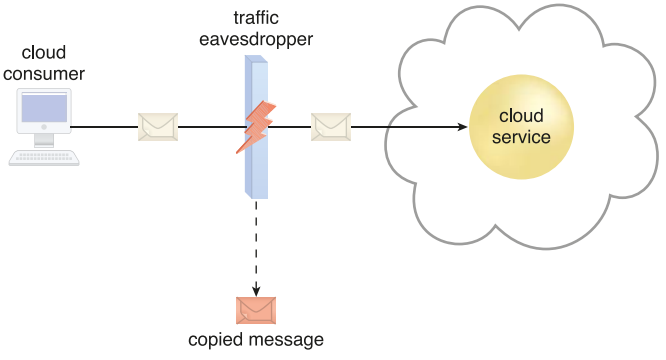


Figure 7.9
An externally positioned malicious service agent carries out a traffic eavesdropping attack by intercepting a message sent by the cloud service consumer to the cloud service. The service agent makes an unauthorized copy of the message before it is sent along its original path to the cloud service.

Malicious Intermediary

The *malicious intermediary* threat arises when messages are intercepted and altered by a malicious service agent, thereby potentially compromising the message's confidentiality and/or integrity. It may also insert harmful data into the message before forwarding it to its destination. Figure 7.10 illustrates a common example of the malicious intermediary attack.

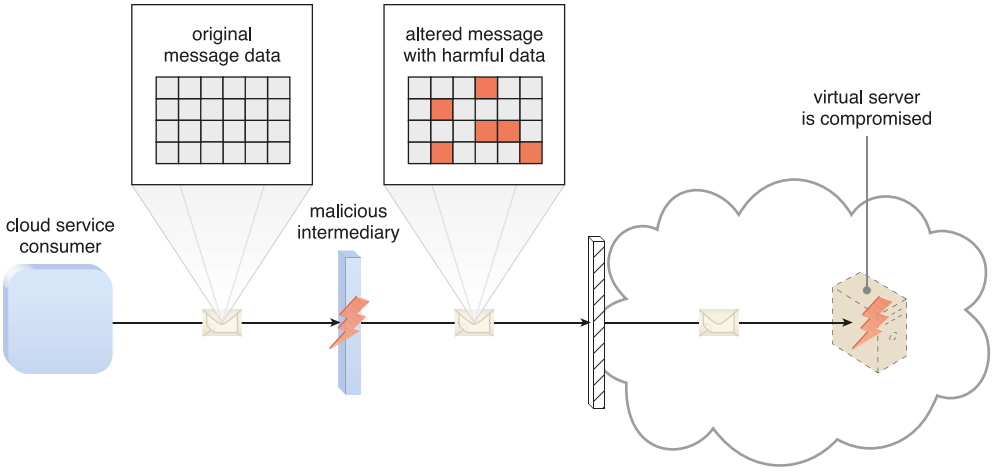


Figure 7.10

The malicious service agent intercepts and modifies a message sent by a cloud service consumer to a cloud service (not shown) being hosted on a virtual server. Because harmful data is packaged into the message, the virtual server is compromised.

NOTE

While not as common, the malicious intermediary attack can also be carried out by a malicious cloud service consumer program.

Denial of Service

The objective of the *denial of service (DoS)* attack is to overload IT resources to the point where they cannot function properly. This form of attack is commonly launched in one of the following ways:

- The workload on cloud services is artificially increased with imitation messages or repeated communication requests.
- The network is overloaded with traffic to reduce its responsiveness and cripple its performance.
- Multiple cloud service requests are sent, each of which is designed to consume excessive memory and processing resources.

Successful DoS attacks produce server degradation and/or failure, as illustrated in Figure 7.11.

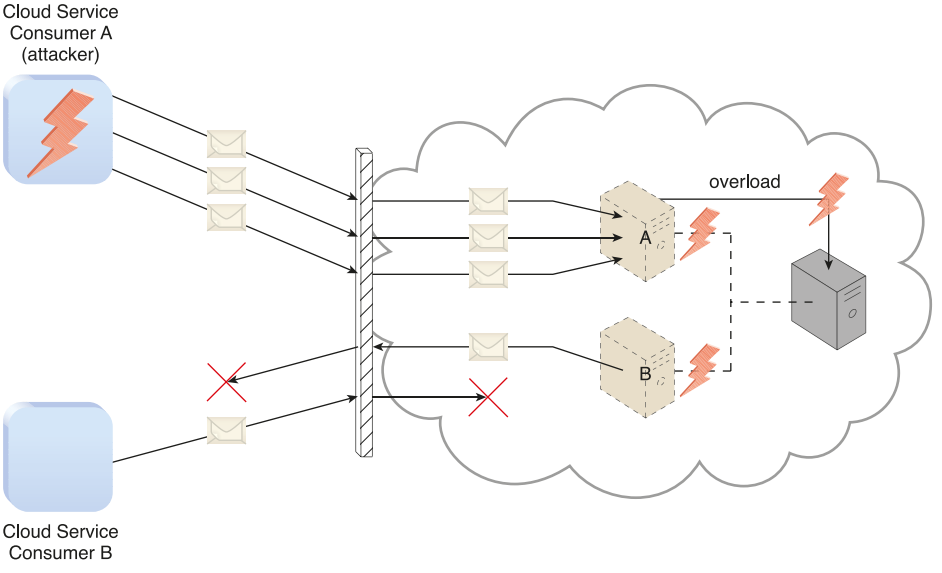


Figure 7.11

Cloud Service Consumer A sends multiple messages to a cloud service (not shown) hosted on Virtual Server A. This overloads the capacity of the underlying physical server, which causes outages with Virtual Servers A and B. As a result, legitimate cloud service consumers, such as Cloud Service Consumer B, become unable to communicate with any cloud services hosted on Virtual Servers A and B.

NOTE

A common variation of the DoS attack is the *DDoS (distributed denial of service)* attack, in which multiple compromised systems are used to flood a targeted website or network with traffic in an attempt to make it unavailable.

Insufficient Authorization

The *insufficient authorization* attack occurs when access is granted to an attacker erroneously or too broadly, resulting in the attacker getting access to IT resources that are normally protected. This is often a result of the attacker gaining direct access to IT resources that were implemented under the assumption that they would only be accessed by trusted consumer programs (Figure 7.12).

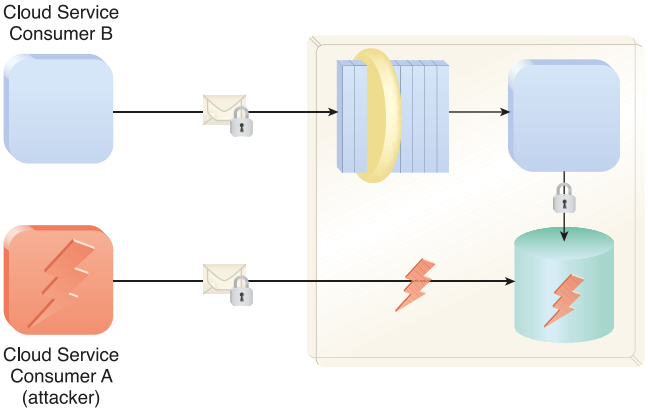


Figure 7.12

Cloud Service Consumer A gains access to a database that was implemented under the assumption that it would only be accessed through a web service with a published service contract (as per Cloud Service Consumer B).

A variation of this attack, known as *weak authentication*, can result when weak passwords or shared accounts are used to protect IT resources. Within cloud environments, these types of attacks can lead to significant impacts depending on the range of IT resources and the range of access to those IT resources the attacker gains (Figure 7.13).

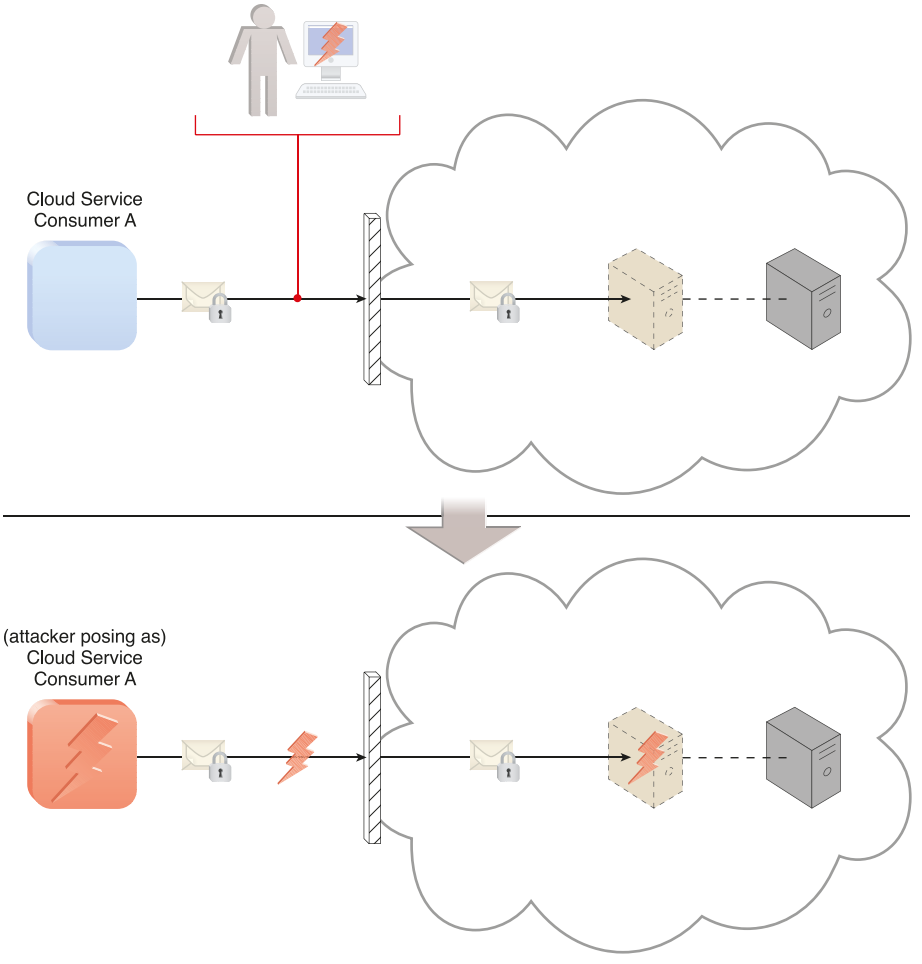


Figure 7.13

An attacker has cracked a weak password used by Cloud Service Consumer A. As a result, a malicious cloud service consumer (owned by the attacker) is designed to pose as Cloud Service Consumer A to gain access to the cloud-based virtual server.

Virtualization Attack

Virtualization provides multiple cloud consumers with access to IT resources that share underlying hardware but are logically isolated from each other. Because cloud providers grant cloud consumers administrative access to virtualized IT resources (such as virtual servers), there is an inherent risk that cloud consumers could abuse this access to attack the underlying physical IT resources.

A *virtualization attack* exploits vulnerabilities in the virtualization platform to jeopardize its confidentiality, integrity, and/or availability. This threat is illustrated in Figure 7.14, where a trusted attacker successfully accesses a virtual server to compromise its underlying physical server. With public clouds, where a single physical IT resource may be providing virtualized IT resources to multiple cloud consumers, such an attack can have significant repercussions.

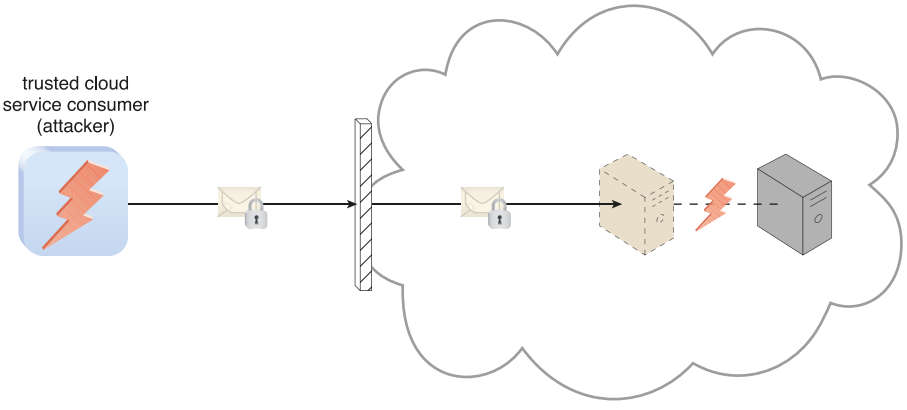


Figure 7.14

An authorized cloud service consumer carries out a virtualization attack by abusing its administrative access to a virtual server to exploit the underlying hardware.

Overlapping Trust Boundaries

If physical IT resources within a cloud are shared by different cloud service consumers, these cloud service consumers have *overlapping trust boundaries*. Malicious cloud service consumers can target shared IT resources with the intention of compromising cloud consumers or other IT resources that share the same trust boundary. The consequence is that some or all of the other cloud service consumers could be impacted by the attack and/or the attacker could use virtual IT resources against others that happen to also share the same trust boundary.

Figure 7.15 illustrates an example in which two cloud service consumers share virtual servers hosted by the same physical server and, resultantly, their respective trust boundaries overlap.

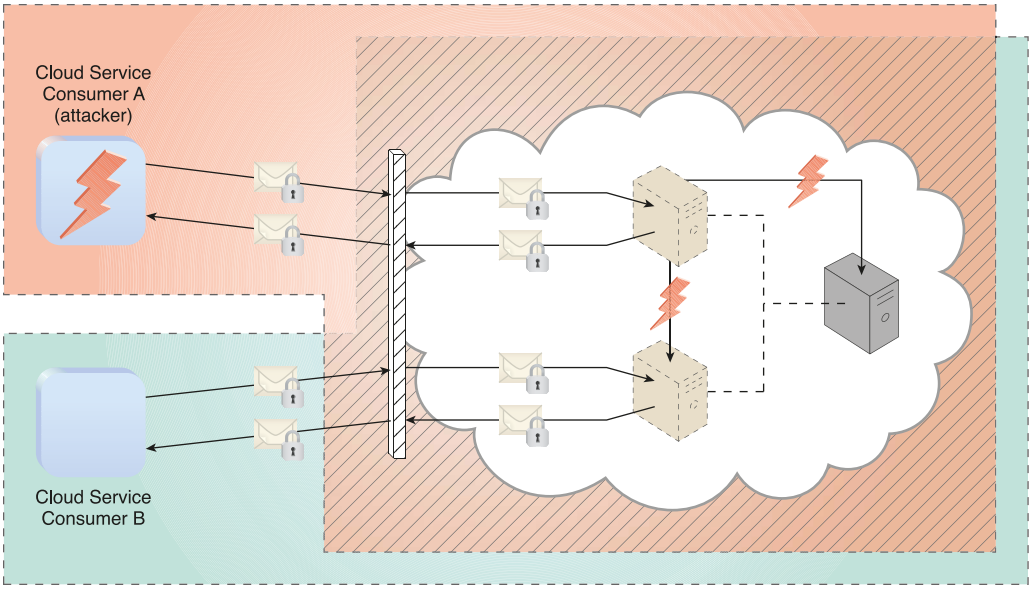


Figure 7.15

Cloud Service Consumer A is trusted by the cloud and therefore gains access to a virtual server, which it then attacks with the intention of attacking the underlying physical server and the virtual server used by Cloud Service Consumer B.

Containerization Attack

The use of containerization introduces a lack of isolation from the host operating system level. Since containers deployed on the same machine share the same host operating system, security threats can increase because access to the entire system can be gained. If the underlying host is compromised, all containers running on the host may be impacted.

Containers can be created from within an operating system running on a virtual server. This can help ensure that, if a security breach occurs that impacts the operating system a container is running on, the attacker can only gain access to and alter the virtual server's operating system or the containers running on a single virtual server, while other virtual servers (or physical servers) remain intact.

Another option is a one service per physical server deployment model, in which all container images deployed on the same host are the same. This can reduce risk without the need to virtualize the IT resources. In this case, a security breach to one cloud service instance would only allow access to other instances, and the residual risk could

be considered as acceptable. However, this approach may not be optimal for deploying many different cloud services because it can significantly increase the total number of physical IT resources that need to be deployed and managed while further increasing cost and operational complexity.

Malware

Malware, also referred to as *malicious software*, is a type of software program designed to cause harm to a computer system or network.

Malware can be used to perform a variety of malicious activities, including:

- stealing protected data
- deleting confidential documents
- listening to private communication
- collecting information about confidential activity

The fundamental basis of a malware attack is the installation of unauthorized software on the victim's computer (Figure 7.16).

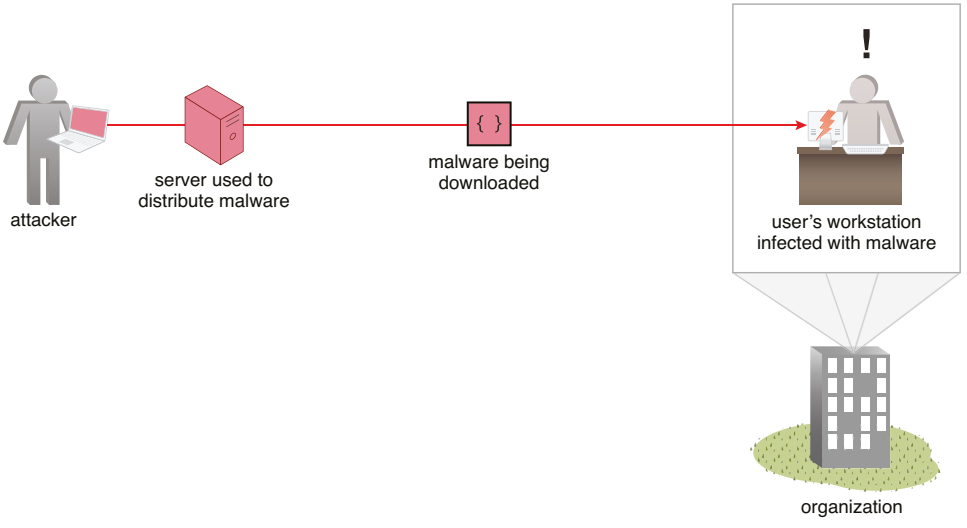


Figure 7.16
An attacker making a server available to a user (via a website, for example) who inadvertently downloads malware to a local workstation.

The following are common types of malware-based cyber attacks:

- *Virus* – Malware that can spread by infecting systems and files with code that enables the virus to replicate and perform additional actions on the infected system.
- *Trojan* – A piece of malicious software that appears to be a legitimate application or service. A Trojan can engage in malicious behavior, often as part of background processes, to carry out activities such as installing backdoor code and injecting code into other running processes. A Trojan may or may not contain a virus.
- *Spyware* – A type of malware that collects information about users or organizations without their knowledge.
- *Adware* – Software designed to display unwanted advertisements or pop-ups. Adware can be considered a security threat because it can collect sensitive information and may slow down systems and make them vulnerable to other types of malware.
- *Ransomware* – Malware that restricts or prevents data usage or access for the purpose of demanding payment of a fee to decrypt or release the data. An ongoing ransomware attack can be carried out using remote code execution (as covered later in this section).
- *Bot* – Malware capable of remotely receiving commands and reporting information to a remote destination. A bot is usually designed to work together with other bots (as per the *Botnet* threat covered later in this section).
- *Rogue Antivirus* – An application claiming to be an antivirus program that, once installed, falsely reports security issues to mislead victims into purchasing a "full" version of the program.
- *Crypto Jacking* – The practice of using browser-based programs that run scripts embedded in web content to mine cryptocurrency without the user's knowledge or consent.
- *Worm* – A self-replicating, self-propagating, self-contained program that uses network mechanisms to spread itself. Worms do not usually cause much harm beyond using up computing resources and are generally not common.

Data science technologies can be used to support malware attacks by analyzing systems to discover and identify new vulnerabilities that can be exploited by malware programs. These technologies can further enable the development of reactive malicious code that can, itself, look for new vulnerabilities.

Insider Threat

An *insider threat* is associated with potential damage that can be inflicted by an organization's staff and others that may have access to the organization's premises or systems.

Common types of insider threats include the following (Figure 7.17):

- *Malicious* – attempts by an insider (such as a disgruntled employee) to access and potentially harm an organization's data, systems, or IT infrastructure
- *Accidental* – accidental damage caused by insiders making mistakes out of ignorance or due to human error, such as accidentally deleting an important file or inadvertently sharing confidential data with an unauthorized party
- *Negligent* – accidental damage caused by insiders due to carelessness or an unwillingness to follow established cybersecurity standards and policies

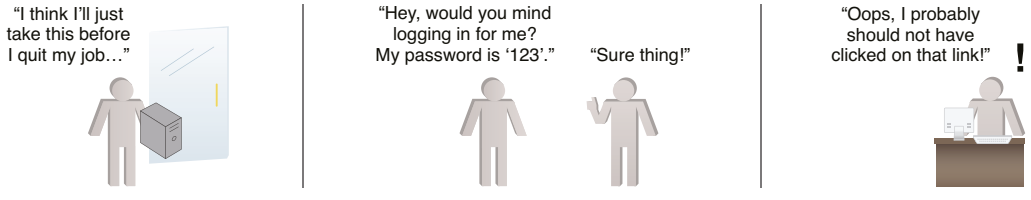


Figure 7.17
Examples of malicious (left), negligent (middle), and accidental (right) insiders posing threats to an organization.

Insider threats can put organizational assets in danger, including physical hardware, physical product inventory, corporate websites, social media communication, and information assets.

Social Engineering and Phishing

Social engineering is a form of attack in which individuals are tricked into revealing sensitive information or performing potentially damaging actions, such as granting access to unauthorized parties (Figure 7.18). Social engineering tactics are popular because it can be easier to exploit people than it is to exploit technology.

Figure 7.18

An example of an attacker attempting to carry out a social engineering attack by extracting sensitive information from an employee who may be working for a cloud consumer or cloud provider organization.

Phishing is a form of social engineering that uses electronic communication, such as sending fraudulent emails that appear to come from valid sources, in an attempt to coerce users into releasing sensitive information, triggering a security breach, or performing other damaging actions.

Botnet

As described earlier in the *Malware* section, a bot is a form of malware capable of receiving and acting upon instructions issued by a remote attacker. A *botnet* attack utilizes multiple bots that are distributed across different hosts to carry out an attack via a coordinated network of bots (a *bot-net*).

A common technique for carrying out a botnet attack is to start with an initial malware infection to create "zombie" hosts. A zombie host is a computer that belongs to an unsuspecting, legitimate organization that an attacker has taken control of. The attacker then typically uses the zombie host to carry out attacks against another party (Figure 7.19).

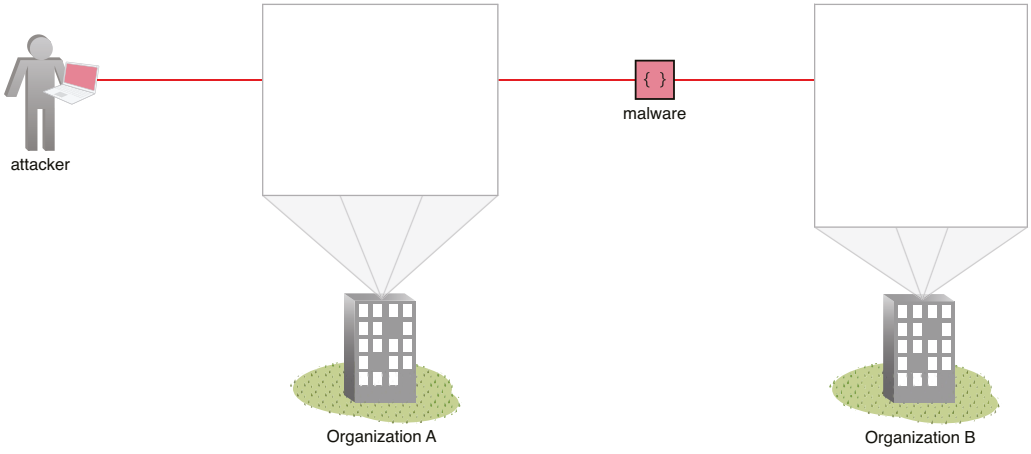


Figure 7.19

An attacker has turned a regular server at Organization A into a zombie server that is controlled by the attacker to transmit malware to a user's computer at Organization B.

A botnet can be comprised of bots located on host servers that belong to the attacker, as well as zombie servers. Once installed, a bot seeks to connect with other bots on other infected hosts and devices to form a network that the attacker can use to perform malicious actions (Figure 7.20), such as carrying out large-scale DDoS attacks and crypto jacking attacks, sending mass emails with harmful content, stealing data, or even recruiting new bots. Botnets can be purchased on the dark web and can be even rented for short periods.

Note that botnet attacks often encompass other attacks and techniques, such as remote code execution, privilege escalation, social engineering, and insider threats.

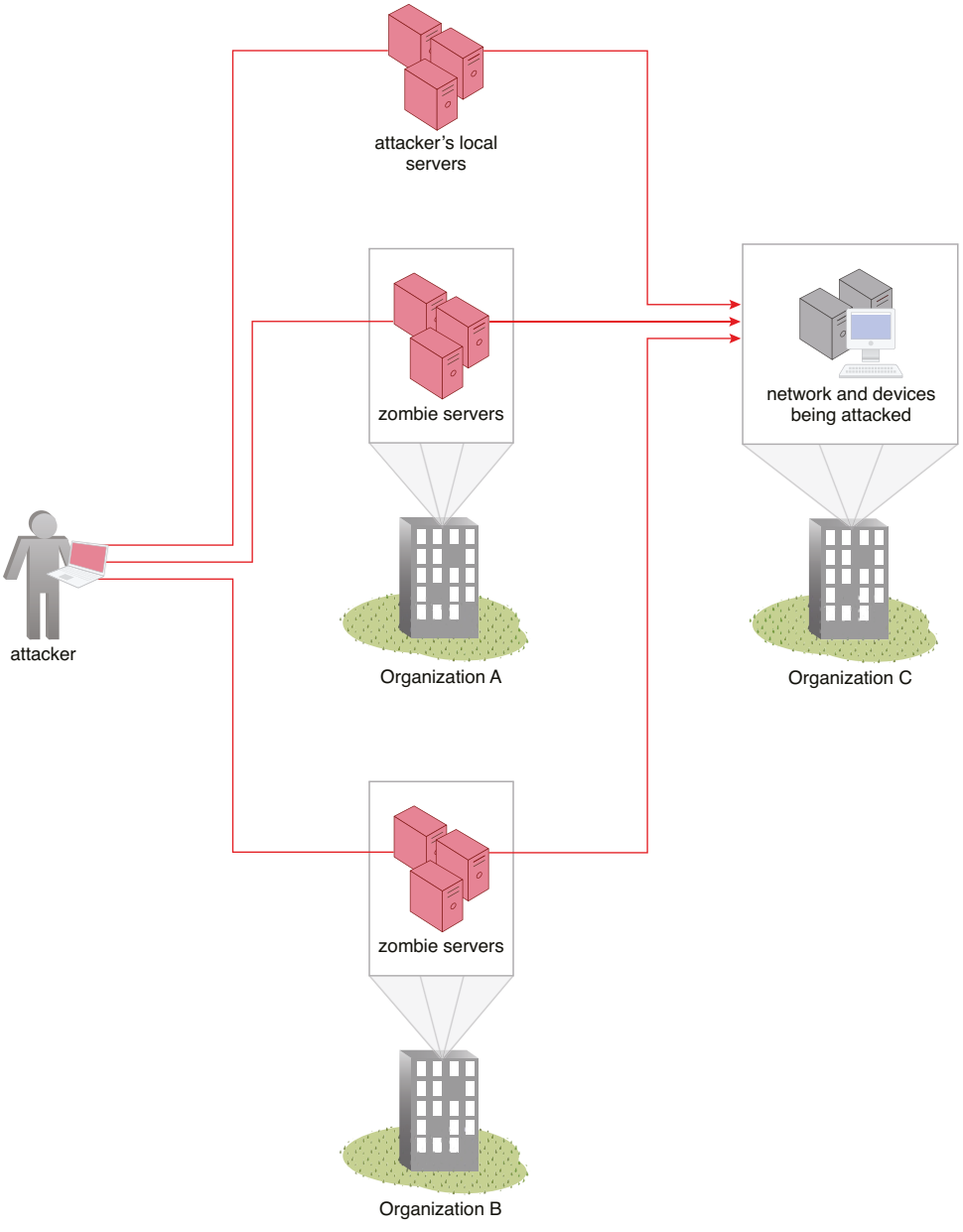


Figure 7.20

An attacker has turned regular servers at Organizations A and B into zombie servers. The attacker uses these servers together with some local servers to carry out an attack on Organization C.

Privilege Escalation

A *privilege escalation* attack occurs when an attacker attempts to gain administrator permissions after compromising a user account with limited access privileges (Figure 7.21). This can be done by exploiting vulnerabilities that unintentionally allow a user account's access levels to be increased.

Figure 7.21
An attacker is able to infiltrate an employee's user account and then exploit a vulnerability to upgrade access privileges.

Data science technologies can be used to support privilege escalation attacks by developing models that can be used to continuously search and analyze potential victim user accounts and systems for exploitable vulnerabilities. For example, another attack could be employed to collect data about how current systems in a network are with regard to third-party software patches. The data science system may be able to process this information, along with additional data about the third-party software programs and how they are configured in the target environment to produce a set of recommended target areas.

Brute Force

In a *brute force* attack, an attacker attempts a broad range of possible username and password combinations to try to determine which one combination is correct and enables the attacker to gain unauthorized access to a system (Figure 7.22).

As such, password-only systems are most vulnerable to brute force attacks, and user accounts with weak passwords are those most easily accessed.

Figure 7.22

An attacker issues a brute force attack by bombarding a website with a series of username and password combinations.

The simplest type of brute force attack is a dictionary attack, in which the attacker reads from a dictionary of possible passwords and essentially tries them all. Credential recycling is another variation, whereby usernames and passwords from prior data breaches are reused to try to break into other systems.

Remote Code Execution

Remote code execution is a cyber attack in which an attacker remotely executes commands on a third party's computing device.

Examples of how this attack can succeed include:

- malicious software (malware) being downloaded by the host (Figure 7.23)
- using tunneling to gain remote access to run host server or database commands or to control system and OS services

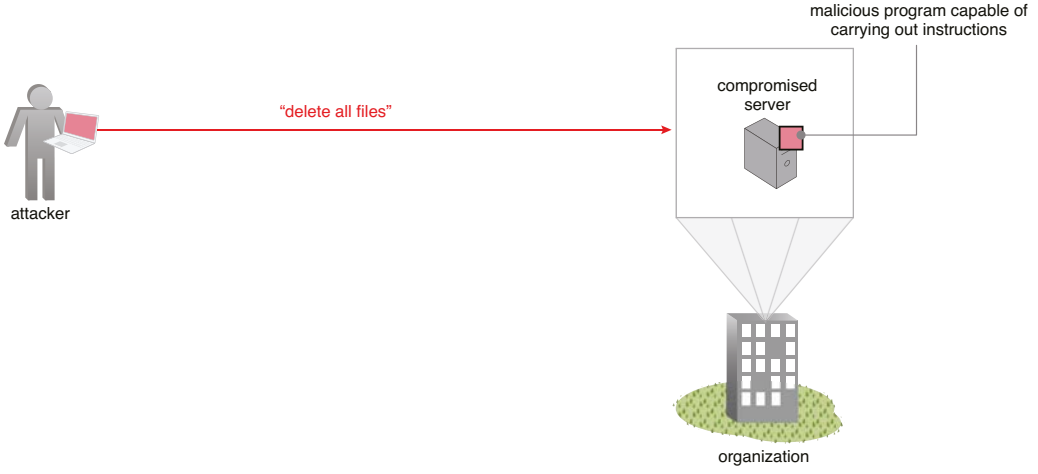


Figure 7.23
Using an already installed malware program, an attacker is able to issue it commands to carry out damaging actions on an organization's server.

This attack can also be enabled by the attacker obtaining login credentials to the host computer via a brute force or Wi-Fi deauthentication attack, or via social engineering and insider threats. A remote code execution attack is usually prefaced by an information-gathering process in which the attacker uses an automated scanning tool to identify vulnerabilities.

The remote code execution technique can be utilized by other cyber attacks, such as botnet attacks and malware attacks that utilize ransomware or Trojans.

SQL Injection

SQL injection is a technique used to attack applications, in which malicious code in the form of SQL statements is inserted into an entry field on a web application user interface, causing the web application server to execute the malicious code (Figure 7.24).

When this attack is successful, access to the server can be compromised, resulting in malware being written into the server's database. Attackers often use search engines to identify vulnerable sites that can be altered using SQL injection.

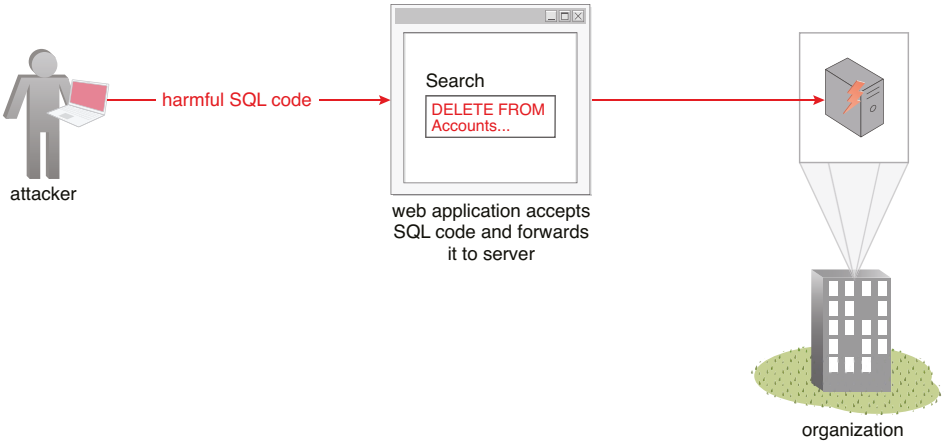


Figure 7.24
An attacker inserts harmful SQL code into a web application's user interface.

NOTE

SQL (or the Structured Query Language) is a syntax used to issue commands to a database, such as queries and updates.

Data science technologies can be used to support SQL injection attacks by analyzing historical SQL commands that have been issued against a given web application to better determine which are more or less effective. The data science system itself can help generate different combinations of SQL code for automated attacks, learning and improving over time, based on the successful or failed outcome of each code submission.

Tunneling

Tunneling is a technique whereby data is embedded in an authorized protocol packet to bypass firewall controls, allowing sensitive data to exit the network and unauthorized or malicious data to enter without ever triggering an alert or log entry (Figure 7.25). Tunneling can be difficult to detect and block because tunneling packets are designed to adhere to the rules of firewalls.

To "tunnel" the data, the attacker uses a software program that can pretend to talk to the protocol but, in reality, transfers data for some other purpose. For example, an established tunnel can be used to place malware on the victim's computer, such as spyware

that remains on a host for a prolonged period to collect confidential information. It can also be used with remote code execution in support of a botnet attack by enabling the attacker to place bots on hosts for the purpose of turning them into zombie servers.

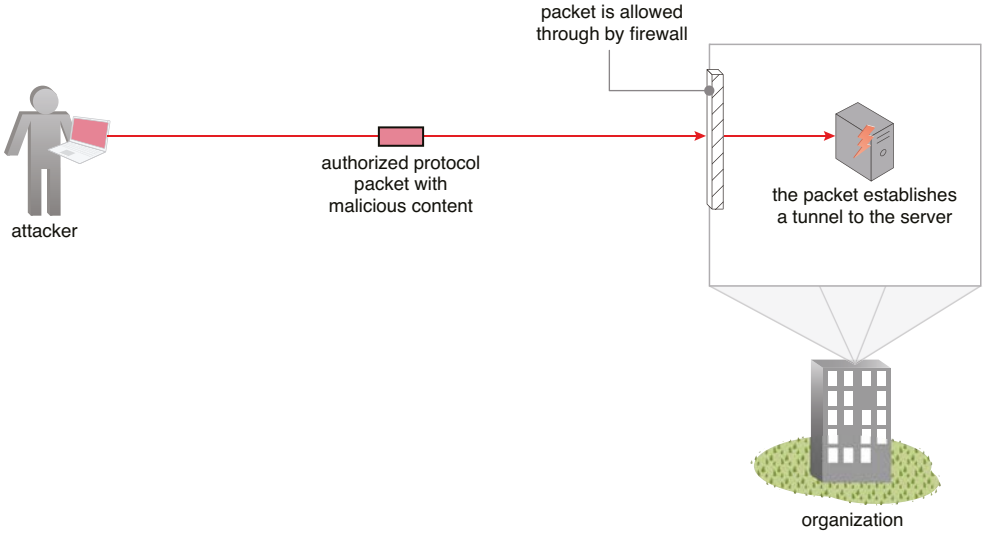


Figure 7.25
An attacker is able to get a malicious packet through an organization's firewall, thereby enabling the attacker to set up a tunnel to an internal server.

NOTE

Commonly used protocols for attacking systems using tunneling techniques include HTTP, SSH, DNS, and ICMP.

Advanced Persistent Threat (APT)

An *advanced persistent threat (APT)* is a method whereby an attacker uses multiple attacks to breach security. Often, the attacks are coordinated to take place over a longer period of time (Figure 7.26). APTs require sophisticated technology and long-term preparation and planning by the attacker, and are therefore more common with attackers that target high-value organizations.

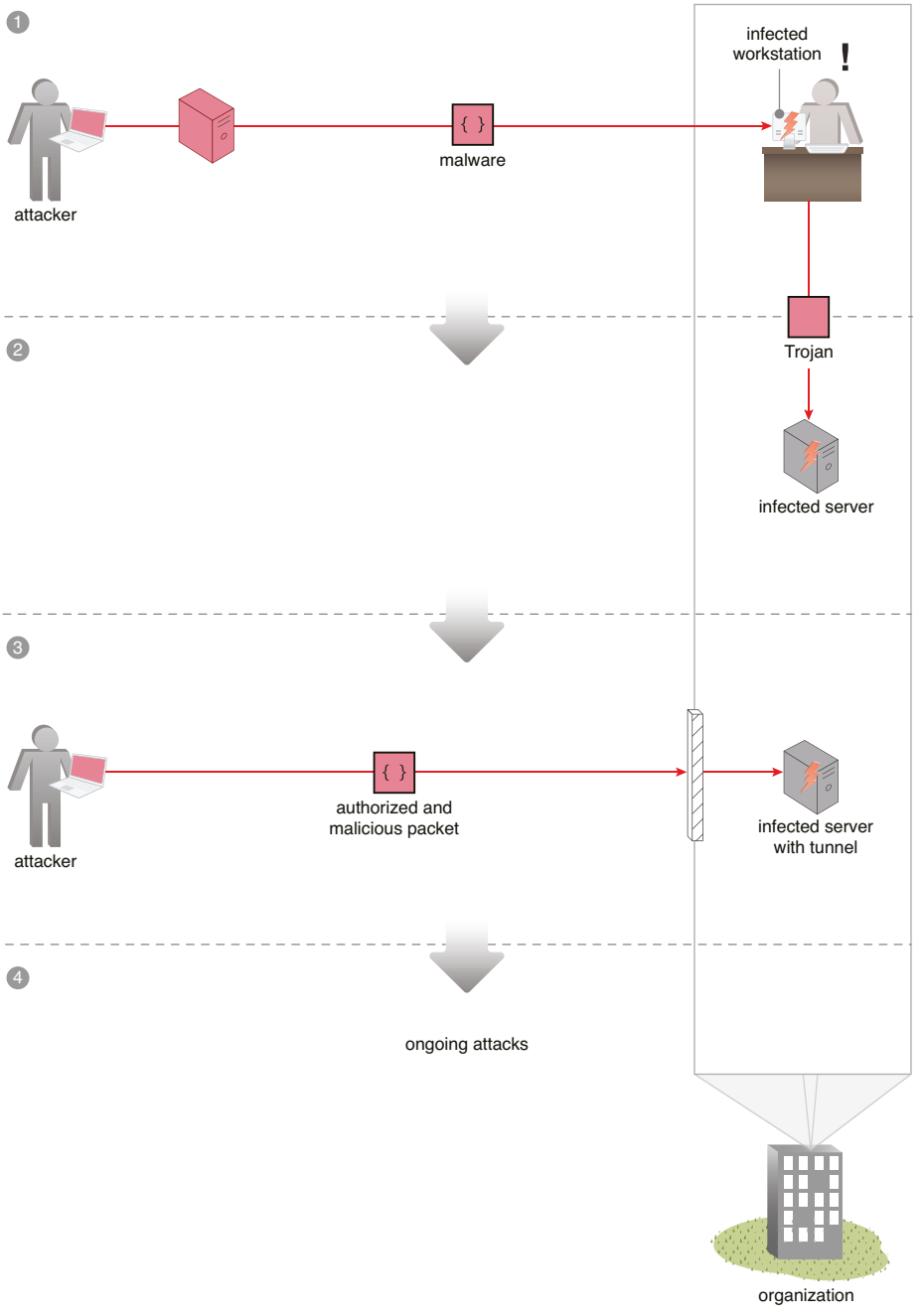


Figure 7.26

A set of coordinated attacks is carried out against an organization over a period of time. The different attacks are carried out in a specific sequence and in support of a common objective.

An objective behind APTs can be to position resources within an organization's environment subsequent to gaining access via a security breach. For example, an APT attack may succeed in gaining access to a network after which the attacker tries to establish a foothold by implanting malware that creates backdoors and tunnels that are then used to continue to *persistently* attack systems over a longer period of time.

The attacker may then attempt to deepen access by using techniques, such as brute force attacks, to gain administrator rights that then enable the attacker to take control of system resources and perhaps even lock others out.

Because successful APT attacks are carried out as a larger campaign over longer periods of time, they enable attackers to observe and learn about an environment to discover further ways to harvest information or value (or do more damage) than what the attackers originally planned.

A critical success factor of APT attacks is often human involvement. Many APT attacks succeed as a result of an insider threat, which can be a human who (possibly inadvertently) has compromised security via social engineering or phishing techniques.

NOTE

Groups of attackers that collectively carry out APT attacks are known as *APT groups*. An APT group can include access brokers that only obtain and sell access information to attackers. For example, the use of access brokers is common among ransomware attackers.

7.5 CASE STUDY EXAMPLE

DTGOV, acting as a third-party provider to so many different government organizations, undergoes a review to identify which threats it will likely be most vulnerable to.

The results indicate the following primary concerns:

- *Virtualization Attacks* – This is a completely new type of attack it had not been prepared for before using cloud services on behalf of its customers.
- *Overlapping Trust Boundaries* – Given that all of its customers will now be sharing resources from a cloud provider, they will be subject to this new threat.

- *Social Engineering and Phishing* – As a service provider, DTGOV has no control over the behavior of the end users of the systems it runs and manages.

DTGOV plans to mitigate all of these threats by revising its customer agreements and by utilizing a number of the security mechanisms covered in Chapters 10 and 11.

7.6 Additional Considerations

This section provides a diverse checklist of issues and guidelines that relate to cloud security. The listed considerations are in no particular order.

Flawed Implementations

The substandard design, implementation, or configuration of cloud service deployments can have undesirable consequences, beyond runtime exceptions and failures. If the cloud provider's software and/or hardware have inherent security flaws or operational weaknesses, attackers can exploit these vulnerabilities to impair the integrity, confidentiality, and/or availability of cloud provider IT resources and cloud consumer IT resources hosted by the cloud provider.

Figure 7.27 depicts a poorly implemented cloud service that results in a server shutdown. Although in this scenario the flaw is exposed accidentally by a legitimate cloud service consumer, it could have easily been discovered and exploited by an attacker.

Security Policy Disparity

When a cloud consumer places IT resources with a public cloud provider, it may need to accept that its traditional information security approach may not be identical or even similar to that of the cloud provider. This incompatibility needs to be assessed to ensure that any data or other IT assets being relocated to a public cloud are adequately protected. Even when leasing raw infrastructure-based IT resources, the cloud consumer may not be granted sufficient administrative control or influence over security policies that apply to the IT resources leased from the cloud provider. This is primarily because those IT resources are still legally owned by the cloud provider and continue to fall under its responsibility.

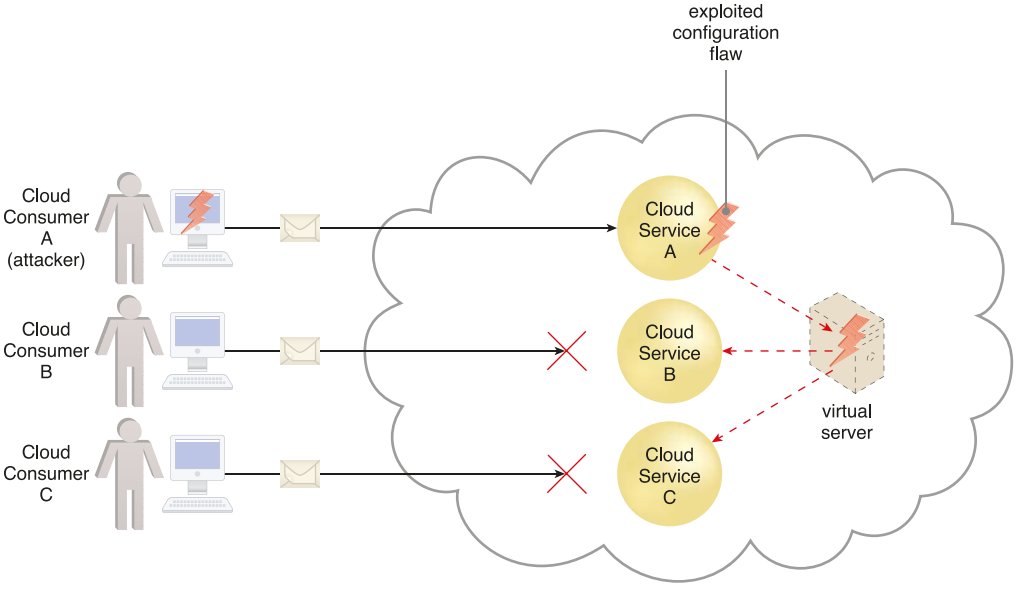


Figure 7.27

Cloud Service Consumer A's message triggers a configuration flaw in Cloud Service A, which in turn causes the virtual server that is also hosting Cloud Services B and C to crash.

Furthermore, with some public clouds, additional third parties, such as security brokers and certificate authorities, may introduce their own distinct set of security policies and practices, further complicating any attempts to standardize the protection of cloud consumer assets.

Contracts

Cloud consumers need to carefully examine contracts and SLAs put forth by cloud providers to ensure that security policies, and other relevant guarantees, are satisfactory when it comes to asset security. There needs to be clear language that indicates the amount of liability assumed by the cloud provider and/or the level of indemnity the cloud provider may ask for. The greater the assumed liability by the cloud provider, the lower the risk to the cloud consumer.

Another aspect of contractual obligations is where the lines are drawn between cloud consumer and cloud provider assets. A cloud consumer that deploys its own solution upon infrastructure supplied by the cloud provider will produce a technology

architecture comprised of artifacts owned by both the cloud consumer and the cloud provider. If a security breach (or other type of runtime failure) occurs, how is blame determined? Furthermore, if the cloud consumer can apply its own security policies to its solution, but the cloud provider insists that its supporting infrastructure be governed by different (and perhaps incompatible) security policies, how can the resulting disparity be overcome?

Sometimes the best solution is to look for a different cloud provider with more compatible contractual terms.

Risk Management

When assessing the potential impacts and challenges pertaining to cloud adoption, cloud consumers are encouraged to perform a formal risk assessment as part of a risk management strategy. A cyclically executed process used to enhance strategic and tactical security, risk management is comprised of a set of coordinated activities for overseeing and controlling risks. The main activities are generally defined as risk assessment, risk treatment, and risk control (Figure 7.28).

- *Risk Assessment* – In the risk assessment stage, the cloud environment is analyzed to identify potential vulnerabilities and shortcomings that threats can exploit. The cloud provider can be asked to produce statistics and other information about past attacks (successful and unsuccessful) carried out in its cloud. The identified risks are quantified and qualified according to the probability of occurrence and the degree of impact in relation to how the cloud consumer plans to utilize cloud-based IT resources.
- *Risk Treatment* – Mitigation policies and plans are designed during the risk treatment stage with the intent of successfully treating the risks that were discovered during risk assessment. Some risks can be eliminated, others can be mitigated, while others can be dealt with via outsourcing or even incorporated into the insurance and/or operating loss budgets. The cloud provider itself may agree to assume responsibility as part of its contractual obligations.
- *Risk Control* – The risk control stage is related to risk monitoring, a three-step process that is comprised of surveying related events, reviewing these events to determine the effectiveness of previous assessments and treatments, and identifying any policy adjustment needs. Depending on the nature of the monitoring required, this stage may be carried out or shared by the cloud provider.

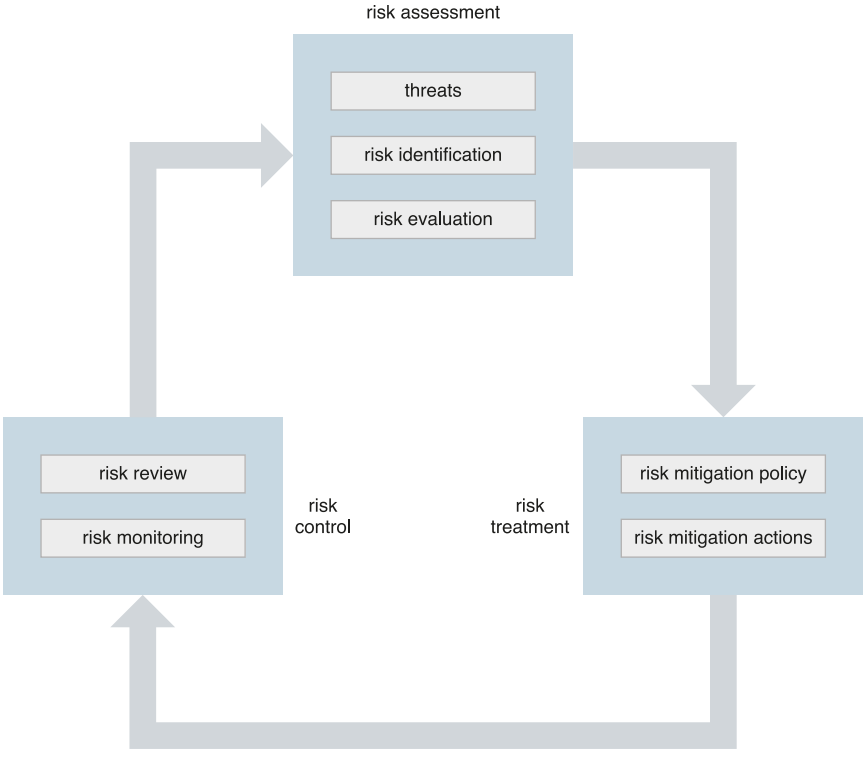


Figure 7.28
The ongoing risk management process, which can be initiated from any of the three stages.

The threat agents and cloud security threats covered in this chapter (as well as others that may surface) can be identified and documented as part of the risk assessment stage. The cloud security and cybersecurity mechanisms covered in Chapters 10 and 11 can be documented and referenced as part of the corresponding risk treatment.

7.7 CASE STUDY EXAMPLE

Based on an assessment of its internal applications, ATN analysts identify a set of risks. One such risk is associated with the myTrendek application that was adopted from OTC, a company ATN recently acquired. This application includes a feature that analyzes telephone and internet usage and enables a multiuser mode that grants

varying access rights. Administrators, supervisors, auditors, and regular users can therefore be assigned different privileges. The application's user base encompasses internal users and external users, such as business partners and contractors.

The myTrendek application poses a number of security challenges pertaining to usage by internal staff:

- authentication does not require or enforce complex passwords
- communication with the application is not encrypted
- European regulations (ETelReg) require that certain types of data collected by the application be deleted after six months

ATN is planning to migrate this application to a cloud via a PaaS environment, but the weak authentication threat and the lack of confidentiality supported by the application make them reconsider. A subsequent risk assessment further reveals that if the application is migrated to a PaaS environment hosted by a cloud that resides outside of Europe, local regulations may be in conflict with ETelReg. Given that the cloud provider is not concerned with ETelReg compliance, this could easily result in monetary penalties being assessed to ATN. Based on the results of the risk assessment, ATN decides not to proceed with its cloud migration plan.

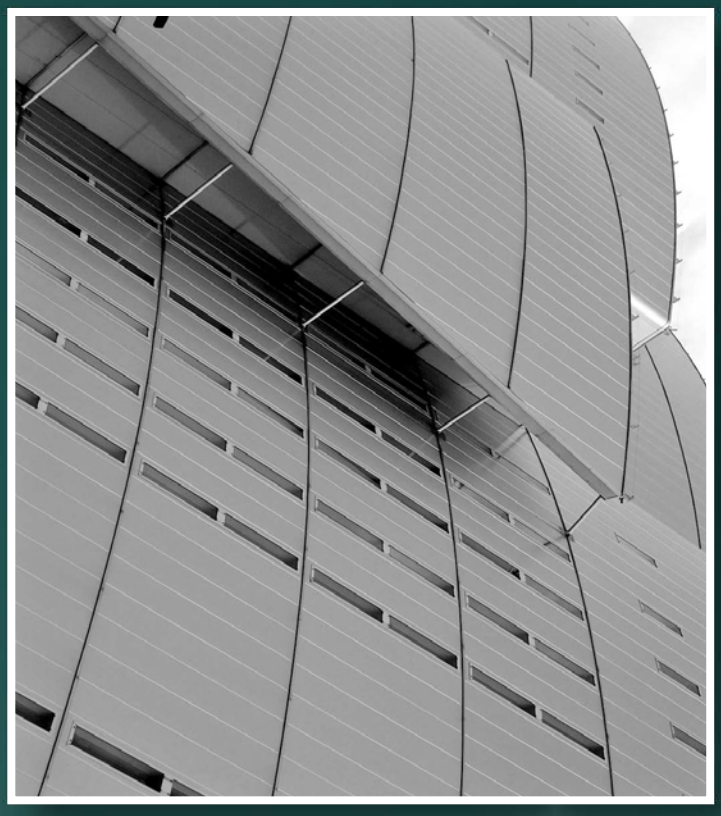
Part II

Cloud Computing Mechanisms

Technology mechanisms represent well-defined IT artifacts that are established within the IT industry and commonly distinct to a certain computing model or platform. The technology-centric nature of cloud computing requires the establishment of a formal set of mechanisms that act as building blocks for cloud technology architectures.

The chapters in this part of the book define 48 common cloud computing mechanisms that can be combined in different and alternative variations.

Select mechanisms are further referenced in the architectural models covered in *Part III: Cloud Computing Architecture*.



Chapter 8

Cloud Infrastructure Mechanisms

Cloud infrastructure mechanisms are foundational building blocks of cloud environments that establish primary artifacts to form the basis of fundamental cloud technology architecture.

The following cloud infrastructure mechanisms are described in this chapter:

- Logical Network Perimeter
- Virtual Server
- Hypervisor
- Cloud Storage Device
- Cloud Usage Monitor
- Resource Replication
- Ready-Made Environment
- Container

Not all of these mechanisms are necessarily broad-reaching, nor does each establish an individual architectural layer. Instead, they should be viewed as core components that are common to cloud platforms.

8.1 Logical Network Perimeter

Defined as the isolation of a network environment from the rest of a communications network, the *logical network perimeter* establishes a virtual network boundary that can encompass and isolate a group of related cloud-based IT resources that may be physically distributed (Figure 8.1).

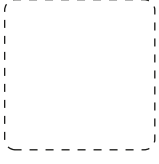


Figure 8.1
The dashed line notation used to indicate the boundary of a logical network perimeter.

This mechanism can be implemented to:

- isolate IT resources in a cloud from non-authorized users
- isolate IT resources in a cloud from nonusers

- isolate IT resources in a cloud from cloud consumers
- control the bandwidth that is available to isolated IT resources

Logical network perimeters are typically established via network devices that supply and control the connectivity of a data center and are commonly deployed as virtualized IT environments that include:

- *Virtual Firewall* – An IT resource that actively filters network traffic to and from the isolated network while controlling its interactions with the internet.
- *Virtual Network* – Usually acquired through VLANs, this IT resource isolates the network environment within the data center infrastructure.

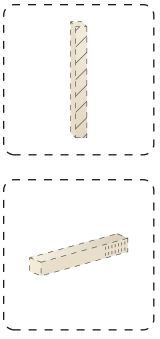


Figure 8.2
The symbols used to represent a virtual firewall (top) and a virtual network (bottom).

Figure 8.2 introduces the notation used to denote these two IT resources. Figure 8.3 depicts a scenario in which one logical network perimeter contains a cloud consumer's on-premises environment, while another contains a cloud provider's cloud-based environment. These perimeters are connected through a VPN that protects communications, since the VPN is typically implemented by point-to-point encryption of the data packets sent between the communicating endpoints.

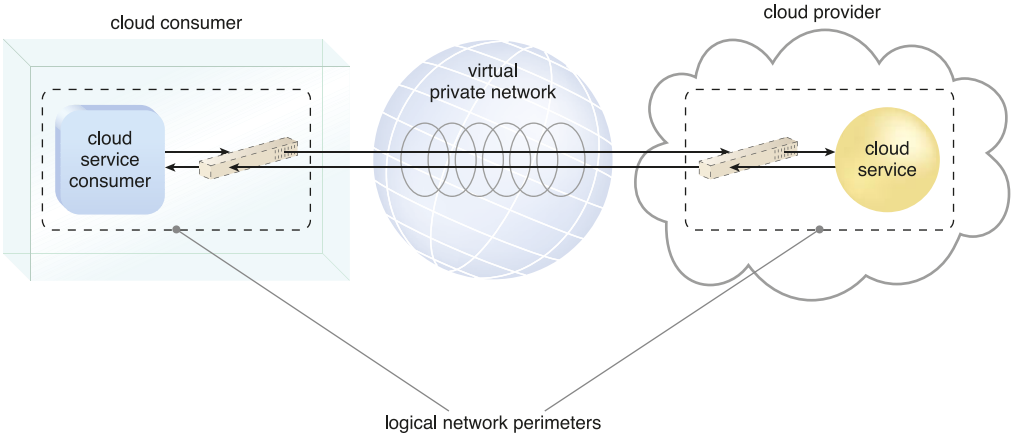


Figure 8.3
Two logical network perimeters surround the cloud consumer and cloud provider environments.

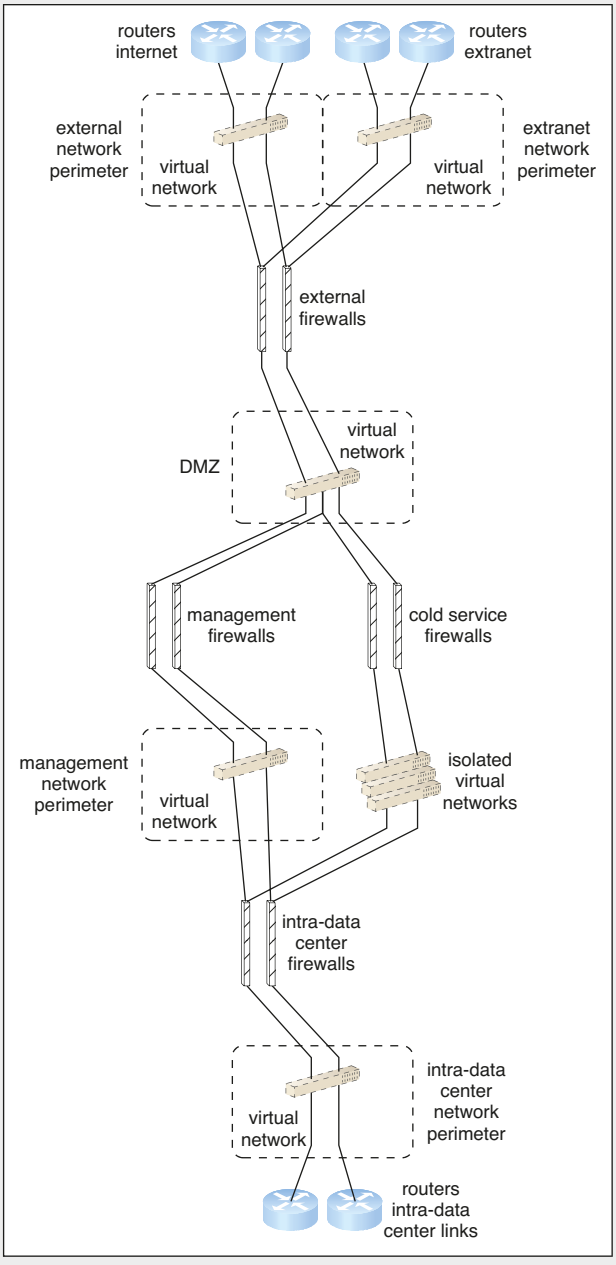
CASE STUDY EXAMPLE

DTGOV has virtualized its network infrastructure to produce a logical network layout favoring network segmentation and isolation. Figure 8.4 depicts the logical network perimeter implemented at each DTGOV data center, as follows:

- The routers that connect to the internet and extranet are networked to external firewalls, which provide network control and protection to the furthest external network boundaries using virtual networks that logically abstract the external network and extranet perimeters. Devices connected to these network perimeters are loosely isolated and protected from external users. No cloud consumer IT resources are available within these perimeters.
- A logical network perimeter classified as a demilitarized zone (DMZ) is established between the external firewalls and its own firewalls. The DMZ is abstracted as a virtual network hosting the proxy servers (not shown in Figure 8.3) that provide intermediate access to commonly used network services (DNS, email, web portal), as well as web servers with external management functions.
- The network traffic leaving the proxy servers passes through a set of management firewalls that isolate the management network perimeter, which hosts the servers providing the bulk of the management services that cloud consumers can externally access. These services are provided in direct support of self-service and on-demand allocation of cloud-based IT resources.
- All the traffic to cloud-based IT resources flows through the DMZ to the cloud service firewalls that isolate every cloud consumer's perimeter network, which is abstracted by a virtual network that is also isolated from other networks.
- Both the management perimeter and isolated virtual networks are connected to the intra-data center firewalls, which regulate the network traffic to and from the other DTGOV data centers that are also connected to intra-data center routers at the intra-data center network perimeter.

The virtual firewalls are allocated to and controlled by a single cloud consumer in order to regulate its virtual IT resource traffic. These IT resources are connected through a virtual network that is isolated from other cloud consumers. The virtual firewall and the isolated virtual network jointly form the cloud consumer's logical network perimeter.

Figure 8.4
A logical network layout is established through a set of logical network perimeters using various firewalls and virtual networks.



8.2 Virtual Server

A *virtual server* is a form of virtualization software that emulates a physical server. Virtual servers are used by cloud providers to share the same physical server with multiple cloud consumers by providing cloud consumers with individual virtual server instances. Figure 8.5 shows three virtual servers being hosted by two physical servers. The number of instances a given physical server can share is limited by its capacity.

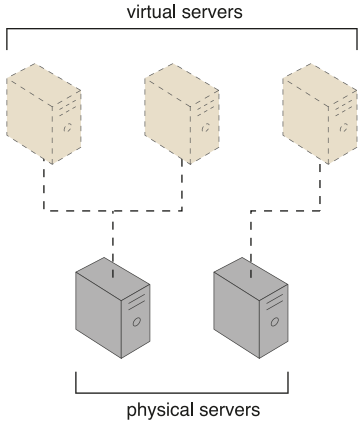


Figure 8.5
The first physical server hosts two virtual servers, while the second physical server hosts one virtual server.

NOTE

- The terms *virtual server* and *virtual machine (VM)* are used synonymously throughout this book.
- The virtual infrastructure manager (VIM) referenced in this chapter is described in Chapter 12 as part of the *Resource Management System* section.

As a commodity mechanism, the virtual server represents the most foundational building block of cloud environments. Each virtual server can host numerous IT resources, cloud-based solutions, and various other cloud computing mechanisms. The instantiation of virtual servers from image files is a resource allocation process that can be completed rapidly and on-demand.

Cloud consumers that install or lease virtual servers can customize their environments independently from other cloud consumers that may be using virtual servers hosted by the same underlying physical server. Figure 8.6 depicts a virtual server that hosts a

cloud service being accessed by Cloud Service Consumer B, while Cloud Service Consumer A accesses the virtual server directly to perform an administration task.

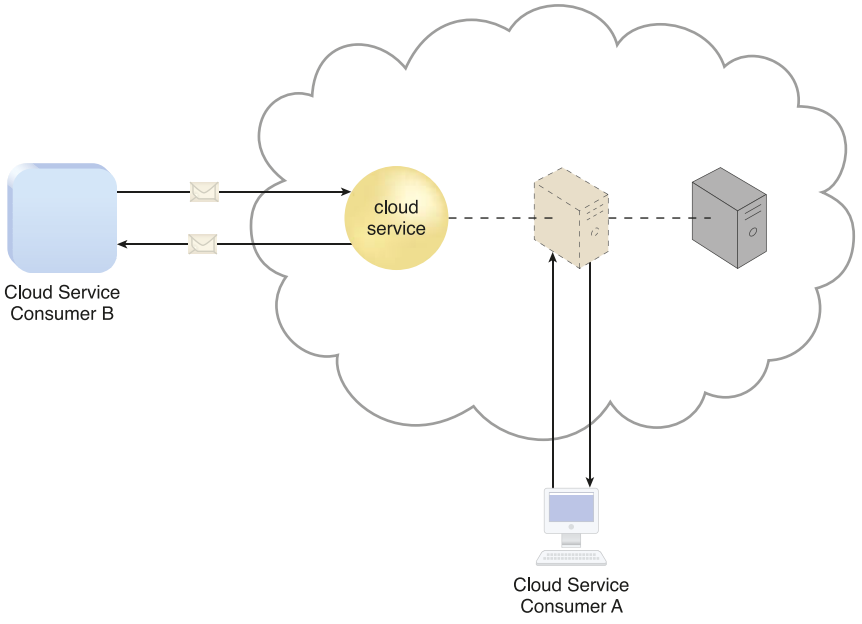


Figure 8.6
A virtual server hosts an active cloud service and is further accessed by a cloud consumer for administrative purposes.

CASE STUDY EXAMPLE

DTGOV's IaaS environment contains hosted virtual servers that were instantiated on physical servers running the same hypervisor software that controls the virtual servers. Their VIM is used to coordinate the physical servers in relation to the creation of virtual server instances. This approach is used at each data center to apply a uniform implementation of the virtualization layer.

Figure 8.7 depicts several virtual servers running over physical servers, all of which are jointly controlled by a central VIM.

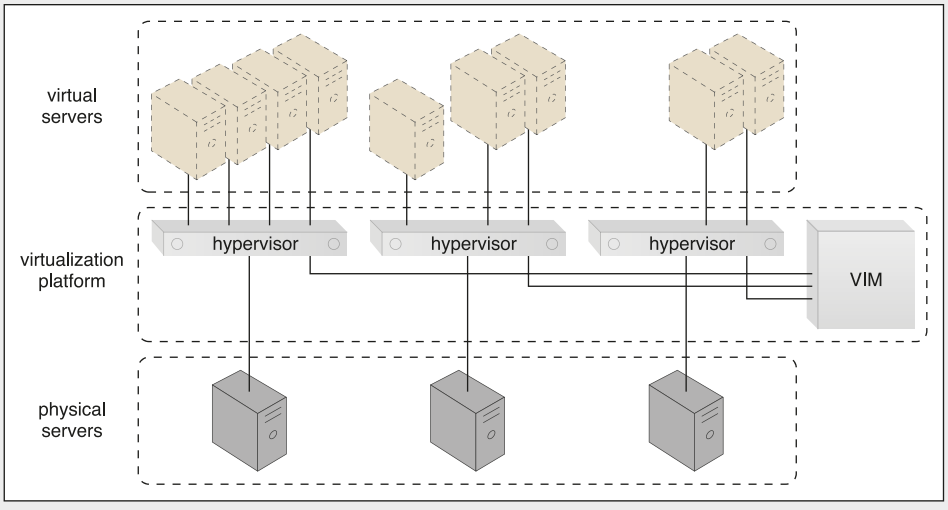


Figure 8.7
Virtual servers are created via the physical servers' hypervisors and a central VIM.

To enable the on-demand creation of virtual servers, DTGOV provides cloud consumers with a set of template virtual servers that are made available through premade VM images.

These VM images are files that represent the virtual disk images used by the hypervisor to boot the virtual server. DTGOV enables the template virtual servers to have various initial configuration options that differ, based on operating system, drivers, and management tools being used. Some template virtual servers also have additional, preinstalled application server software.

The following virtual server packages are offered to DTGOV's cloud consumers. Each package has different predefined performance configurations and limitations:

- *Small Virtual Server Instance* – 1 virtual processor core, 4 GB of virtual RAM, 20 GB of storage space in the root file system
- *Medium Virtual Server Instance* – 2 virtual processor cores, 8 GB of virtual RAM, 20 GB of storage space in the root file system

- *Large Virtual Server Instance* – 8 virtual processor cores, 16 GB of virtual RAM, 20 GB of storage space in the root file system
- *Memory Large Virtual Server Instance* – 8 virtual processor cores, 64 GB of virtual RAM, 20 GB of storage space in the root file system
- *Processor Large Virtual Server Instance* – 32 virtual processor cores, 16 GB of virtual RAM, 20 GB of storage space in the root file system
- *Ultra-Large Virtual Server Instance* – 128 virtual processor cores, 512 GB of virtual RAM, 40 GB of storage space in the root file system

Additional storage capacity can be added to a virtual server by attaching a virtual disk from a cloud storage device. All the template virtual machine images are stored on a common cloud storage device that is accessible only through the cloud consumers' management tools that are used to control the deployed IT resources. Once a new virtual server needs to be instantiated, the cloud consumer can choose the most suitable virtual server template from the list of available configurations. A copy of the virtual machine image is made and allocated to the cloud consumer, who can then assume the administrative responsibilities.

The allocated VM image is updated whenever the cloud consumer customizes the virtual server. After the cloud consumer initiates the virtual server, the allocated VM image and its associated performance profile are passed to the VIM, which creates the virtual server instance from the appropriate physical server.

DTGOV uses the process illustrated in Figure 8.8 to support the creation and management of virtual servers that have different initial software configurations and performance characteristics.

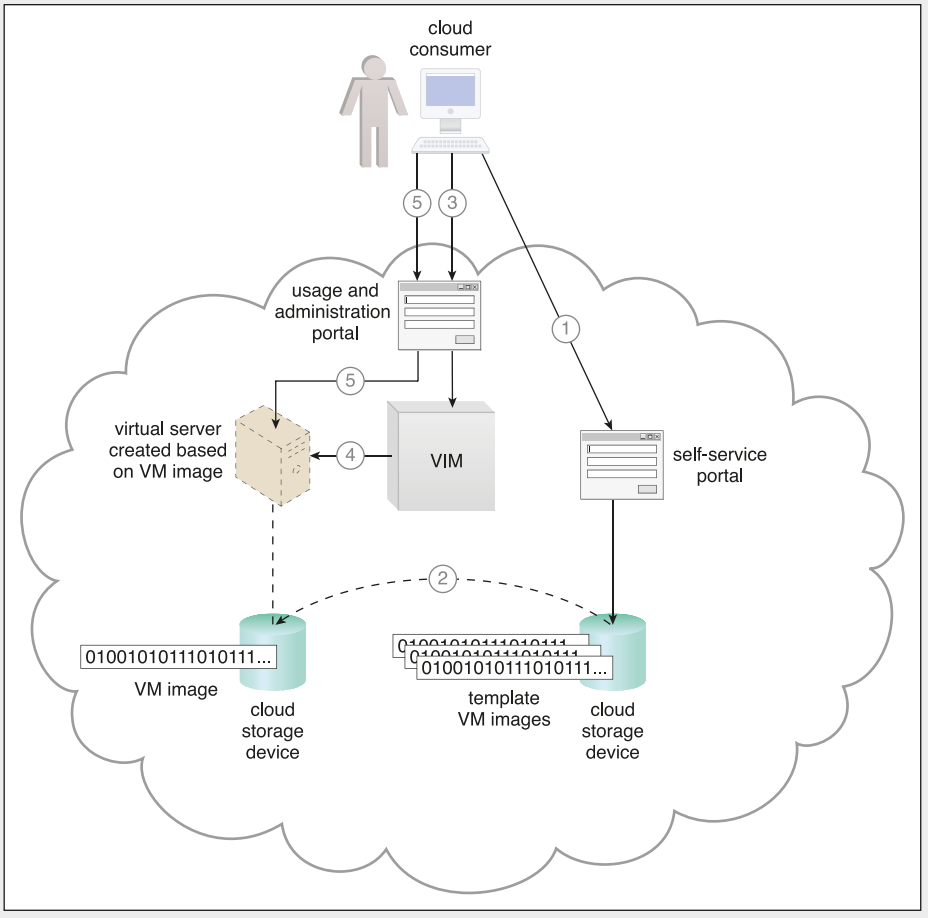


Figure 8.8

The cloud consumer uses the self-service portal to select a template virtual server for creation (1). A copy of the corresponding VM image is created in a cloud consumer–controlled cloud storage device (2). The cloud consumer initiates the virtual server using the usage and administration portal (3), which interacts with the VIM to create the virtual server instance via the underlying hardware (4). The cloud consumer is able to use and customize the virtual server via other features on the usage and administration portal (5). (Note that the self-service portal and the usage and administration portal are explained in Chapter 12.)

8.3 Hypervisor

The *hypervisor* mechanism is a fundamental part of virtualization infrastructure that is primarily used to generate virtual server instances of a physical server. A hypervisor is generally limited to one physical server and can therefore only create virtual images of that server (Figure 8.9). Similarly, a hypervisor can only assign virtual servers it generates to resource pools that reside on the same underlying physical server. A hypervisor has limited virtual server management features, such as increasing the virtual server's capacity or shutting it down. The VIM provides a range of features for administering multiple hypervisors across physical servers.

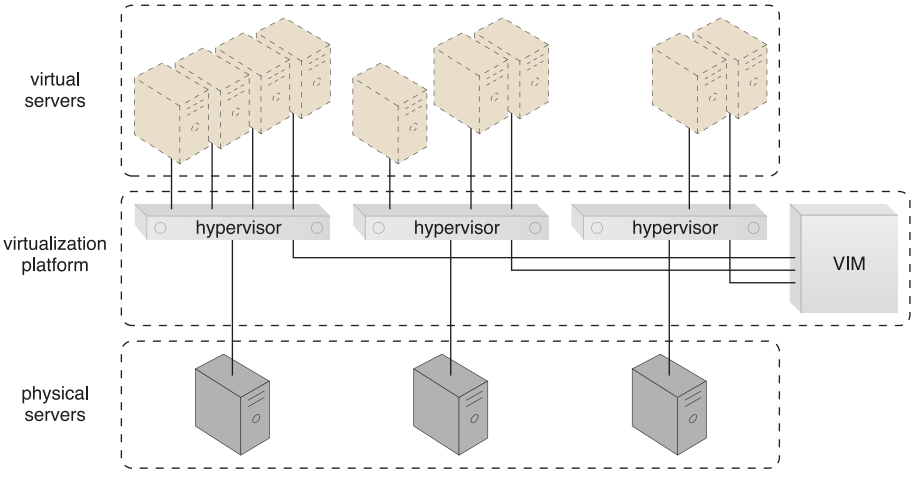


Figure 8.9

Virtual servers are created via individual hypervisors on individual physical servers. All three hypervisors are jointly controlled by the same VIM.

Hypervisor software can be installed directly in bare-metal servers and provides features for controlling, sharing, and scheduling the usage of hardware resources, such as processor power, memory, and I/O. These can appear to each virtual server's operating system as dedicated resources.

CASE STUDY EXAMPLE

DTGOV has established a virtualization platform in which the same hypervisor software product is running on all physical servers. The VIM coordinates the hardware resources in each data center so that virtual server instances can be created from the most expedient underlying physical server. As a result, cloud consumers are able to lease virtual servers with auto-scaling features.

To offer flexible configurations, the DTGOV virtualization platform provides live VM migration of virtual servers among physical servers inside the same data center. This is illustrated in Figures 8.10 and 8.11, where a virtual server live-migrates from one busy physical server to another that is idle, allowing it to scale up in response to an increase in its workload.

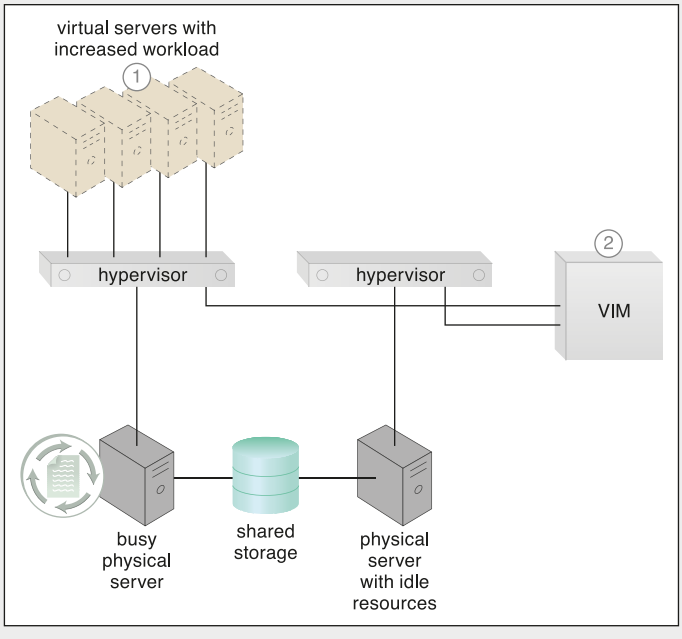


Figure 8.10
A virtual server capable of auto-scaling experiences an increase in its workload (1). The VIM decides that the virtual server cannot scale up because its underlying physical server host is being used by other virtual servers (2).

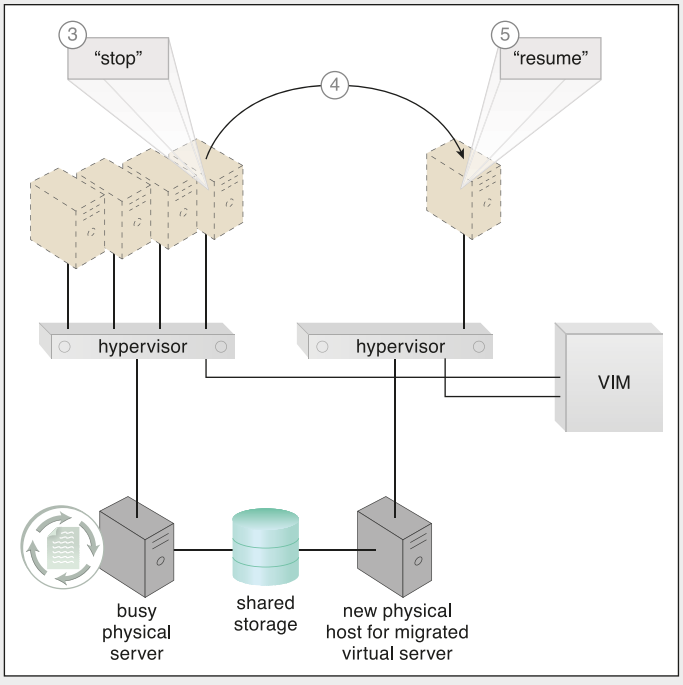


Figure 8.11
The VIM commands the hypervisor on the busy physical server to suspend execution of the virtual server (3). The VIM then commands the instantiation of the virtual server on the idle physical server. State information (such as dirty memory pages and processor registers) is synchronized via a shared cloud storage device (4). The VIM commands the hypervisor at the new physical server to resume the virtual server processing (5).

8.4 Cloud Storage Device

The *cloud storage device* mechanism represents storage devices that are designed specifically for cloud-based provisioning. Instances of these devices can be virtualized, similar to how physical servers can spawn virtual server images. Cloud storage devices are commonly able to provide fixed-increment capacity allocation in support of the pay-per-use mechanism. Cloud storage devices can be exposed for remote access via cloud storage services.

NOTE

This is a parent mechanism that represents cloud storage devices in general. There are numerous specialized cloud storage devices, several of which are described in the architectural models covered in Part III of this book.

A primary concern related to cloud storage is the security, integrity, and confidentiality of data, which becomes more prone to being compromised when entrusted to external cloud providers and other third parties. There can also be legal and regulatory implications that result from relocating data across geographical or national boundaries. Another issue applies specifically to the performance of large databases. LANs provide locally stored data with network reliability and latency levels that are superior to those of WANs.

Cloud Storage Levels

Cloud storage device mechanisms provide common logical units of data storage, such as:

- *Files* – Collections of data are grouped into files that are located in folders.
- *Blocks* – The lowest level of storage and the closest to the hardware, a block is the smallest unit of data that is still individually accessible.
- *Datasets* – Sets of data are organized into a table-based, delimited, or record format.
- *Objects* – Data and its associated metadata are organized as web-based resources.

Each of these data storage levels is commonly associated with a certain type of technical interface which corresponds to a particular type of cloud storage device and cloud storage service used to expose its API (Figure 8.12).

Network Storage Interfaces

Legacy network storage most commonly falls under the category of network storage interfaces. It includes storage devices in compliance with industry standard protocols, such as SCSI for storage blocks and the server message block (SMB), common internet file system (CIFS), and network file system (NFS) for file and network storage. File storage entails storing individual data in separate files that can be different sizes and formats and organized into folders and subfolders. Original files are often replaced by the new files that are created when data has been modified.

When a cloud storage device mechanism is based on this type of interface, its data searching and extraction performance will tend to be suboptimal. Storage processing levels and thresholds for file allocation are usually determined by the file system itself. Block storage requires data to be in a fixed format (known as a *data block*), which is the smallest unit that can be stored and accessed and the storage format closest to hardware. Using either the logical unit number (LUN) or virtual volume block-level storage will typically have better performance than file-level storage.

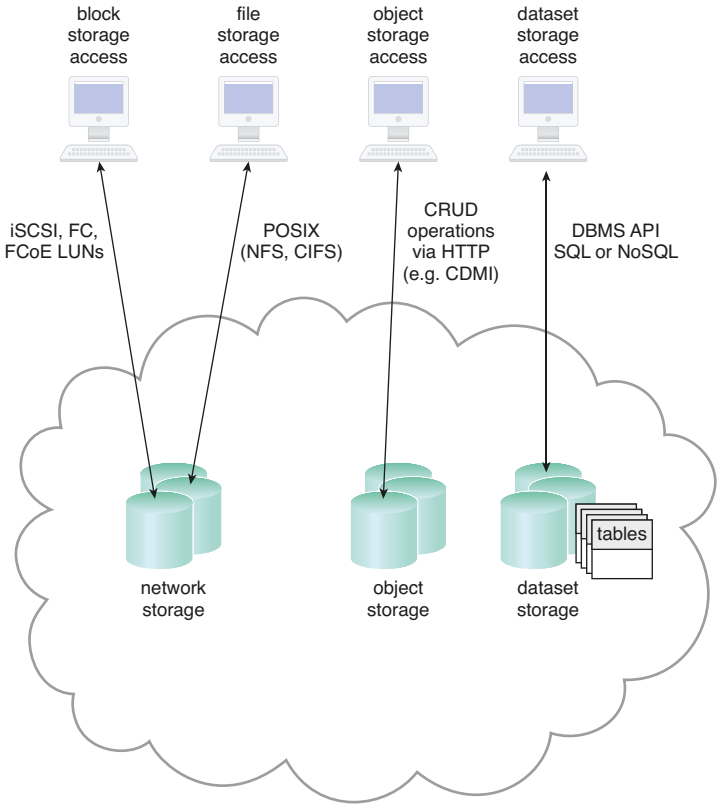


Figure 8.12

Different cloud service consumers utilize different technologies to interface with virtualized cloud storage devices. (Adapted from the CDMI Cloud Storage Reference Model.)

Object Storage Interfaces

Various types of data can be referenced and stored as web resources. This is referred to as object storage, which is based on technologies that can support a range of data and media types. Cloud storage device mechanisms that implement this interface can typically be accessed via REST or web service–based cloud services using HTTP as the prime protocol. The Storage Networking Industry Association's Cloud Data Management Interface (SNIA's CDMI) supports the use of object storage interfaces.

Database Storage Interfaces

Cloud storage device mechanisms based on database storage interfaces typically support a query language in addition to basic storage operations. Storage management is carried out using a standard API or an administrative user interface.

This classification of storage interface is divided into two main categories according to storage structure, as follows.

Relational Data Storage

Traditionally, many on-premises IT environments store data using relational databases or relational database management systems (RDBMSs). Relational databases (or relational storage devices) rely on tables to organize similar data into rows and columns. Tables can have relationships with each other to give the data increased structure, to protect data integrity, and to avoid data redundancy (which is referred to as data normalization). Working with relational storage commonly involves the use of the industry standard Structured Query Language (SQL).

A cloud storage device mechanism implemented using relational data storage could be based on any number of commercially available database products, such as IBM DB2, Oracle Database, Microsoft SQL Server, and MySQL.

Challenges with cloud-based relational databases commonly pertain to scaling and performance. Scaling a relational cloud storage device vertically can be more complex and cost-ineffective than horizontal scaling. Databases with complex relationships and/or containing large volumes of data can be afflicted with higher processing overhead and latency, especially when accessed remotely via cloud services.

Non-Relational Data Storage

Non-relational storage (also commonly referred to as *NoSQL* storage) moves away from the traditional relational database model in that it establishes a "looser" structure for stored data with less emphasis on defining relationships and realizing data normalization. The primary motivation for using non-relational storage is to avoid the potential complexity and processing overhead that can be imposed by relational databases. Also, non-relational storage can be more horizontally scalable than relational storage.

The trade-off with non-relational storage is that the data loses much of the native form and validation due to limited or primitive schemas or data models. Furthermore, non-relational repositories don't tend to support relational database functions, such as transactions or joins.

Normalized data exported into a non-relational storage repository will usually become denormalized, meaning that the size of the data will typically grow. An extent of normalization can be preserved, but usually not for complex relationships. Cloud providers often offer non-relational storage that provides scalability and availability of stored data over multiple server environments. However, many non-relational storage mechanisms are proprietary and therefore can severely limit data portability.

CASE STUDY EXAMPLE

DTGOV provides cloud consumers access to a cloud storage device based on an object storage interface. The cloud service that exposes this API offers basic functions on stored objects, such as search, create, delete, and update. The search function uses a hierarchical object arrangement that resembles a file system. DTGOV further offers a cloud service that is used exclusively with virtual servers and enables the creation of cloud storage devices via a block storage network interface. Both cloud services use APIs that are compliant with SNIA's CDMI v1.0.

The object-based cloud storage device has an underlying storage system with variable storage capacity, which is directly controlled by a software component that also exposes the interface. This software enables the creation of isolated cloud storage devices that are allocated to cloud consumers. The storage system uses a security credential management system to administer user-based access control to the device's data objects (Figure 8.13).

Access control is granted on a per-object basis and uses separate access policies for creating, reading from, and writing to each data object. Public access permissions are allowed, although they are read-only. Access groups are formed by nominated users that must be previously registered via the credential management system. Data objects can be accessed from both web applications and web service interfaces, which are implemented by the cloud storage software.

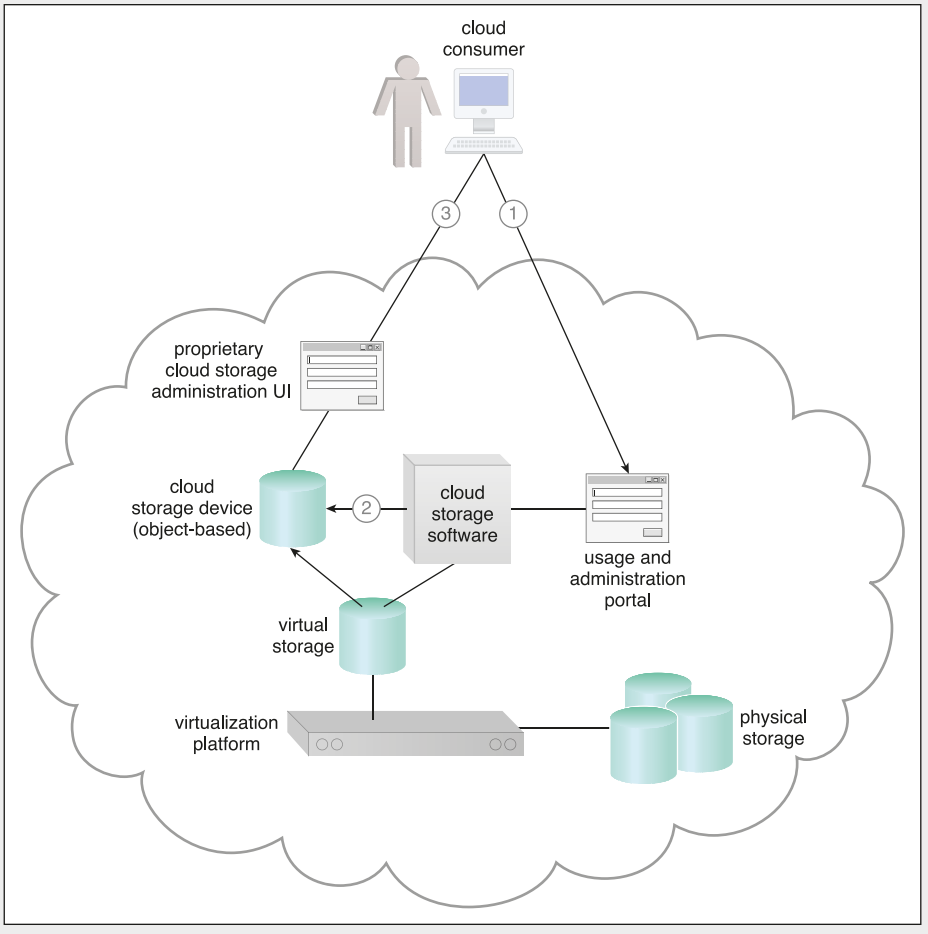


Figure 8.13

The cloud consumer interacts with the usage and administration portal to create a cloud storage device and define access control policies (1). The usage and administration portal interacts with the cloud storage software to create the cloud storage device instance and apply the required access policy to its data objects (2). Each data object is assigned to a cloud storage device, and all the data objects are stored in the same virtual storage volume. The cloud consumer uses the proprietary cloud storage device user interface to interact directly with the data objects (3). (Note that the usage and administration portal is explained in Chapter 12.)

The creation of the cloud consumers' block-based cloud storage devices is managed by the virtualization platform, which instantiates the LUN's implementation of the virtual storage (Figure 8.14). The cloud storage device (or the LUN) must be assigned

by the VIM to an existing virtual server before it can be used. The capacity of block-based cloud storage devices is expressed by 1 GB increments. It can be created as fixed storage that cloud consumers can modify administratively or as variable-size storage that has an initial 5 GB capacity, which automatically increases and decreases by 5 GB increments according to usage demands.

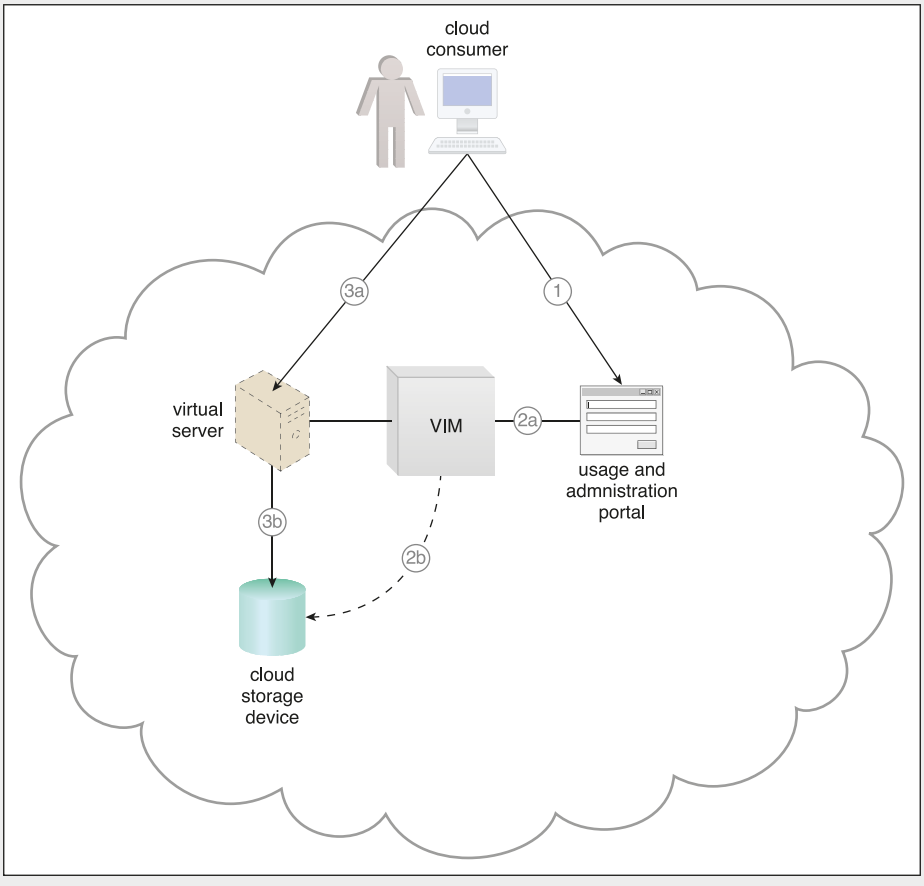


Figure 8.14

The cloud consumer uses the usage and administration portal to create and assign a cloud storage device to an existing virtual server (1). The usage and administration portal interacts with the VIM software (2a), which creates and configures the appropriate LUN (2b). Each cloud storage device uses a separate LUN controlled by the virtualization platform. The cloud consumer remotely logs into the virtual server directly (3a) to access the cloud storage device (3b).

8.5 Cloud Usage Monitor

The *cloud usage monitor* mechanism is a lightweight and autonomous software program responsible for collecting and processing IT resource usage data.

NOTE

This is a parent mechanism that represents a broad range of cloud usage monitors, several of which are established as specialized mechanisms in Chapter 9, and several more of which are described in the cloud architectural models covered in Part III of this book.

Depending on the type of usage metrics they are designed to collect and the manner in which usage data needs to be collected, cloud usage monitors can exist in different formats. The upcoming sections describe three common agent-based implementation formats. Each can be designed to forward collected usage data to a log database for post-processing and reporting purposes.

Monitoring Agent

A *monitoring agent* is an intermediary, event-driven program that exists as a service agent and resides along existing communication paths to transparently monitor and analyze dataflows (Figure 8.15). This type of cloud usage monitor is commonly used to measure network traffic and message metrics.

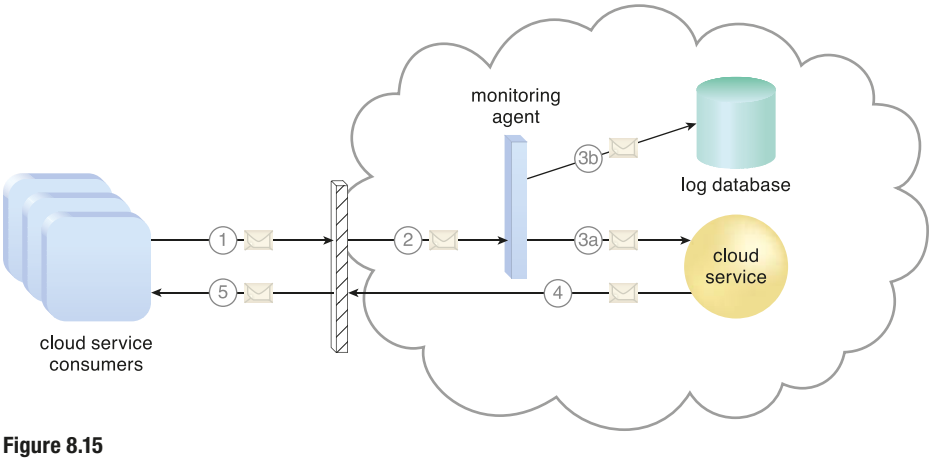


Figure 8.15

A cloud service consumer sends a request message to a cloud service (1). The monitoring agent intercepts the message to collect relevant usage data (2) before allowing it to continue to the cloud service (3a). The monitoring agent stores the collected usage data in a log database (3b). The cloud service replies with a response message (4) that is sent back to the cloud service consumer without being intercepted by the monitoring agent (5).

Resource Agent

A *resource agent* is a processing module that collects usage data by having event-driven interactions with specialized resource software (Figure 8.16). This module is used to monitor usage metrics based on predefined, observable events at the resource software level, such as initiating, suspending, resuming, and vertical scaling.

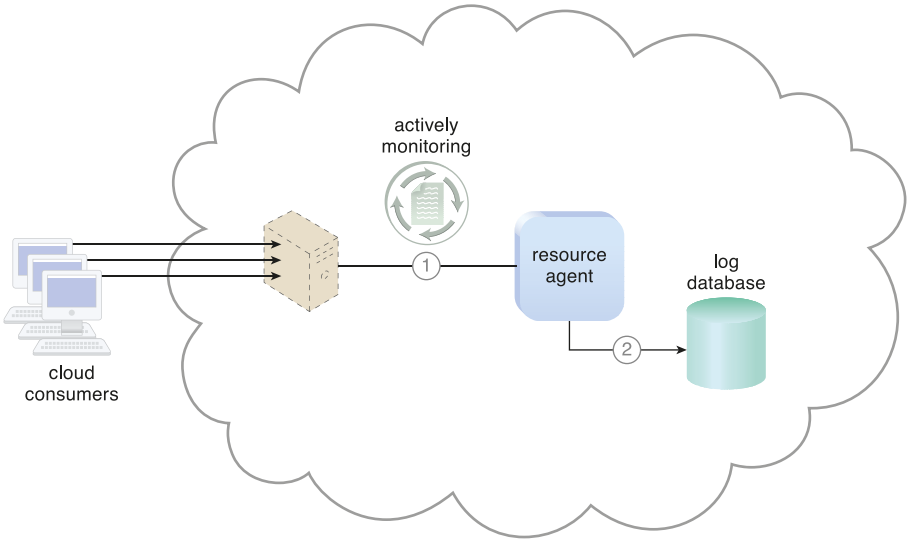


Figure 8.16

The resource agent is actively monitoring a virtual server and detects an increase in usage (1). The resource agent receives a notification from the underlying resource management program that the virtual server is being scaled up and stores the collected usage data in a log database, as per its monitoring metrics (2).

Polling Agent

A *polling agent* is a processing module that collects cloud service usage data by polling IT resources. This type of cloud service monitor is commonly used to periodically monitor IT resource status, such as uptime and downtime (Figure 8.17).

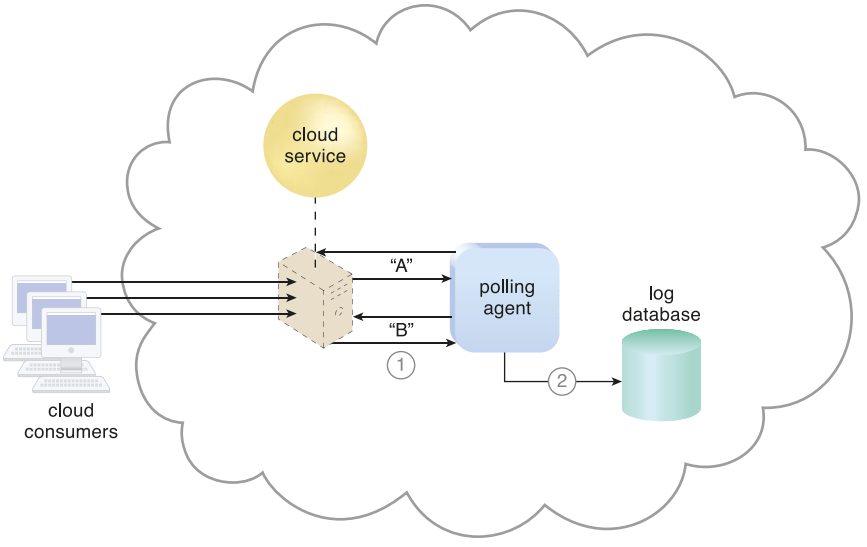


Figure 8.17

A polling agent monitors the status of a cloud service hosted by a virtual server by sending periodic polling request messages and receiving polling response messages that report usage status "A" after a number of polling cycles, until it receives a usage status of "B" (1), upon which the polling agent records the new usage status in the log database (2).

CASE STUDY EXAMPLE

One of the challenges encountered during DTGOV's cloud adoption initiative has been ensuring that collected usage data is accurate. The resource allocation methods of previous IT outsourcing models had resulted in clients being billed chargeback fees based on the number of physical servers that was listed in annual leasing contracts, regardless of actual usage.

DTGOV now needs to define a model that allows virtual servers of varying performance levels to be leased and billed hourly. Usage data needs to be measured at an extremely granular level to achieve the necessary degree of accuracy. DTGOV implements a resource agent that relies on the resource usage events generated by the VIM platform to calculate the virtual server usage data.

The resource agent is designed with logic and metrics that are based on the following rules:

1. Each resource usage event that is generated by the VIM software can contain the following data:
   - *Event Type (EV_TYPE)* – Generated by the VIM platform. There are five types of events:

     VM Starting (creation at the hypervisor)

     VM Started (completion of the boot procedure)

     VM Stopping (shutting down)

     VM Stopped (termination at the hypervisor)

     VM Scaled (change of performance parameters)
   - *VM Type (VM_TYPE)* – This represents a type of virtual server, as dictated by its performance parameters. A predefined list of possible virtual server configurations provides the parameters that are described by the metadata whenever a VM starts or scales.
   - *Unique VM Identifier (VM_ID)* – This identifier is provided by the VIM platform.
   - *Unique Cloud Consumer Identifier (CS_ID)* – Another identifier provided by the VIM platform to represent the cloud consumer.
   - *Event Timestamp (EV_T)* – An identification of an event occurrence that is expressed in date–time format, with the time zone of the data center and referenced to UTC as defined in RFC 3339 (as per the ISO 8601 profile).
2. Usage measurements are recorded for every virtual server that a cloud consumer creates.
3. Usage measurements are recorded for a measurement period whose length is defined by two timestamps called t_{start} and t_{end}. The start of the measurement period defaults to the beginning of the calendar month (t_{start} = 2012-12-01T00:00:00-08:00) and finishes at the end of the calendar month (t_{end} = 2012-12-31T23:59:59-08:00). Customized measurement periods are also supported.

4. Usage measurements are recorded at each minute of usage. The virtual server usage measurement period starts when the virtual server is created at the hypervisor and stops at its termination.
5. Virtual servers can be started, scaled, and stopped multiple times during the measurement period. The time interval between each occurrence i (i = 1, 2, 3, …) of these pairs of successive events that are declared for a virtual server is called a usage cycle that is known as T_{cycle_i}:
   - *VM_Starting, VM_Stopping* – VM size is unchanged at the end of the cycle
   - *VM_Starting, VM_Scaled* – VM size has changed at the end of the cycle
   - *VM_Scaled, VM_Scaled* – VM size has changed while scaling, at the end of the cycle
   - *VM_Scaled, VM_Stopping* – VM size has changed at the end of the cycle
6. The total usage, U_{total}, for each virtual server during the measurement period is calculated using the following resource usage event log database equations:
   - For each VM_TYPE and VM_ID in the log database: $U_{\text{total\_VM\_type\_j}} = \sum_{t_{start}}^{t_{end}} T_{cycle_i}$
   - As per the total usage time that is measured for each VM_TYPE, the vector of usage for each VM_ID is U_{total}: U_{total} = {type 1, $U_{total\_VM\_type\_1}$, type 2, $U_{total\_VM\_type\_2}$, …}

Figure 8.18 depicts the resource agent interacting with the VIM's event-driven API.

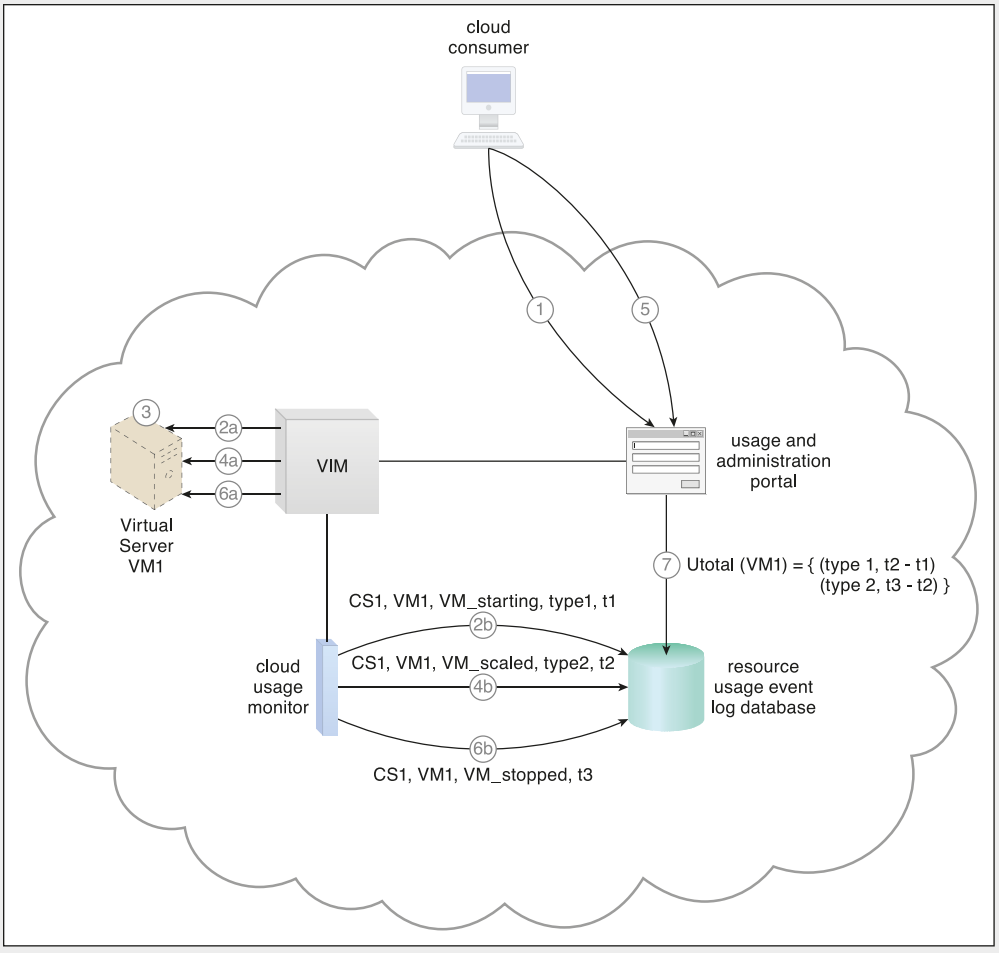


Figure 8.18

The cloud consumer (CS_ID = CS1) requests the creation of a virtual server (VM_ID = VM1) of configuration size type 1 (VM_TYPE = type1) (1). The VIM creates the virtual server (2a). The VIM's event-driven API generates a resource usage event with timestamp = t1, which the cloud usage monitor software agent captures and records in the resource usage event log database (2b). Virtual server usage increases and reaches the auto-scaling threshold (3). The VIM scales up Virtual Server VM1 (4a) from configuration type 1 to type 2 (VM_TYPE = type2). The VIM's event-driven API generates a resource usage event with timestamp = t2, which is captured and recorded in the resource usage event log database by the cloud usage monitor software agent (4b). The cloud consumer shuts down the virtual server (5). The VIM stops Virtual Server VM1 (6a) and its event-driven API generates a resource usage event with timestamp = t3, which the cloud usage monitor software agent captures and records in the log database (6b). The usage and administration portal accesses the log database and calculates the total usage (Utotal) for Virtual Server Utotal VM1 (7).

8.6 Resource Replication

Defined as the creation of multiple instances of the same IT resource, replication is typically performed when an IT resource's availability and performance need to be enhanced. Virtualization technology is used to implement the *resource replication* mechanism to replicate cloud-based IT resources (Figure 8.19).

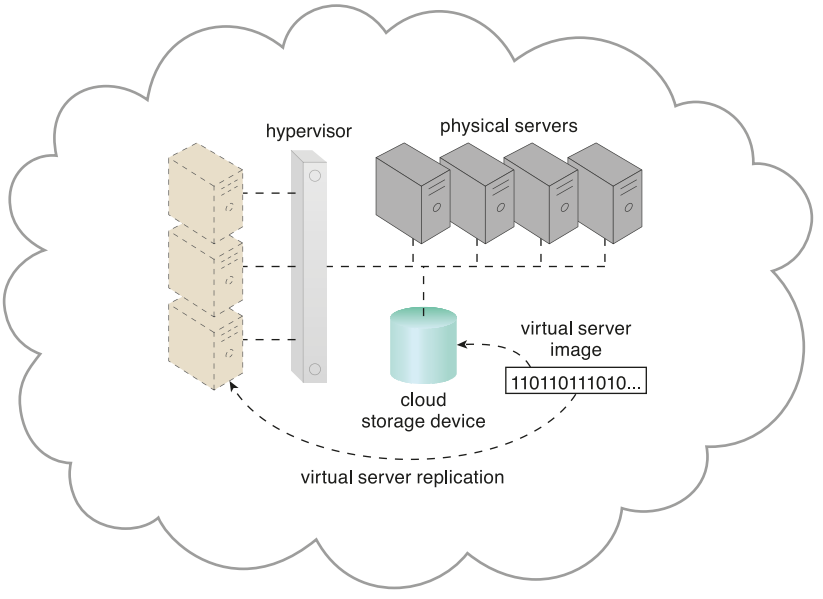


Figure 8.19
The hypervisor replicates several instances of a virtual server, using a stored virtual server image.

NOTE

This is a parent mechanism that represents different types of software programs capable of replicating IT resources. The most common example is the hypervisor mechanism described in this chapter. For example, the virtualization platform's hypervisor can access a virtual server image to create several instances, or to deploy and replicate ready-made environments and entire applications. Other common types of replicated IT resources include cloud service implementations and various forms of data and cloud storage device replication.

CASE STUDY EXAMPLE

DTGOV establishes a set of high-availability virtual servers that can be automatically relocated to physical servers running in different data centers in response to severe failure conditions. This is illustrated in the scenario depicted in Figures 8.20 to 8.22, where a virtual server that resides on a physical server running at one data center experiences a failure condition. VIMs from different data centers coordinate to overcome the unavailability by reallocating the virtual server to a different physical server running in another data center.

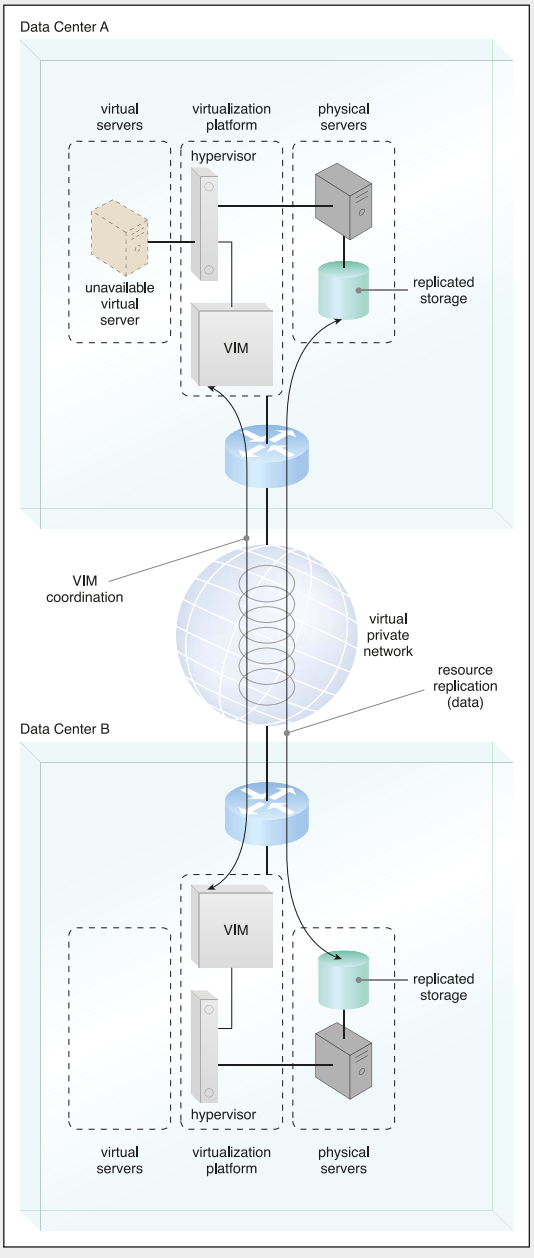


Figure 8.20

A high-availability virtual server is running in Data Center A. VIM instances in Data Centers A and B are executing a coordination function that allows detection of failure conditions. Stored VM images are replicated between data centers as a result of the high-availability architecture.

Figure 8.21
The virtual server becomes unavailable in Data Center A. The VIM in Data Center B detects the failure condition and starts to reallocate the high-availability server from Data Center A to Data Center B.

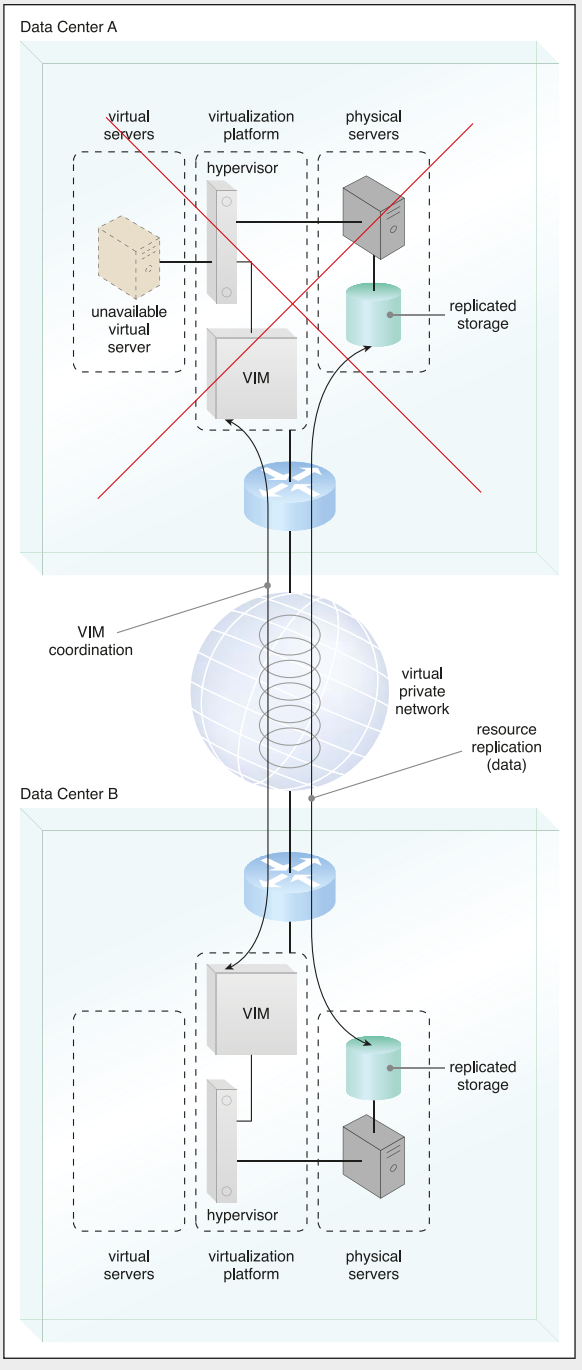
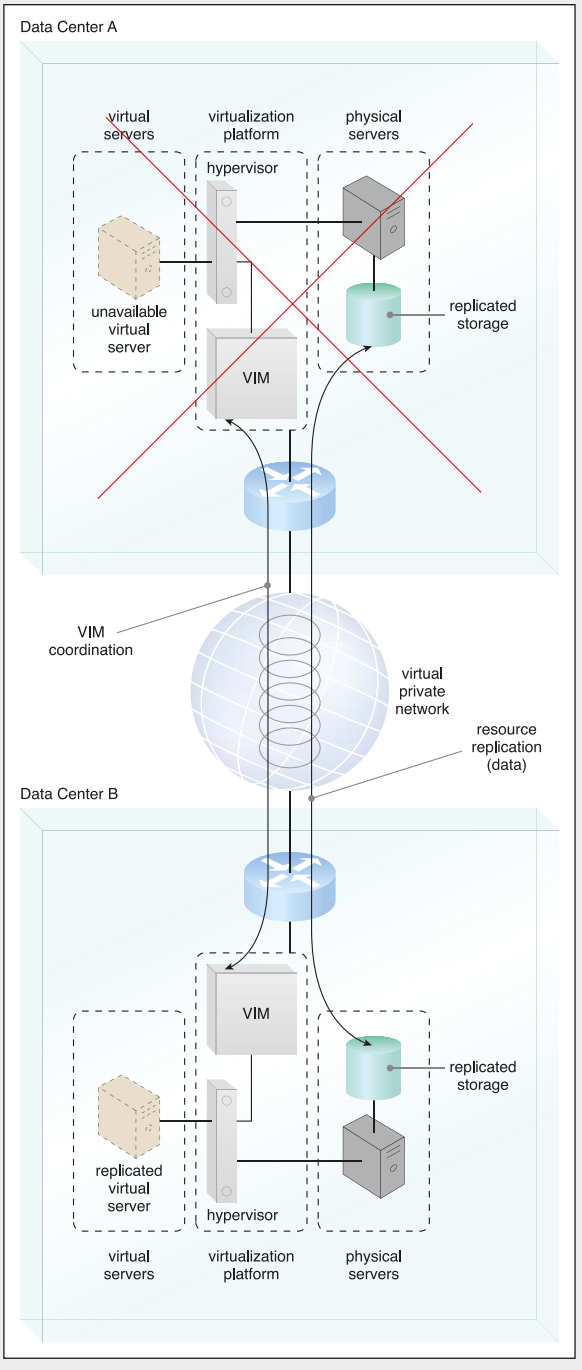


Figure 8.22
A new instance of the virtual server is created and made available in Data Center B.



8.7 Ready-Made Environment

The *ready-made environment* mechanism (Figure 8.23) is a defining component of the PaaS cloud delivery model that represents a predefined, cloud-based platform comprised of a set of already installed IT resources, ready to be used and customized by a cloud consumer. These environments are utilized by cloud consumers to remotely develop and deploy their own services and applications within a cloud. Typical ready-made environments include preinstalled IT resources, such as databases, middleware, development tools, and governance tools.

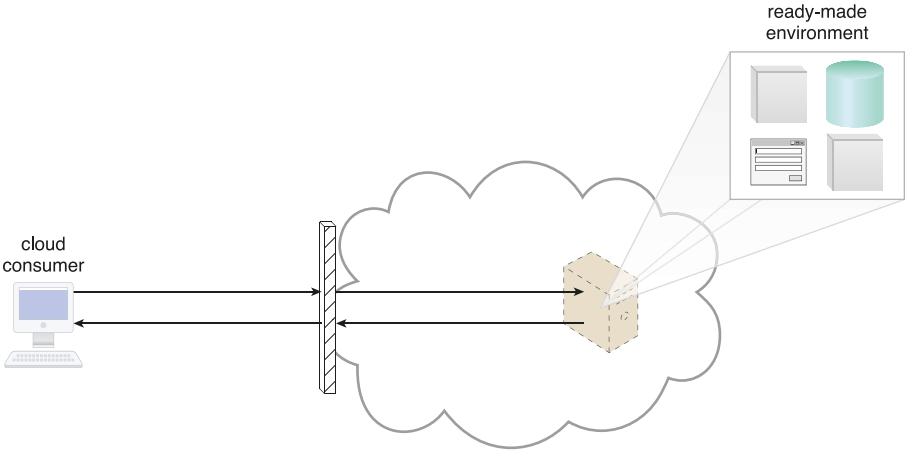


Figure 8.23
A cloud consumer accesses a ready-made environment hosted on a virtual server.

A ready-made environment is generally equipped with a complete software development kit (SDK) that provides cloud consumers with programmatic access to the development technologies that comprise their preferred programming stacks.

Middleware is available for multitenant platforms to support the development and deployment of web applications. Some cloud providers offer runtime execution environments for cloud services that are based on different runtime performance and billing parameters. For example, a front-end instance of a cloud service can be configured to respond to time-sensitive requests more effectively than a back-end instance. The former variation will be billed at a different rate than the latter.

As further demonstrated in the upcoming case study example, a solution can be partitioned into groups of logic that can be designated for both front-end and back-end instance invocation so as to optimize runtime execution and billing.

CASE STUDY EXAMPLE

ATN developed and deployed several noncritical business applications using a leased PaaS environment. One was a Java-based Part Number Catalog web application used for the switches and routers they manufacture. This application is used by different factories, but it does not manipulate transaction data, which is instead processed by a separate stock control system.

The application logic was split into front-end and back-end processing logic. The front-end logic was used to process simple queries and updates to the catalog. The back-end part contains the logic required to render the complete catalog and correlate similar components and legacy part numbers.

Figure 8.24 illustrates the development and deployment environment for ATN's Part Number Catalog application. Note how the cloud consumer assumes both the developer and end-user roles.

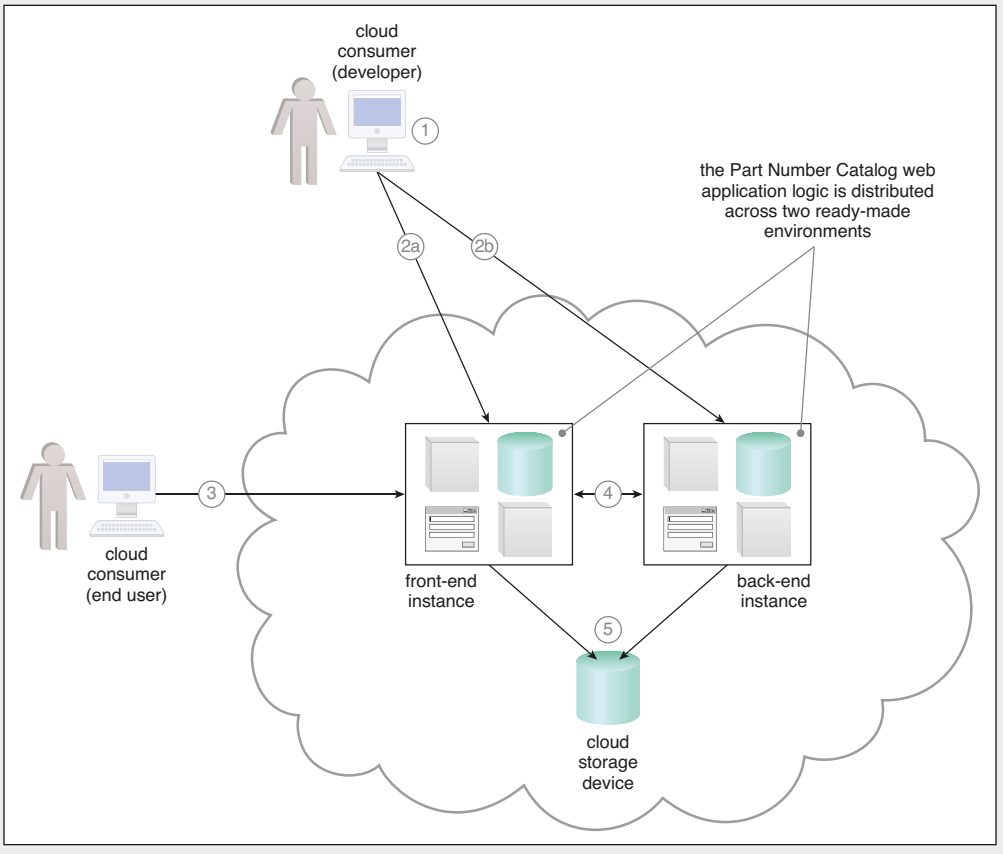


Figure 8.24

The developer uses the provided SDK to develop the Part Number Catalog web application (1). The application software is deployed on a web platform that was established by two ready-made environments called the front-end instance (2a) and the back-end instance (2b). The application is made available for usage, and one end user accesses its front-end instance (3). The software running in the front-end instance invokes a long-running task at the back-end instance that corresponds to the processing required by the end user (4). The application software deployed at both the front-end and back-end instances is backed by a cloud storage device that provides persistent storage of the application data (5).

8.8 Container

Containers can provide an effective means of deploying and delivering cloud services. Containerization technology is explained in Chapter 6.

Chapter 9

Specialized Cloud Mechanisms

A typical cloud technology architecture contains numerous moving parts to address distinct usage requirements of IT resources and solutions. Each mechanism covered in this chapter fulfills a specific runtime function in support of one or more cloud characteristics.

The following specialized cloud mechanisms are described in this chapter:

- Automated Scaling Listener
- Load Balancer
- SLA Monitor
- Pay-Per-Use Monitor
- Audit Monitor
- Failover System
- Resource Cluster
- Multi-Device Broker
- State Management Database

All of these mechanisms can be considered extensions to cloud infrastructure and can be combined in numerous ways as part of distinct and custom technology architectures, many examples of which are provided in Part III of this book.

9.1 Automated Scaling Listener

The *automated scaling listener* mechanism is a service agent that monitors and tracks communications between cloud service consumers and cloud services for dynamic scaling purposes. Automated scaling listeners are deployed within the cloud, typically near the firewall, from where they automatically track workload status information. Workloads can be determined by the volume of cloud consumer–generated requests or via back-end processing demands triggered by certain types of requests. For example, a small amount of incoming data can result in a large amount of processing.

Automated scaling listeners can provide different types of responses to workload fluctuation conditions, such as:

- Automatically scaling IT resources out or in based on parameters previously defined by the cloud consumer (commonly referred to as *auto-scaling*).
- Automatic notification of the cloud consumer when workloads exceed current thresholds or fall below allocated resources (Figure 9.1). This way, the cloud consumer can choose to adjust its current IT resource allocation.

Different cloud provider vendors have different names for service agents that act as automated scaling listeners.

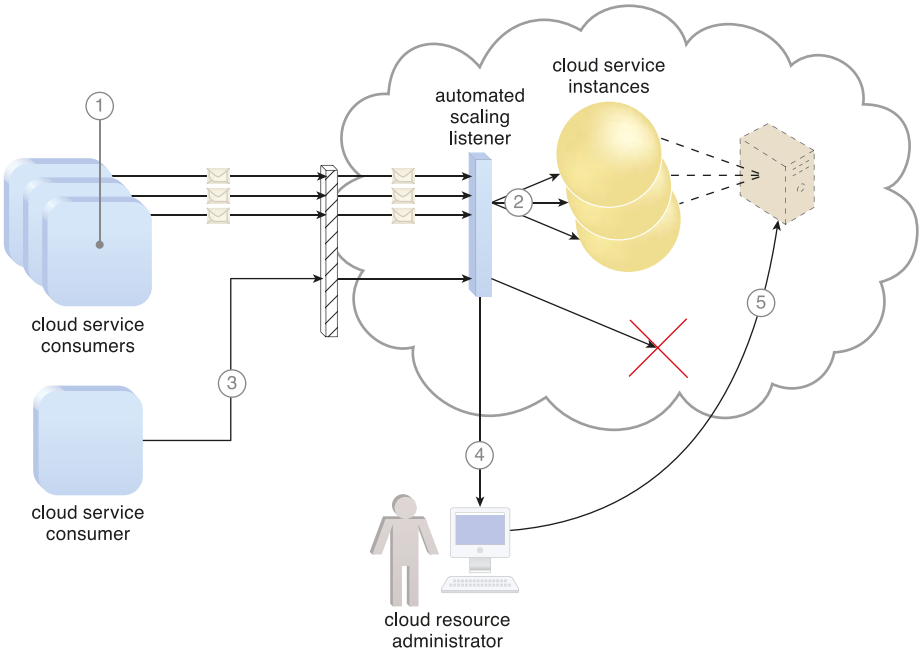


Figure 9.1

Three cloud service consumers attempt to access one cloud service simultaneously (1). The automated scaling listener scales out and initiates the creation of three redundant instances of the service (2). A fourth cloud service consumer attempts to use the cloud service (3). Programmed to allow up to only three instances of the cloud service, the automated scaling listener rejects the fourth attempt and notifies the cloud consumer that the requested workload limit has been exceeded (4). The cloud consumer's cloud resource administrator accesses the remote administration environment to adjust the provisioning setup and increase the redundant instance limit (5).

CASE STUDY EXAMPLE

NOTE

This case study example makes reference to the live VM migration component, which is introduced in the *Hypervisor Clustering Architecture* section in Chapter 14, and further described and demonstrated in subsequent architecture scenarios.

DTGOV's physical servers vertically scale virtual server instances, starting with the smallest virtual machine configuration (1 virtual processor core, 4 GB of virtual RAM) to the largest (128 virtual processor cores, 512 GB of virtual RAM). The virtualization platform is configured to automatically scale a virtual server at runtime, as follows:

- *Scaling Down* – The virtual server continues residing on the same physical host server while being scaled down to a lower performance configuration.
- *Scaling Up* – The virtual server's capacity is doubled on its original physical host server. The VIM may also live migrate the virtual server to another physical server if the original host server is overcommitted. Migration is automatically performed at runtime and does not require the virtual server to shut down.

Auto-scaling settings controlled by cloud consumers determine the runtime behavior of automated scaling listener agents, which run on the hypervisor that monitors the resource usage of the virtual servers. For example, one cloud consumer has it set up so that whenever resource usage exceeds 80% of a virtual server's capacity for 60 consecutive seconds, the automated scaling listener triggers the scaling-up process by sending the VIM platform a scale-up command. Conversely, the automated scaling listener also commands the VIM to scale down whenever resource usage dips 15% below capacity for 60 consecutive seconds (Figure 9.2).

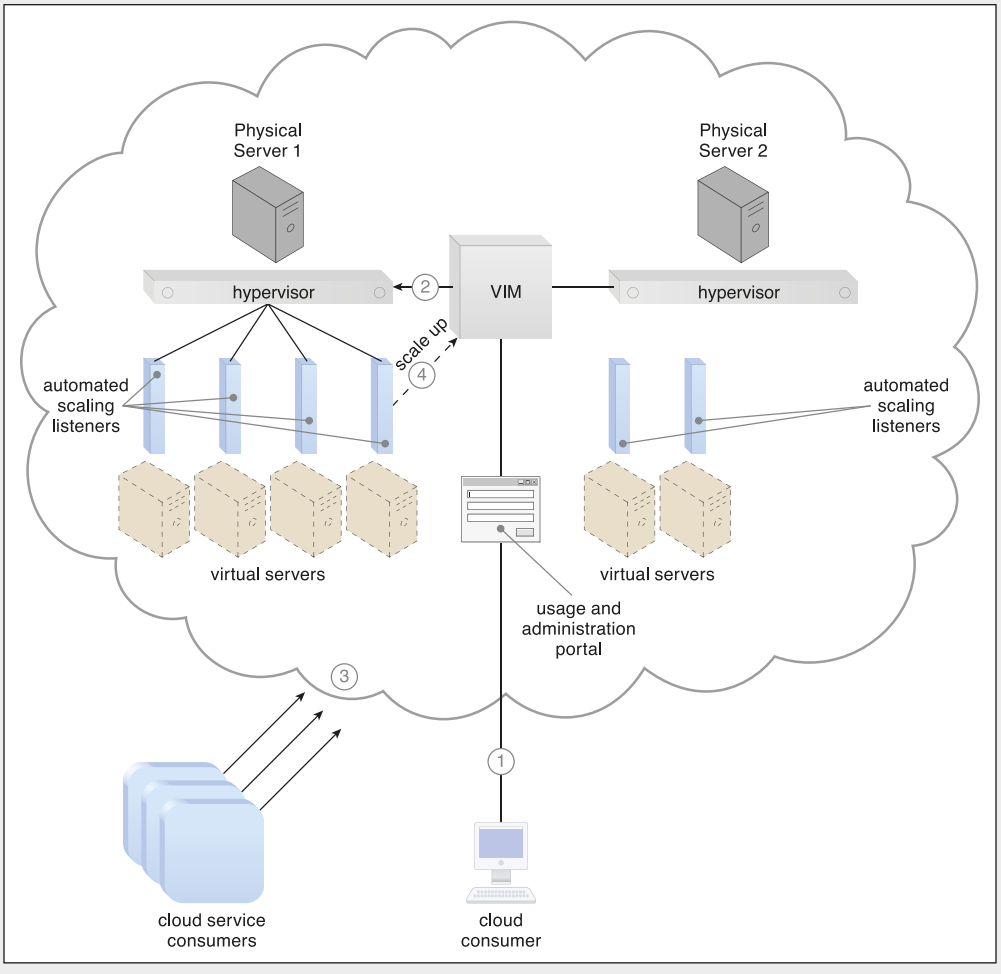


Figure 9.2

A cloud consumer creates and starts a virtual server with 8 virtual processor cores and 16 GB of virtual RAM (1). The VIM creates the virtual server at the cloud service consumer's request and allocates it to Physical Server 1 to join 3 other active virtual servers (2). Cloud consumer demand causes the virtual server usage to increase by over 80% of the CPU capacity for 60 consecutive seconds (3). The automated scaling listener running at the hypervisor detects the need to scale up and commands the VIM accordingly (4).

Figure 9.3 illustrates the live migration of a virtual machine, as performed by the VIM.

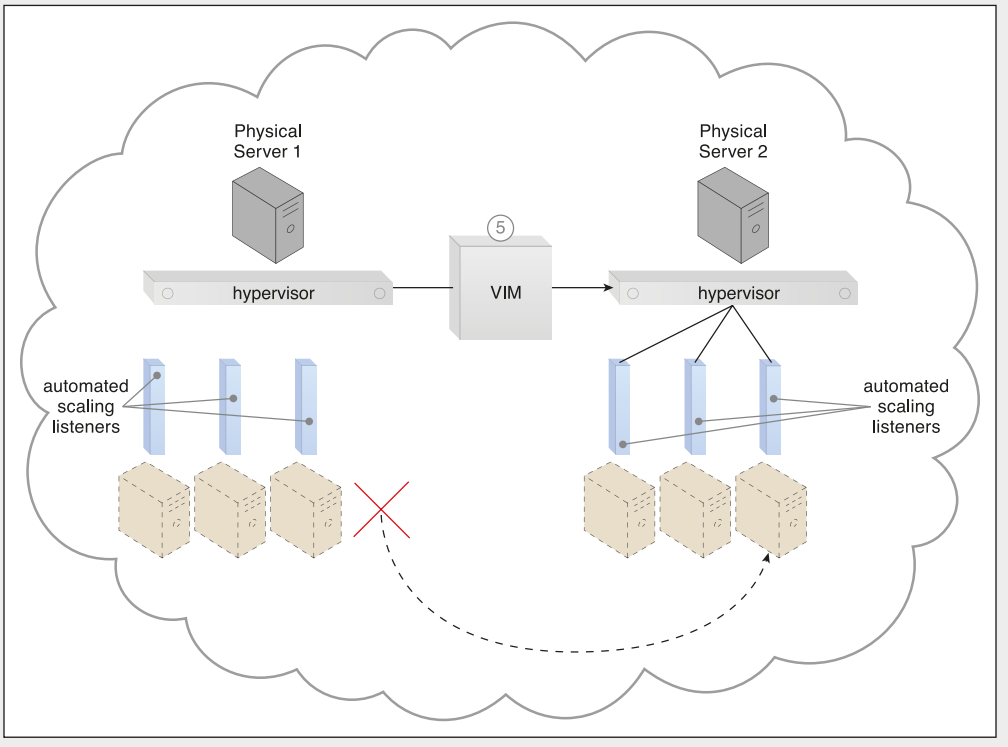


Figure 9.3

The VIM determines that scaling up the virtual server on Physical Server 1 is not possible and proceeds to live migrate it to Physical Server 2.

The scaling down of the virtual server by the VIM is depicted in Figure 9.4.

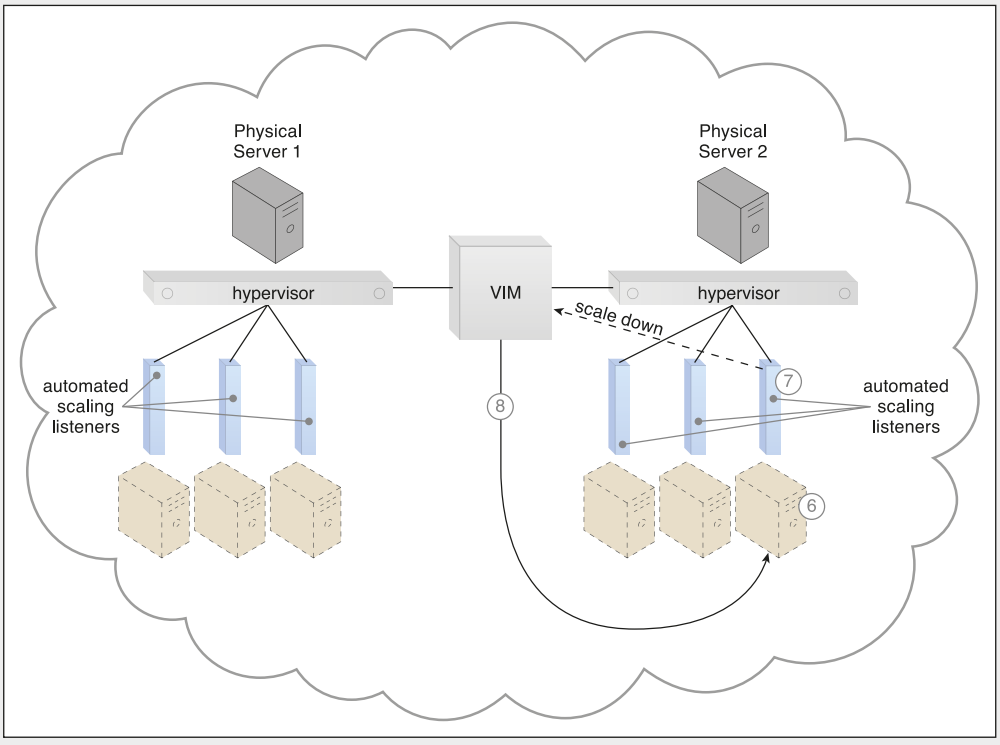


Figure 9.4

The virtual server's CPU/RAM usage remains below 15% capacity for 60 consecutive seconds (6). The automated scaling listener detects the need to scale down and commands the VIM (7), which scales down the virtual server (8) while it remains active on Physical Server 2.

9.2 Load Balancer

A common approach to horizontal scaling is to balance a workload across two or more IT resources to increase performance and capacity beyond what a single IT resource can provide. The *load balancer* mechanism is a runtime agent with logic fundamentally based on this premise.

Beyond simple division of labor algorithms (Figure 9.5), load balancers can perform a range of specialized runtime workload distribution functions that include:

- *Asymmetric Distribution* – larger workloads are issued to IT resources with higher processing capacities
- *Workload Prioritization* – workloads are scheduled, queued, discarded, and distributed according to their priority levels
- *Content-Aware Distribution* – requests are distributed to different IT resources as dictated by the request content

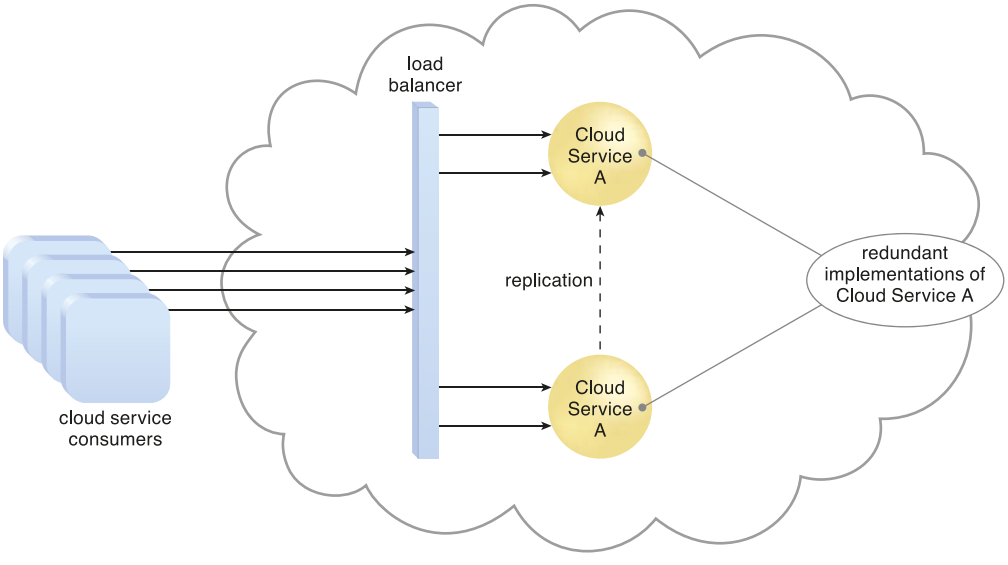


Figure 9.5

A load balancer implemented as a service agent transparently distributes incoming workload request messages across two redundant cloud service implementations, which in turn maximizes performance for the cloud service consumers.

A load balancer is programmed or configured with a set of performance and QoS rules and parameters with the general objectives of optimizing IT resource usage, avoiding overloads, and maximizing throughput.

The load balancer mechanisms can exist as a:

- multi-layer network switch
- dedicated hardware appliance
- dedicated software-based system (common in server operating systems)
- service agent (usually controlled by cloud management software)

The load balancer is typically located on the communication path between the IT resources generating the workload and the IT resources performing the workload processing. This mechanism can be designed as a transparent agent that remains hidden from the cloud service consumers, or as a proxy component that abstracts the IT resources performing their workload.

CASE STUDY EXAMPLE

The ATN Part Number Catalog cloud service does not manipulate transaction data even though it is used by multiple factories in different regions. It has peak usage periods during the first few days of every month that coincide with the preparatory processing of heavy stock control routines at the factories. ATN followed their cloud provider's recommendations and upgraded the cloud service to be highly scalable to support the anticipated workload fluctuations.

After developing the necessary upgrades, ATN decides to test the scalability by using a robot automation testing tool that simulates heavy workloads. The tests need to determine whether the application can seamlessly scale to serve peak workloads that are 1,000 times greater than their average workloads. The robots proceed to simulate workloads that last 10 minutes.

The application's resulting auto-scaling functionality is demonstrated in Figure 9.6.

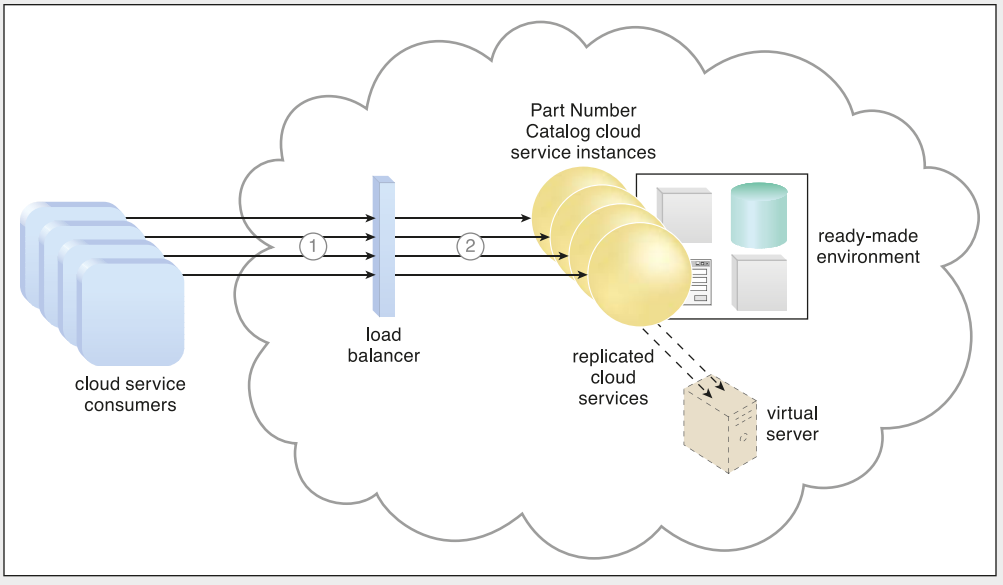


Figure 9.6
New instances of the cloud services are automatically created to meet increasing usage requests. The load balancer uses round-robin scheduling to ensure that the traffic is distributed evenly among the active cloud services.

9.3 SLA Monitor

The *SLA monitor* mechanism is used to specifically observe the runtime performance of cloud services to ensure that they are fulfilling the contractual QoS requirements that are published in SLAs (Figure 9.7). The data collected by the SLA monitor is processed by an SLA management system to be aggregated into SLA reporting metrics. The system can proactively repair or failover cloud services when exception conditions occur, such as when the SLA monitor reports a cloud service as "down."

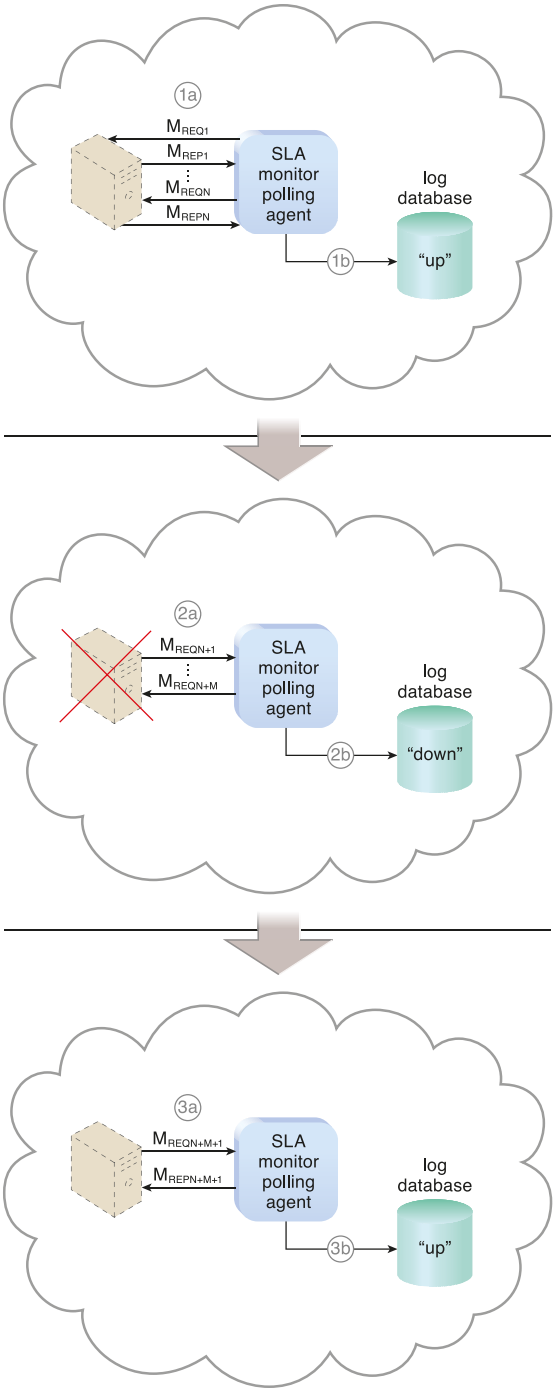
The SLA management system mechanism is discussed in Chapter 12.

Figure 9.7

The SLA monitor polls the cloud service by sending over polling request messages (M_{REQ1} to M_{REQN}). The monitor receives polling response messages (M_{REP1} to M_{REPN}) that report that the service was "up" at each polling cycle (1a). The SLA monitor stores the "up" time—the time period of all polling cycles 1 to N—in the log database (1b).

The SLA monitor polls the cloud service that sends polling request messages (M_{REQN+1} to M_{REQN+M}). Polling response messages are not received (2a). The response messages continue to time out, so the SLA monitor stores the "down" time—the time period of all polling cycles N+1 to N+M—in the log database (2b).

The SLA monitor sends a polling request message ($M_{REQN+M+1}$) and receives the polling response message ($M_{REPN+M+1}$) (3a). The SLA monitor stores the "up" time in the log database (3b).



CASE STUDY EXAMPLE

The standard SLA for virtual servers in DTGOV's leasing agreements defines a minimum IT resource availability of 99.95%, which is tracked using two SLA monitors: one based on a polling agent, and the other based on a regular monitoring agent implementation.

SLA Monitor Polling Agent

DTGOV's polling SLA monitor runs in the external perimeter network to detect physical server timeouts. It is able to identify data center network, hardware, and software failures (with minute granularity) that result in physical server nonresponsiveness. Three consecutive timeouts of 20-second polling periods are required to declare IT resource unavailability.

Three types of events are generated:

- *PS_Timeout* – the physical server polling has timed out
- *PS_Unreachable* – the physical server polling has consecutively timed out three times
- *PS_Reachable* – the previously unavailable physical server becomes responsive to polling again

SLA Monitoring Agent

The VIM's event-driven API implements the SLA monitor as a monitoring agent to generate the following three events:

- *VM_Unreachable* – the VIM cannot reach the VM
- *VM_Failure* – the VM has failed and is unavailable
- *VM_Reachable* – the VM is reachable

The events generated by the polling agent have timestamps that are logged into an SLA event log database and used by the SLA management system to calculate IT resource availability. Complex rules are used to correlate events from different polling SLA monitors and the affected virtual servers, and to discard any false positives for periods of unavailability.

Figures 9.8 and 9.9 show the steps taken by SLA monitors during a data center network failure and recovery.

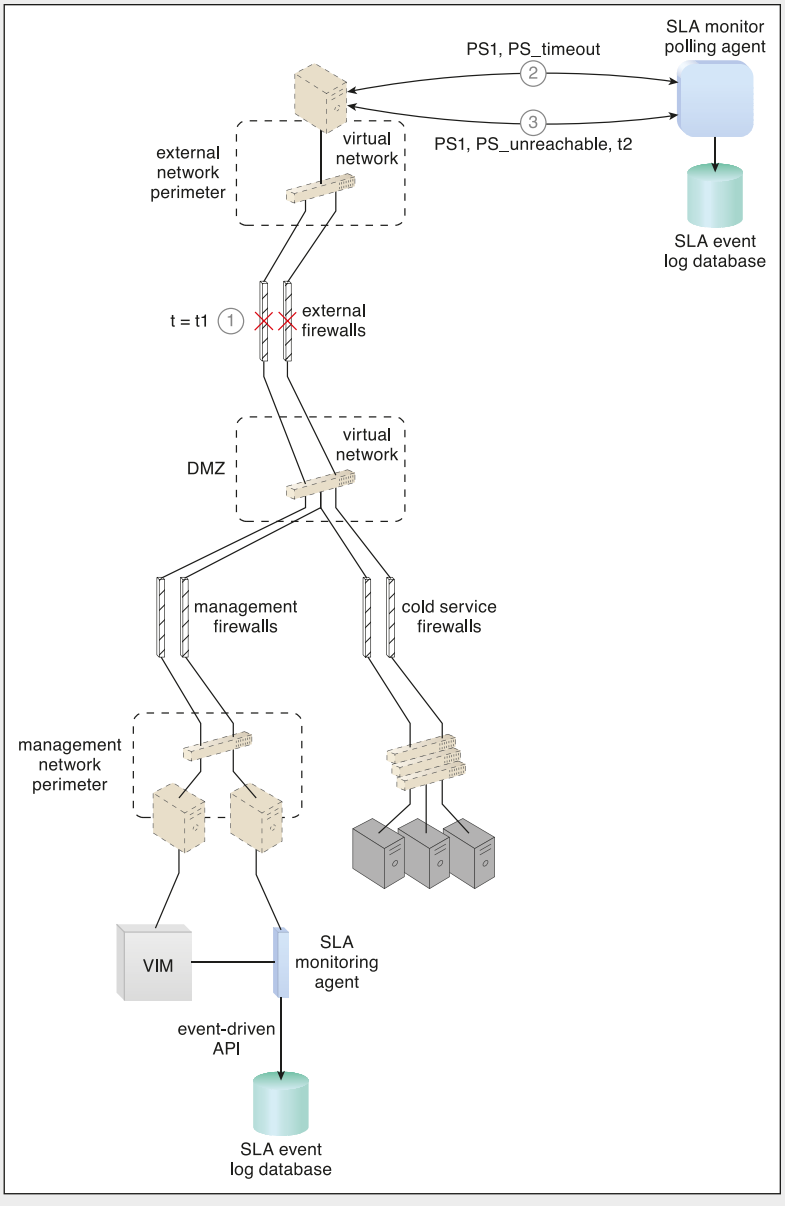


Figure 9.8

At timestamp = t1, a firewall cluster has failed and all the IT resources in the data center become unavailable (1). The SLA monitor polling agent stops receiving responses from physical servers and starts to issue PS_timeout events (2). The SLA monitor polling agent starts issuing PS_unreachable events after three successive PS_timeout events. The timestamp is now t2 (3).

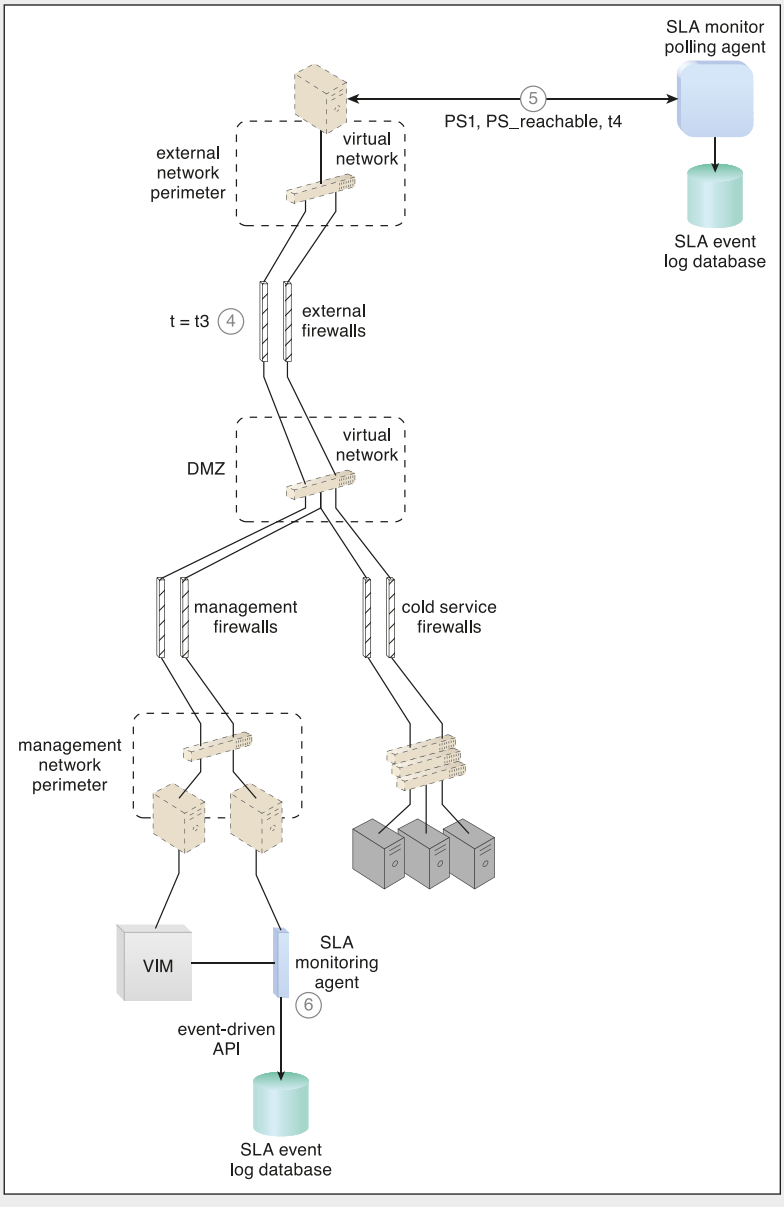


Figure 9.9

The IT resource becomes operational at timestamp = t3 (4). The SLA monitor polling agent receives responses from the physical servers and issues PS_reachable events. The timestamp is now t4 (5). The SLA monitoring agent did not detect any unavailability since the communication between the VIM platform and physical servers was not affected by the failure (6).

The SLA management system uses the information stored in the log database to calculate the period of unavailability as t4 – t3, which affected all the virtual servers in the data center.

Figures 9.10 and 9.11 illustrate the steps that are taken by the SLA monitors during the failure and subsequent recovery of a physical server that is hosting three virtual servers (VM1, VM2, VM3).

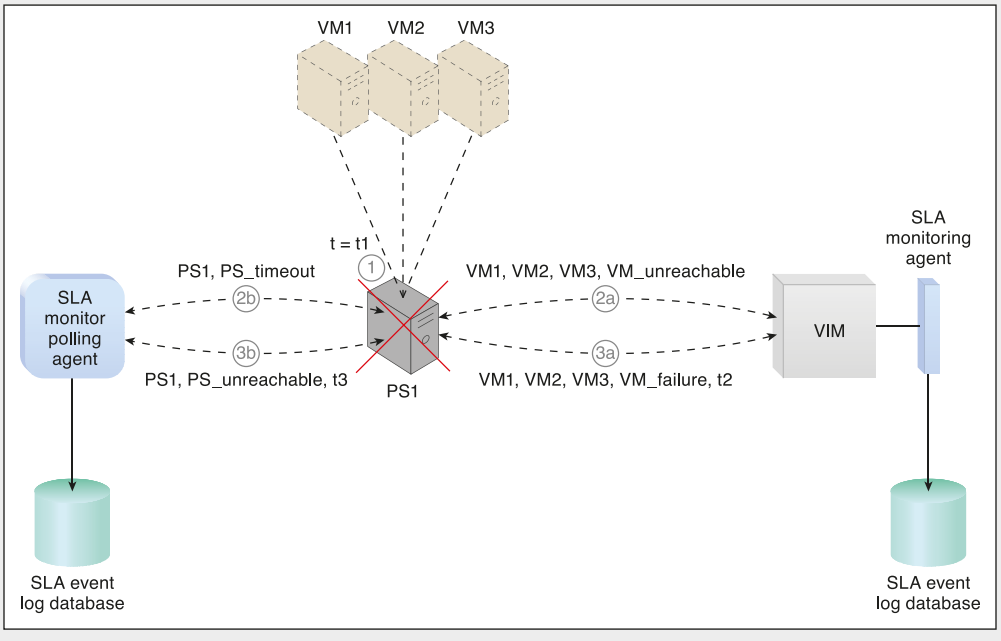


Figure 9.10

At timestamp = t1, the physical host server has failed and becomes unavailable (1). The SLA monitoring agent captures a VM_unreachable event that is generated for each virtual server in the failed host server (2a). The SLA monitor polling agent stops receiving responses from the host server and issues PS_timeout events (2b). At timestamp = t2, the SLA monitoring agent captures a VM_failure event that is generated for each of the failed host server's three virtual servers (3a). The SLA monitor polling agent starts to issue PS_unavailable events after three successive PS_timeout events at timestamp = t3 (3b).

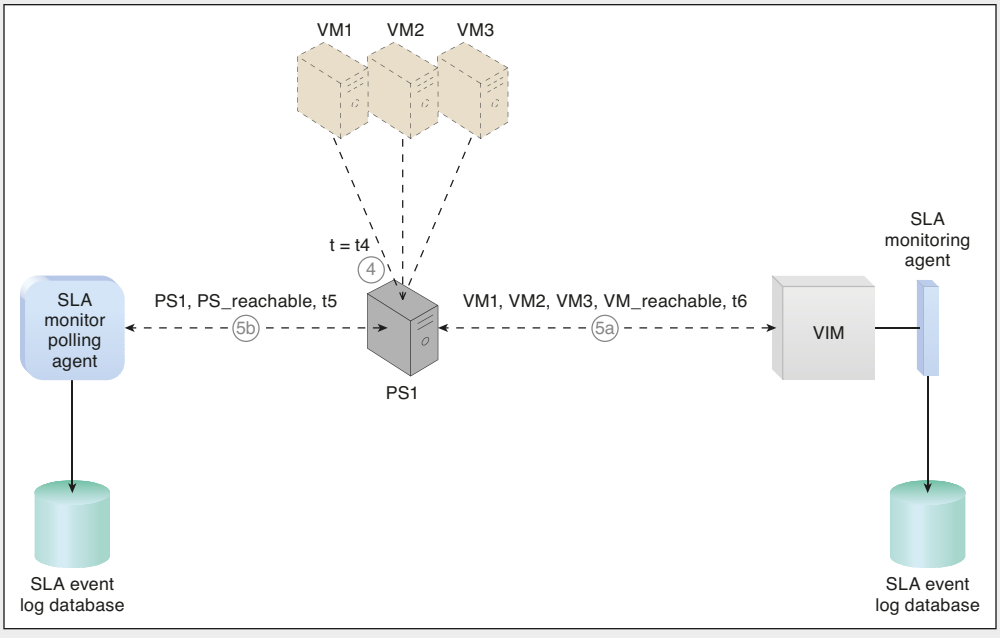


Figure 9.11

The host server becomes operational at timestamp = t4 (4). The SLA monitor polling agent receives responses from the physical server and issues PS_reachable events at timestamp = t5 (5a). At timestamp = t6, the SLA monitoring agent captures a VM_reachable event that is generated for each virtual server (5b). The SLA management system calculates the unavailability period that affected all the virtual servers as t6 – t2.

9.4 Pay-Per-Use Monitor

The *pay-per-use monitor* mechanism measures cloud-based IT resource usage in accordance with predefined pricing parameters and generates usage logs for fee calculations and billing purposes.

Some typical monitoring variables are:

- request/response message quantity
- transmitted data volume
- bandwidth consumption

The data collected by the pay-per-use monitor is processed by a billing management system that calculates the payment fees. The billing management system mechanism is covered in Chapter 12.

Figure 9.12 shows a pay-per-use monitor implemented as a resource agent used to determine the usage period of a virtual server.

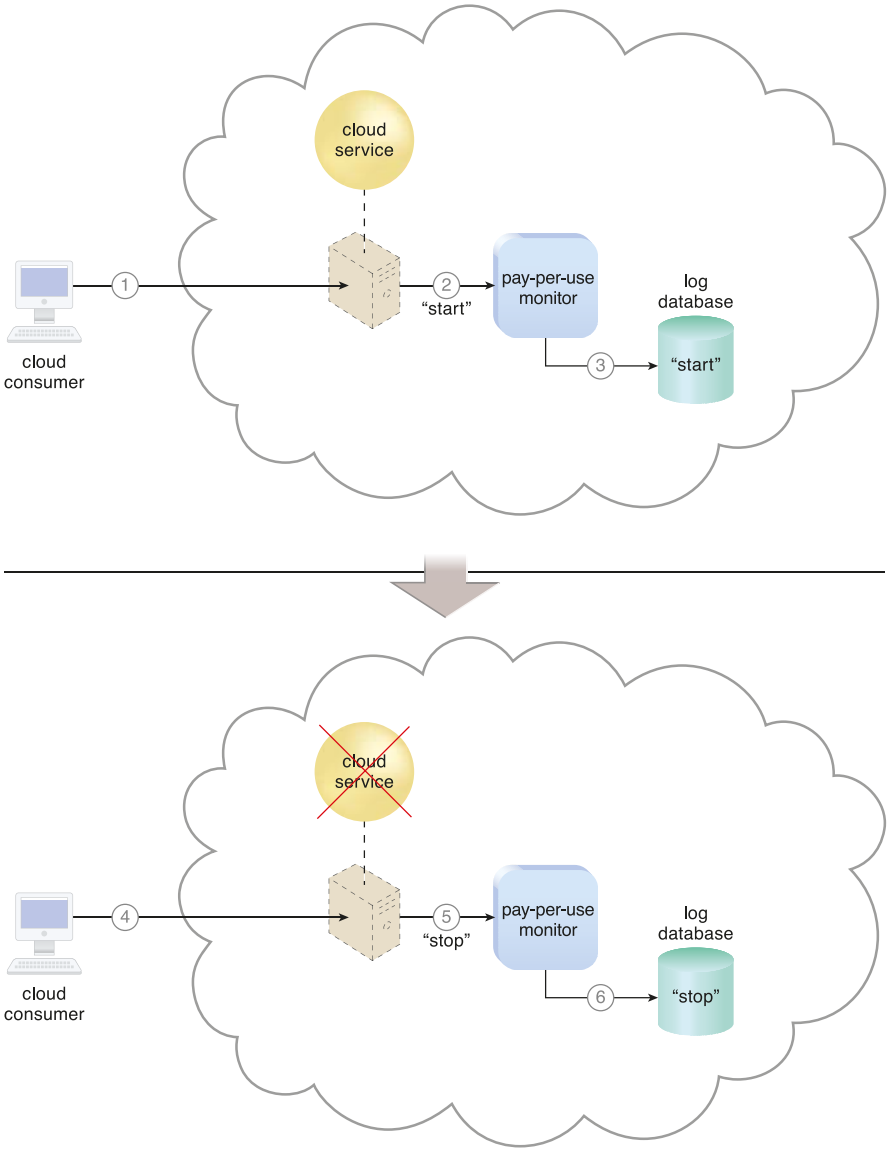


Figure 9.12

A cloud consumer requests the creation of a new instance of a cloud service (1). The IT resource is instantiated and the pay-per-use monitor receives a "start" event notification from the resource software (2). The pay-per-use monitor stores the value timestamp in the log database (3). The cloud consumer later requests that the cloud service instance be stopped (4). The pay-per-use monitor receives a "stop" event notification from the resource software (5) and stores the value timestamp in the log database (6).

Figure 9.13 illustrates a pay-per-use monitor designed as a monitoring agent that transparently intercepts and analyzes runtime communication with a cloud service.

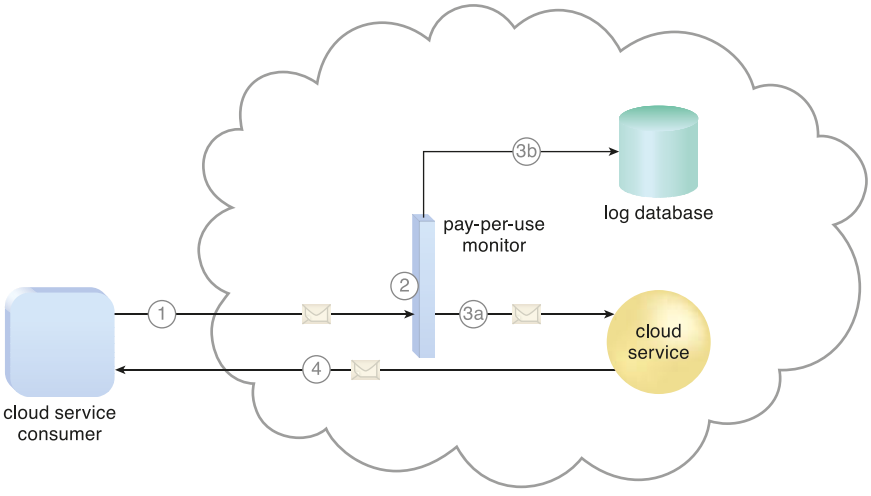


Figure 9.13

A cloud service consumer sends a request message to the cloud service (1). The pay-per-use monitor intercepts the message (2), forwards it to the cloud service (3a), and stores the usage information in accordance with its monitoring metrics (3b). The cloud service forwards the response messages back to the cloud service consumer to provide the requested service (4).

CASE STUDY EXAMPLE

DTGOV decides to invest in a commercial system capable of generating invoices based on events predefined as "billable" and customizable pricing models. The installation of the system results in two proprietary databases: the billing event database and the pricing scheme database.

Runtime events are collected via cloud usage monitors that are implemented as extensions to the VIM platform using the VIM's API. The pay-per-use monitor polling agent periodically supplies the billing system with billable events information. A separate monitoring agent provides further supplemental billing-related data, such as:

- *Cloud Consumer Subscription Type* – This information is used to identify the type of pricing model for usage fee calculations, including prepaid subscription with usage quota, postpaid subscription with maximum usage quota, and postpaid subscription with unlimited usage.
- *Resource Usage Category* – The billing management system uses this information to identify the range of usage fees that are applicable to each usage event. Examples include normal usage, reserved IT resource usage, and premium (managed) service usage.
- *Resource Usage Quota Consumption* – When usage contracts define IT resource usage quotas, usage event conditions are typically supplemented with quota consumption and updated quota limits.

Figure 9.14 illustrates the steps that are taken by DTGOV's pay-per-use monitor during a typical usage event.

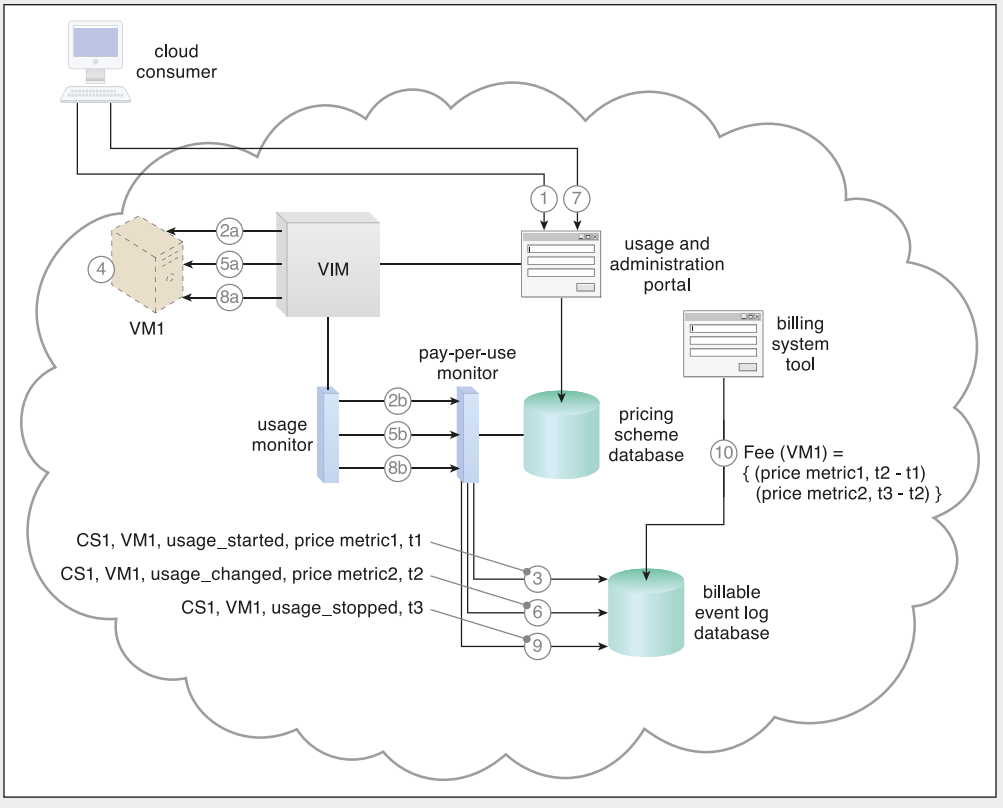


Figure 9.14

The cloud consumer (CS_ID = CS1) creates and starts a virtual server (VM_ID = VM1) of configuration size type 1 (VM_TYPE = type1) (1). The VIM creates the virtual server instance as requested (2a). The VIM's event-driven API generates a resource usage event with timestamp = t1, which is captured and forwarded to the pay-per-use monitor by the cloud usage monitor (2b). The pay-per-use monitor interacts with the pricing scheme database to identify the chargeback and usage metrics that apply to the resource usage. A "started usage" billable event is generated and stored in the billable event log database (3). The virtual server's usage increases and reaches the auto-scaling threshold (4). The VIM scales up Virtual Server VM1 (5a) from configuration type 1 to type 2 (VM_TYPE = type2). The VIM's event-driven API generates a resource usage event with timestamp = t2, which is captured and forwarded to the pay-per-use monitor by the cloud usage monitor (5b). The pay-per-use monitor interacts with the pricing scheme database to identify the chargeback and usage metrics that apply to the updated IT resource usage. A "changed usage" billable event is generated and stored in the billable event log database (6). The cloud consumer shuts down the virtual server (7) and the VIM stops Virtual Server VM1 (8a). The VIM's event-driven API generates a resource usage event with timestamp = t3, which is captured and forwarded to the pay-per-use monitor by the cloud usage monitor (8b). The pay-per-use monitor interacts with the pricing scheme database to identify the chargeback and usage metrics that apply to the updated IT resource usage. A "finished usage" billable event is generated and stored in the billable event log database (9). The billing system tool can now be used by the cloud provider to access the log database and calculate the total usage fee for the virtual server as Fee(VM1) (10).

9.5 Audit Monitor

The *audit monitor* mechanism is used to collect audit tracking data for networks and IT resources in support of (or dictated by) regulatory and contractual obligations. Figure 9.15 depicts an audit monitor implemented as a monitoring agent that intercepts "login" requests and stores the requestor's security credentials, as well as both failed and successful login attempts, in a log database for future audit reporting purposes.

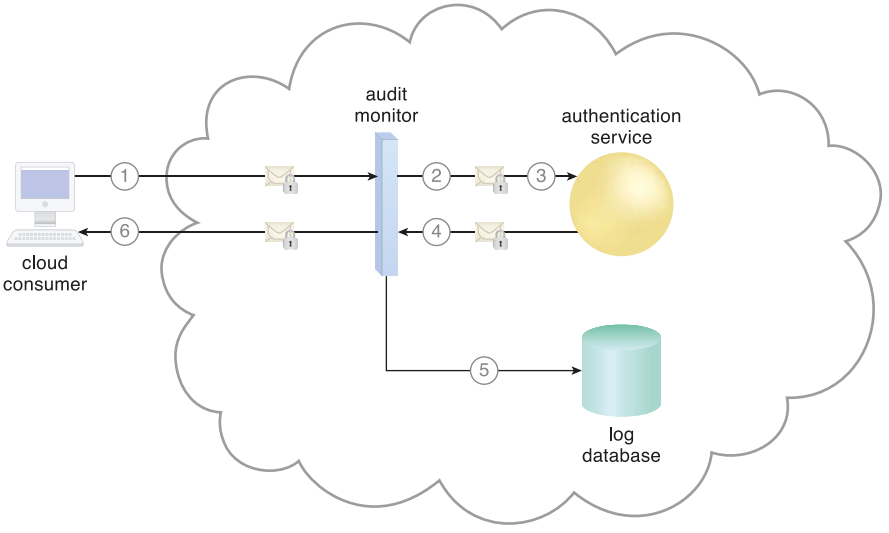


Figure 9.15

A cloud service consumer requests access to a cloud service by sending a login request message with security credentials (1). The audit monitor intercepts the message (2) and forwards it to the authentication service (3). The authentication service processes the security credentials. A response message is generated for the cloud service consumer, in addition to the results from the login attempt (4). The audit monitor intercepts the response message and stores the entire collected login event details in the log database, as per the organization's audit policy requirements (5). Access has been granted, and a response is sent back to the cloud service consumer (6).

CASE STUDY EXAMPLE

A key feature of Innovartus's role-playing solution is its unique user interface. However, the advanced technologies used for its design have imposed licensing restrictions that legally prevent Innovartus from charging users in certain geographical regions for usage of the solution. Innovartus's legal department is working on getting

these issues resolved. But in the meantime, it has provided the IT department with a list of countries in which the application can either not be accessed by users or in which user access needs to be free of charge.

To collect information about the origin of clients accessing the application, Innovartus asks its cloud provider to establish an audit monitoring system. The cloud provider deploys an audit monitoring agent to intercept each inbound message, analyze its corresponding HTTP header, and collect details about the origin of the end user. As per Innovartus's request, the cloud provider further adds a log database to collect the regional data of each end-user request for future reporting purposes. Innovartus further upgrades its application so that end users from select countries are able to access the application at no charge (Figure 9.16).

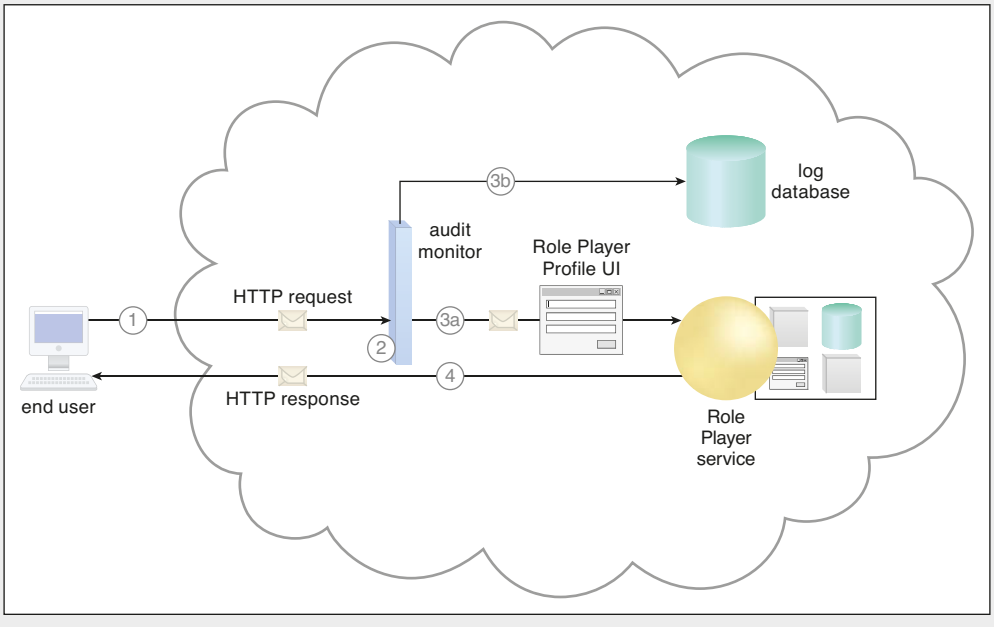


Figure 9.16

An end user attempts access to the Role Player cloud service (1). An audit monitor transparently intercepts the HTTP request message and analyzes the message header to determine the geographical origin of the end user (2). The audit monitoring agent determines that the end user is from a region in which Innovartus is not authorized to charge a fee for access to the application. The agent forwards the message to the cloud service (3a) and generates the audit track information for storage in the log database (3b). The cloud service receives the HTTP message and grants the end user access at no charge (4).

9.6 Failover System

The *failover system* mechanism is used to increase the reliability and availability of IT resources by using established clustering technology to provide redundant implementations. A failover system is configured to automatically switch over to a redundant or standby IT resource instance whenever the currently active IT resource becomes unavailable.

Failover systems are commonly used for mission-critical programs and reusable services that can introduce a single point of failure for multiple applications. A failover system can span more than one geographical region so that each location hosts one or more redundant implementations of the same IT resource.

The resource replication mechanism is sometimes utilized by the failover system to provide redundant IT resource instances, which are actively monitored for the detection of errors and unavailability conditions.

Failover systems come in two basic configurations.

Active–Active

In an *active–active* configuration, redundant implementations of the IT resource actively serve the workload synchronously (Figure 9.17). Load balancing among active instances is required. When a failure is detected, the failed instance is removed from the load balancing scheduler (Figure 9.18). Whichever IT resource remains operational when a failure is detected takes over the processing (Figure 9.19).

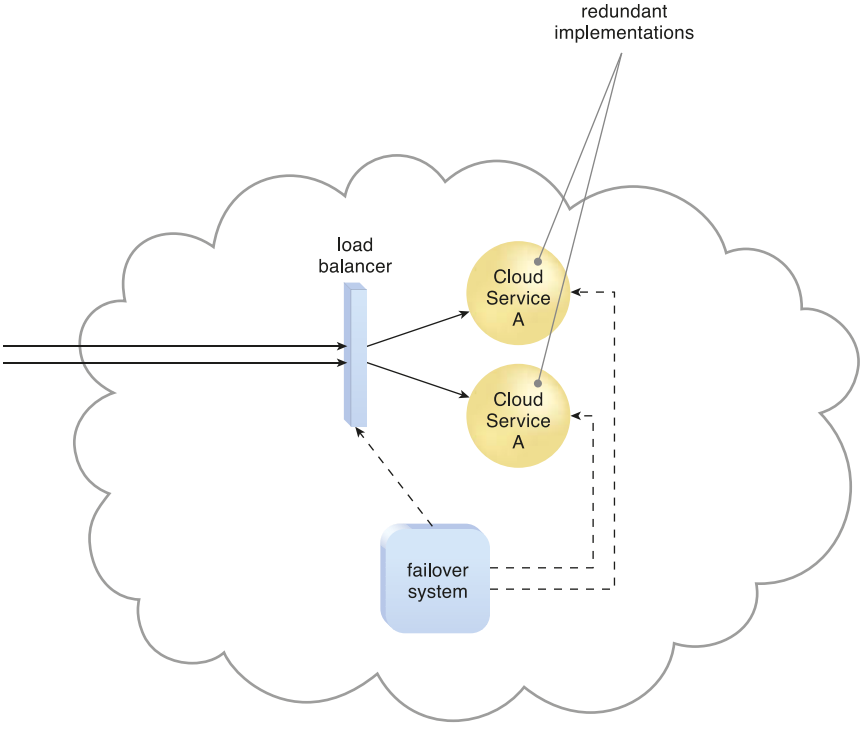


Figure 9.17
The failover system monitors the operational status of Cloud Service A.

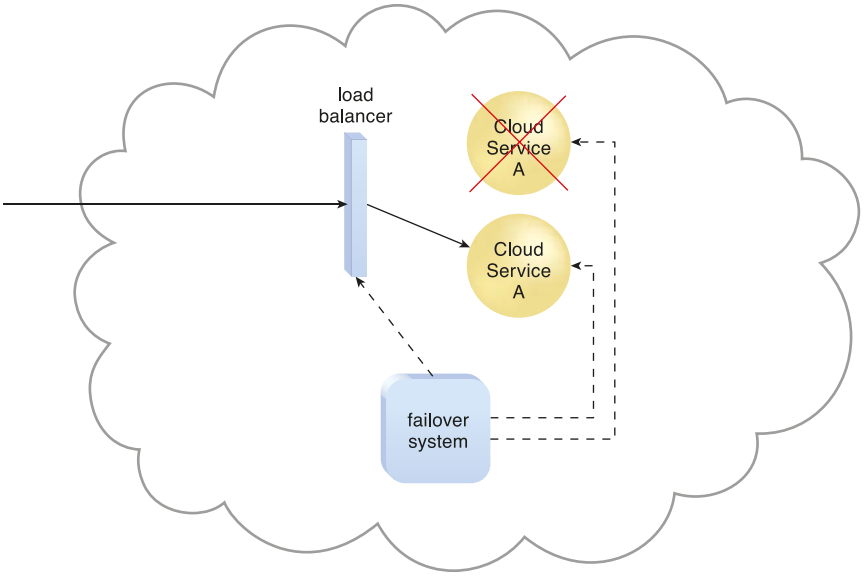


Figure 9.18

When a failure is detected in one Cloud Service A implementation, the failover system commands the load balancer to switch over the workload to the redundant Cloud Service A implementation.

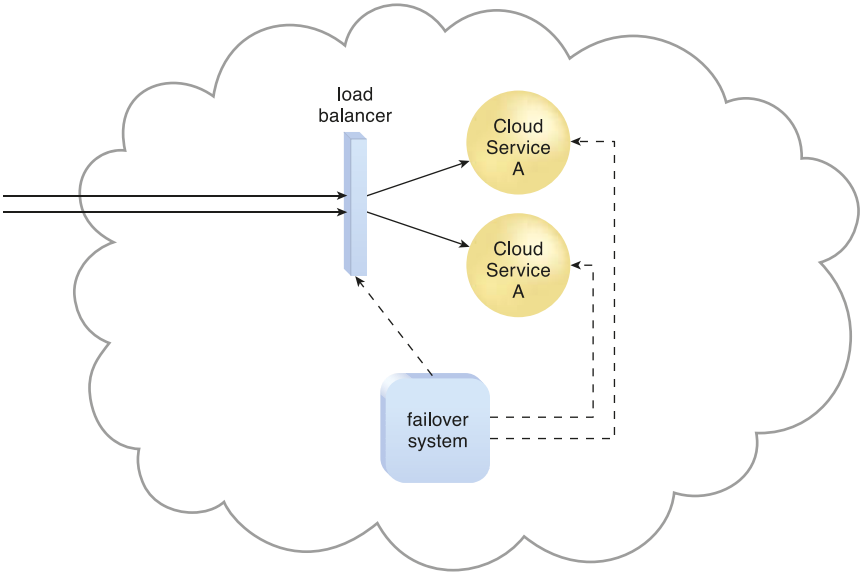


Figure 9.19

The failed Cloud Service A implementation is recovered or replicated into an operational cloud service. The failover system now commands the load balancer to distribute the workload again.

Active–Passive

In an *active–passive* configuration, a standby or inactive implementation is activated to take over the processing from the IT resource that becomes unavailable, and the corresponding workload is redirected to the instance taking over the operation (Figures 9.20 to 9.22).

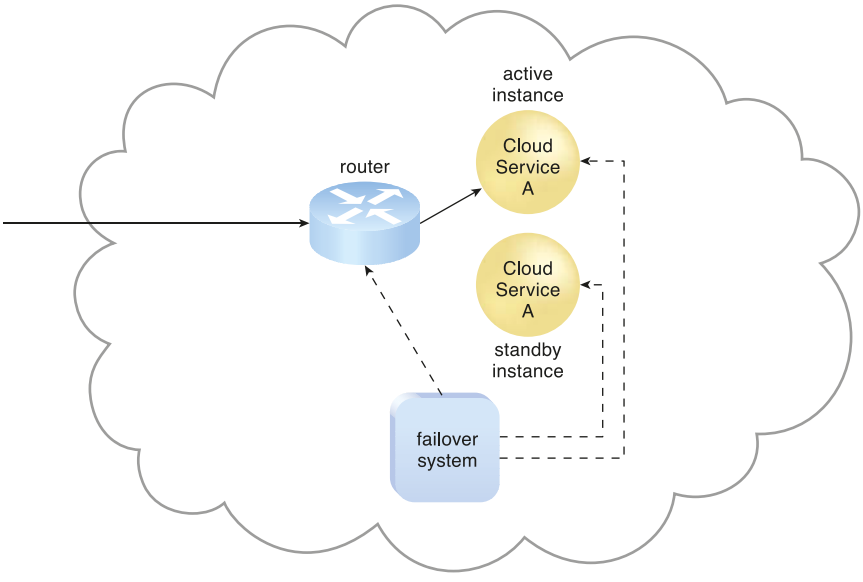


Figure 9.20

The failover system monitors the operational status of Cloud Service A. The Cloud Service A implementation acting as the active instance is receiving cloud service consumer requests.

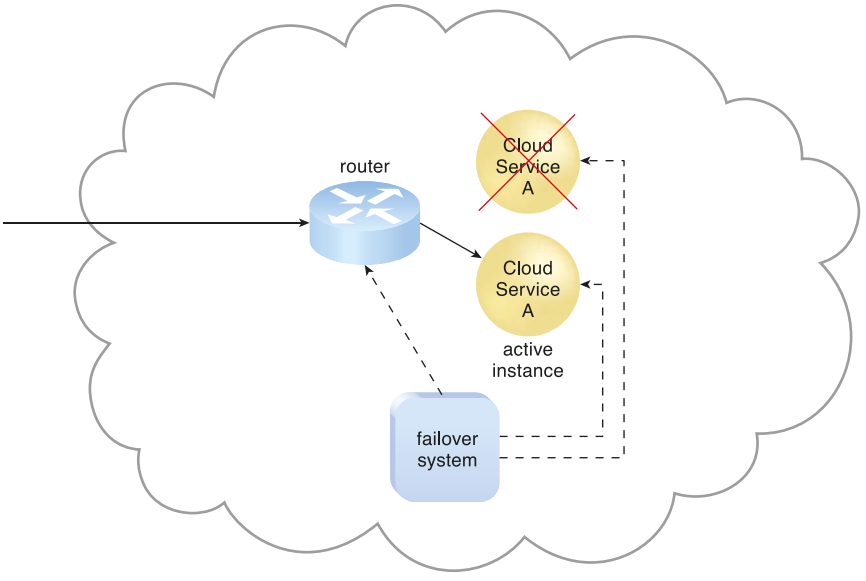


Figure 9.21

The Cloud Service A implementation acting as the active instance encounters a failure that is detected by the failover system, which subsequently activates the inactive Cloud Service A implementation and redirects the workload toward it. The newly invoked Cloud Service A implementation now assumes the role of active instance.

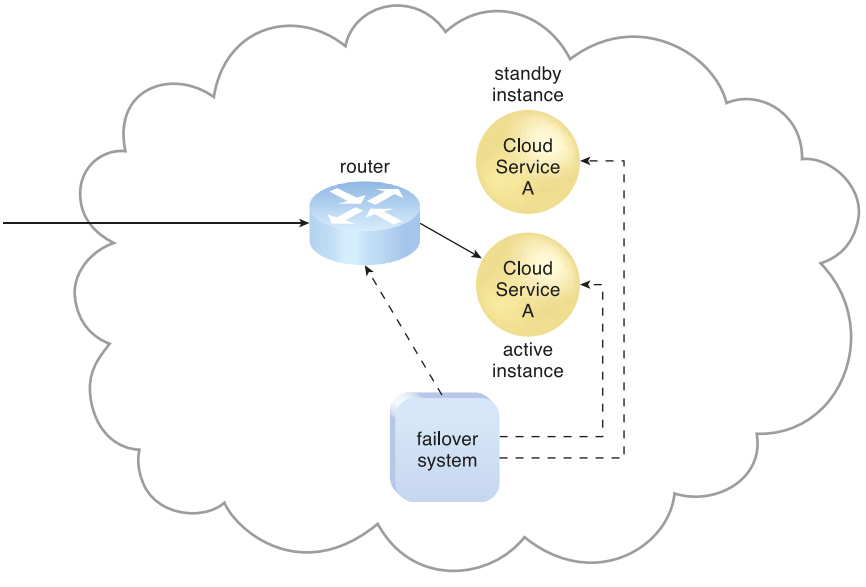


Figure 9.22

The failed Cloud Service A implementation is recovered or replicated, and is now positioned as the standby instance, while the previously invoked Cloud Service A continues to serve as the active instance.

Some failover systems are designed to redirect workloads to active IT resources that rely on specialized load balancers that detect failure conditions and exclude failed IT resource instances from the workload distribution. This type of failover system is suitable for IT resources that do not require execution state management and provide stateless processing capabilities. In technology architectures that are typically based on clustering and virtualization technologies, the redundant or standby IT resource implementations are also required to share their state and execution context. A complex task that was executed on a failed IT resource can remain operational in one of its redundant implementations.

CASE STUDY EXAMPLE

DTGOV creates a resilient virtual server to support the allocation of virtual server instances that are hosting critical applications, which are being replicated in multiple data centers. The replicated resilient virtual server has an associated active–passive failover system. Its network traffic flow can be switched between the IT resource instances that are residing at different data centers, if the active instance were to fail (Figure 9.23).

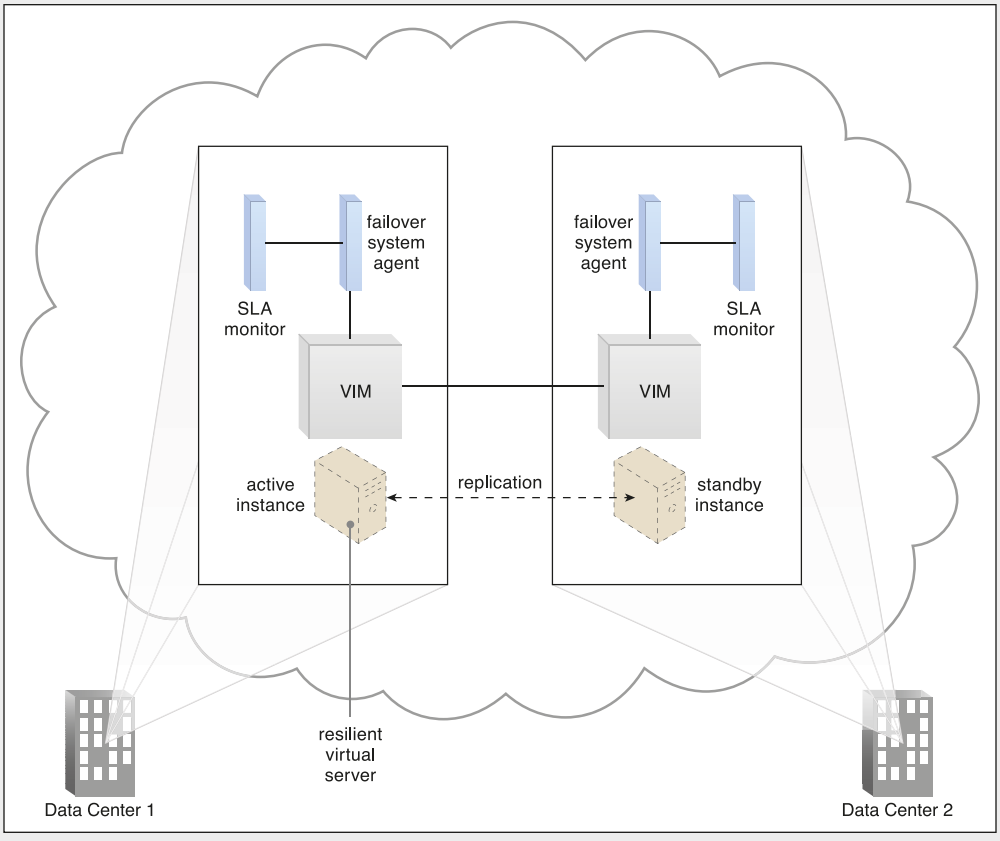


Figure 9.23

A resilient virtual server is established by replicating the virtual server instance across two different data centers, as performed by the VIM that is running at both data centers. The active instance receives the network traffic and is vertically scaling in response, while the standby instance has no workload and runs at the minimum configuration.

Figure 9.24 illustrates SLA monitors detecting failure in an active instance of a virtual server.

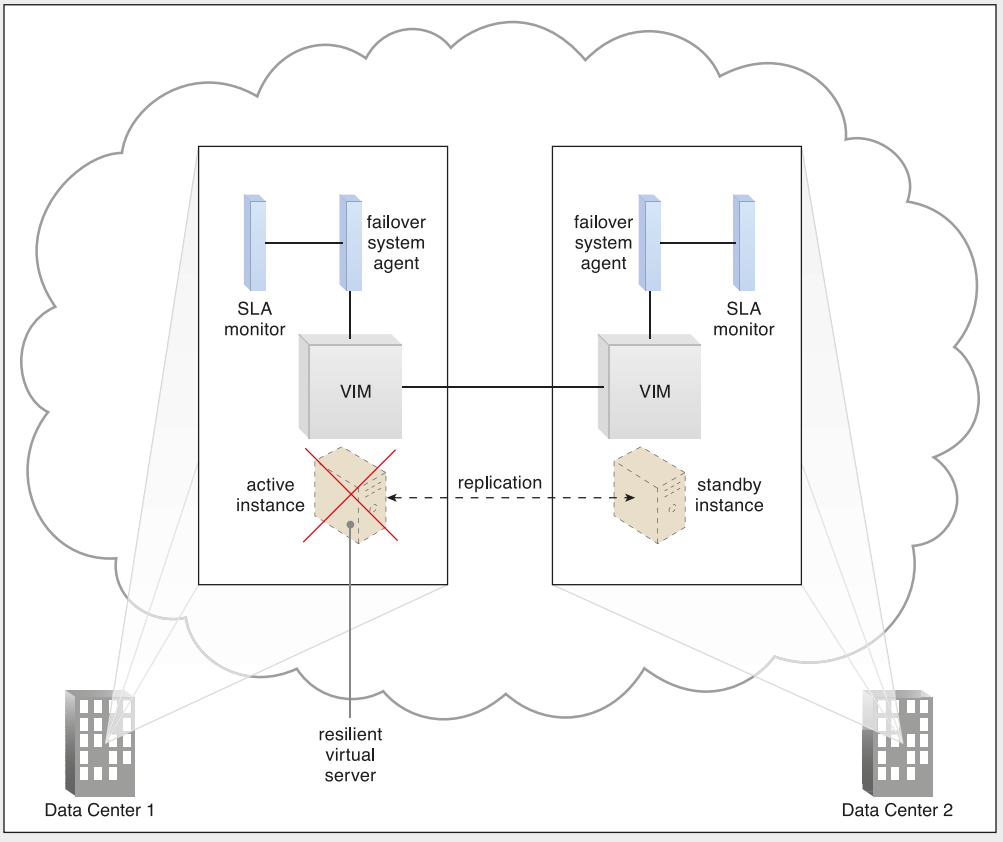


Figure 9.24
SLA monitors detect when the active virtual server instance becomes unavailable.

Figure 9.25 shows traffic being switched over to the standby instance, which has now become active.

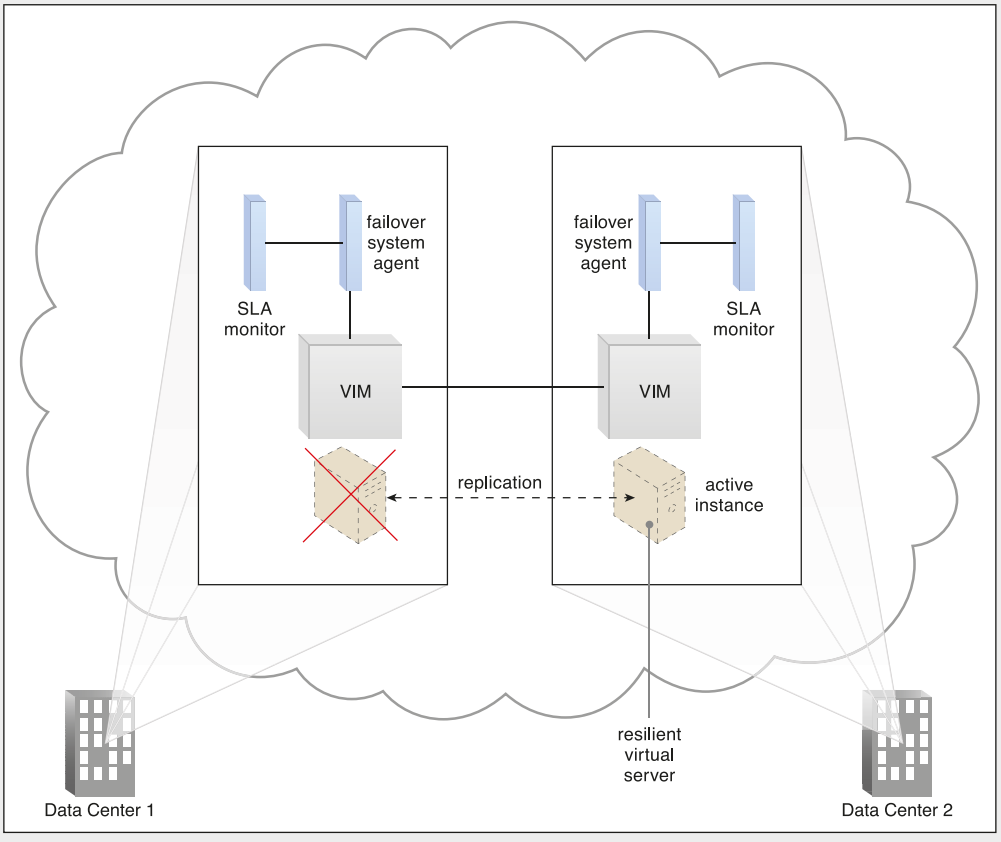


Figure 9.25

The failover system is implemented as an event-driven software agent that intercepts the message notifications the SLA monitors send regarding server unavailability. In response, the failover system interacts with the VIM and network management tools to redirect all the network traffic to the now-active standby instance.

In Figure 9.26, the failed virtual server becomes operational and turns into the standby instance.

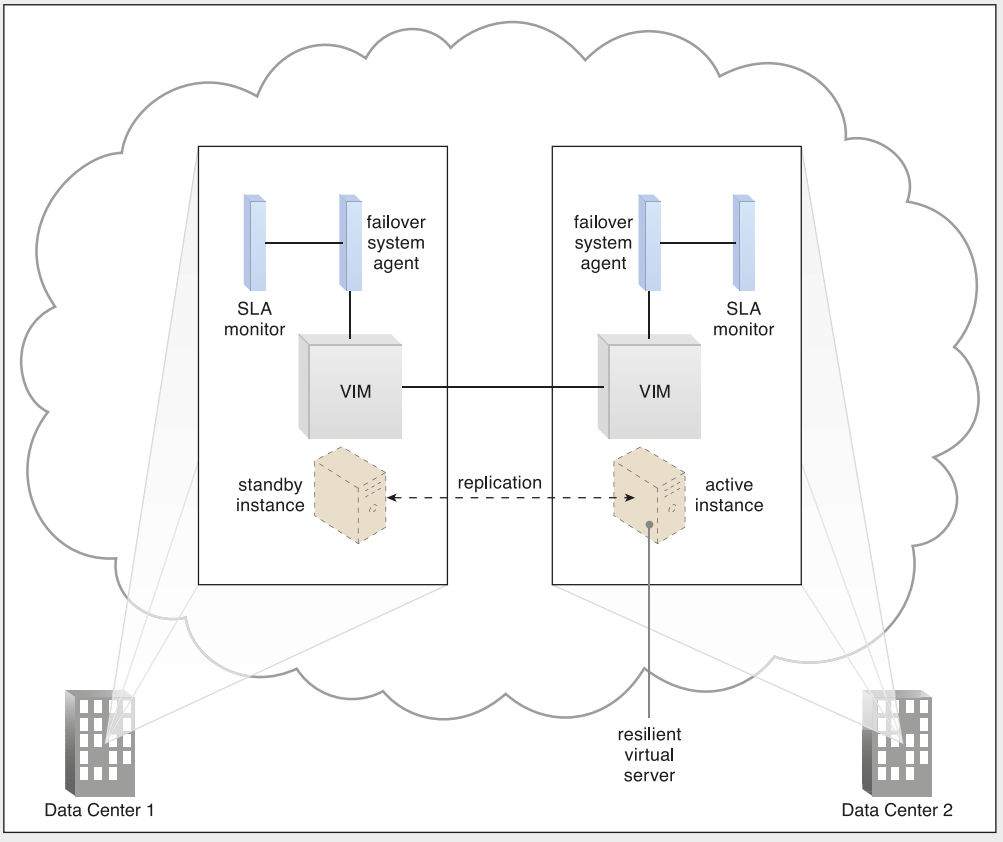


Figure 9.26
The failed virtual server instance is revived and scaled down to the minimum standby instance configuration after it resumes normal operation.

9.7 Resource Cluster

Cloud-based IT resources that are geographically diverse can be logically combined into groups to improve their allocation and use. The *resource cluster* mechanism (Figure 9.27) is used to group multiple IT resource instances so that they can be operated as a single IT resource. This increases the combined computing capacity, load balancing, and availability of the clustered IT resources.

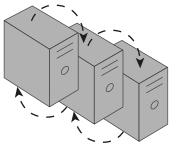


Figure 9.27
The curved dashed lines are used to indicate that IT resources are clustered.

Resource cluster architectures rely on high-speed dedicated network connections, or cluster nodes, between IT resource instances to communicate about workload distribution, task scheduling, data sharing, and system synchronization. A cluster management platform that is running as distributed middleware in all the cluster nodes is usually responsible for these activities. This platform implements a coordination function that allows distributed IT resources to appear as one IT resource, and also executes IT resources inside the cluster.

Common resource cluster types include:

- *Server Cluster* – Physical or virtual servers are clustered to increase performance and availability. Hypervisors running on different physical servers can be configured to share virtual server execution state (such as memory pages and processor register state) to establish clustered virtual servers. In such configurations, which usually require physical servers to have access to shared storage, virtual servers are able to live migrate from one physical server to another. In this process, the virtualization platform suspends the execution of a given virtual server at one physical server and resumes it on another physical server. The process is transparent to the virtual server operating system and can be used to increase scalability by live-migrating a virtual server that is running at an overloaded physical server to another physical server that has suitable capacity.
- *Database Cluster* – Designed to improve data availability, this high-availability resource cluster has a synchronization feature that maintains the consistency of data being stored at different storage devices used in the cluster. The redundant capacity is usually based on an active–active or active–passive failover system committed to maintaining the synchronization conditions.
- *Large Dataset Cluster* – Data partitioning and distribution is implemented so that the target datasets can be efficiently partitioned without compromising data integrity or computing accuracy. Each cluster node processes workloads without communicating with other nodes as much as in other cluster types.

Many resource clusters require cluster nodes to have almost identical computing capacity and characteristics to simplify the design of and maintain consistency within the resource cluster architecture. The cluster nodes in high-availability cluster architectures need to access and share common storage IT resources. This can require two layers of communication between the nodes—one for accessing the storage device and another to execute IT resource orchestration (Figure 9.28). Some resource clusters are designed with more loosely coupled IT resources that only require the network layer (Figure 9.29).

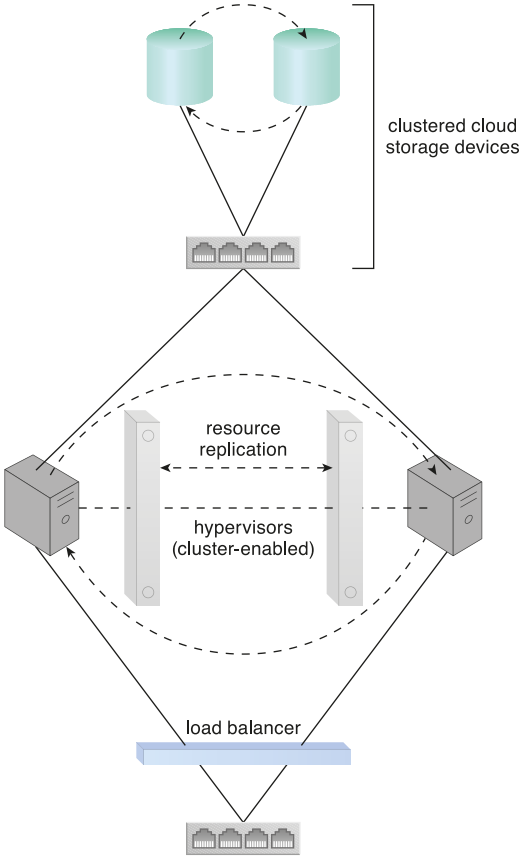


Figure 9.28
Load balancing and resource replication are implemented through a cluster-enabled hypervisor. A dedicated storage area network is used to connect the clustered storage and the clustered servers, which are able to share common cloud storage devices. This simplifies the storage replication process, which is independently carried out at the storage cluster. (See the *Hypervisor Clustering Architecture* section in Chapter 14 for a more detailed description.)

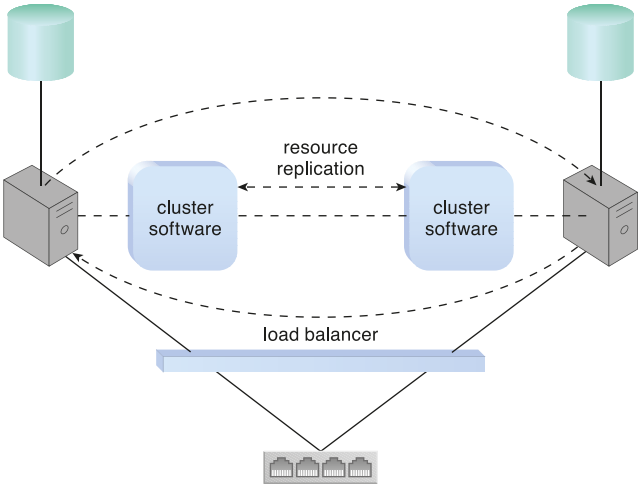


Figure 9.29
A loosely coupled server cluster that incorporates a load balancer. There is no shared storage. Resource replication is used to replicate cloud storage devices through the network by the cluster software.

There are two basic types of resource clusters:

- *Load Balanced Cluster* – This resource cluster specializes in distributing workloads among cluster nodes to increase IT resource capacity while preserving the centralization of IT resource management. It usually implements a load balancer mechanism that is either embedded within the cluster management platform or set up as a separate IT resource.
- *High-Availability (HA) Cluster* – A high-availability cluster maintains system availability in the event of multiple node failures and has redundant implementations of most or all of the clustered IT resources. It implements a failover system mechanism that monitors failure conditions and automatically redirects the workload away from any failed nodes.

The provisioning of clustered IT resources can be considerably more expensive than the provisioning of individual IT resources that have an equivalent computing capacity.

CASE STUDY EXAMPLE

DTGOV is considering introducing a clustered virtual server to run in a high-availability cluster as part of the virtualization platform (Figure 9.30). The virtual servers can live migrate among the physical servers, which are pooled in a high-availability hardware cluster that is controlled by coordinated cluster-enabled hypervisors. The coordination function keeps replicated snapshots of the running virtual servers to facilitate migration to other physical servers in the event of a failure.

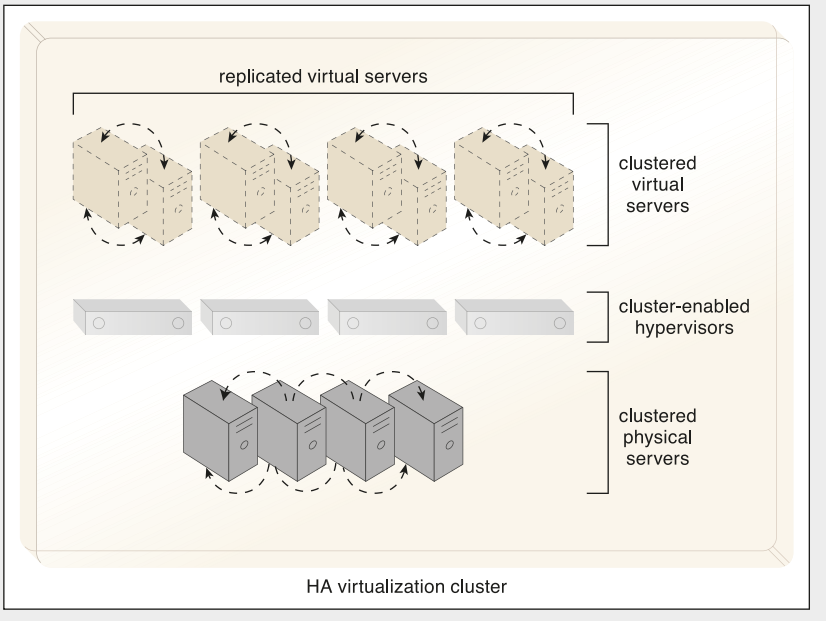


Figure 9.30

An HA virtualization cluster of physical servers is deployed using a cluster-enabled hypervisor, which guarantees that the physical servers are constantly in sync. Every virtual server that is instantiated in the cluster is automatically replicated in at least two physical servers.

Figure 9.31 identifies the virtual servers that are migrated from their failed physical host server to other available physical servers.

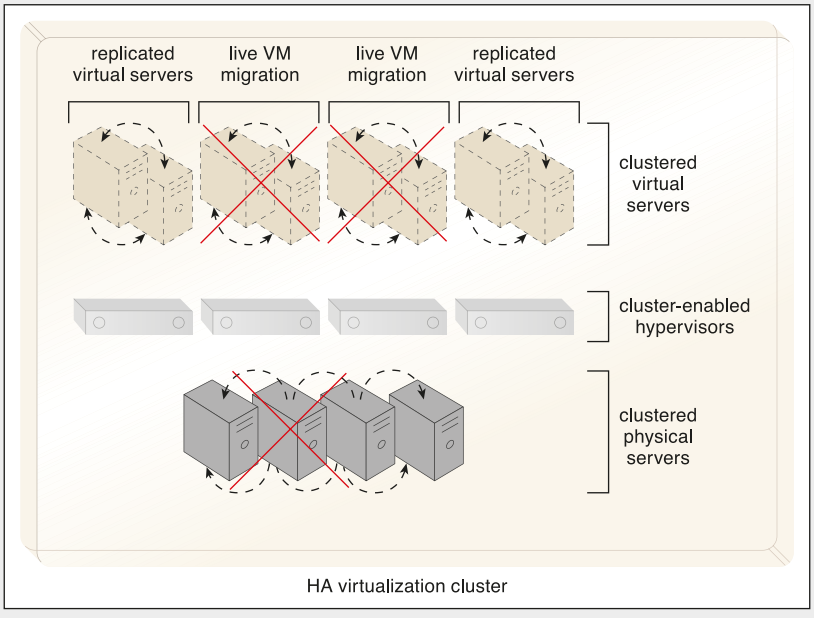


Figure 9.31
All the virtual servers that are hosted on a physical server experiencing failure are automatically migrated to other physical servers.

9.8 Multi-Device Broker

An individual cloud service may need to be accessed by a range of cloud service consumers differentiated by their hosting hardware devices and/or communication requirements. To overcome incompatibilities between a cloud service and a disparate cloud service consumer, mapping logic needs to be created to transform (or convert) information that is exchanged at runtime.

The *multi-device broker* mechanism is used to facilitate runtime data transformation so as to make a cloud service accessible to a wider range of cloud service consumer programs and devices (Figure 9.32).

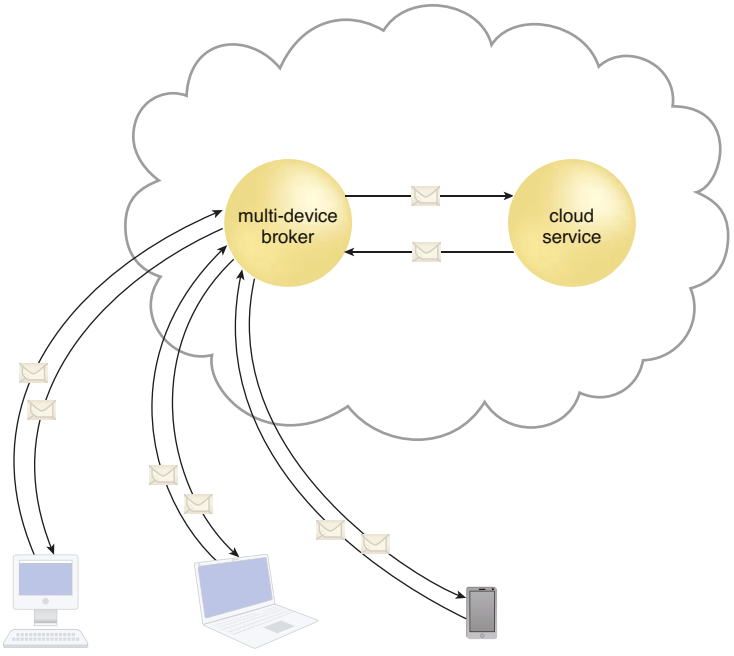


Figure 9.32
A multi-device broker contains the mapping logic necessary to transform data exchanges between a cloud service and different types of cloud service consumer devices. This scenario depicts the multi-device broker as a cloud service with its own API. This mechanism can also be implemented as a service agent that intercepts messages at runtime to perform necessary transformations.

Multi-device brokers commonly exist as gateways or incorporate gateway components, such as:

- *XML Gateway* – transmits and validates XML data
- *Cloud Storage Gateway* – transforms cloud storage protocols and encodes storage devices to facilitate data transfer and storage
- *Mobile Device Gateway* – transforms the communication protocols used by mobile devices into protocols that are compatible with a cloud service

The levels at which transformation logic can be created include:

- transport protocols
- messaging protocols
- storage device protocols
- data schemas/data models

For example, a multi-device broker may contain mapping logic that coverts both transport and messaging protocols for a cloud service consumer accessing a cloud service with a mobile device.

CASE STUDY EXAMPLE

Innovartus has decided to make its role-playing application available to various mobile and smartphone devices. A complication that hindered Innovartus's development team during the mobile enhancement design stage was the difficulty of reproducing identical user experiences across different mobile platforms. To resolve this issue, Innovartus implements a multi-device broker to intercept incoming messages from devices, identify the software platform, and convert the message format into the native, server-side application format (Figure 9.33).

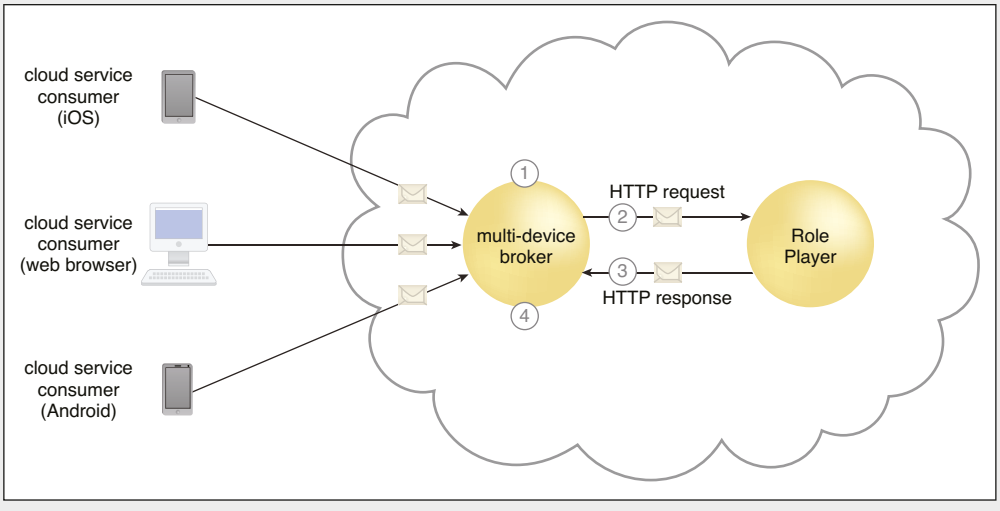


Figure 9.33

The multi-device broker intercepts incoming messages and detects the platform (web browser, iOS, Android) of the source device (1). The multi-device broker transforms the message into the standard format required by the Innovartus cloud service (2). The cloud service processes the request and responds using the same standard format (3). The multi-device broker transforms the response message into the format required by the source device and delivers the message (4).

9.9 State Management Database

A *state management database* is a storage device that is used to temporarily persist state data for software programs. As an alternative to caching state data in memory, software programs can off-load state data to the database to reduce the amount of runtime memory they consume (Figures 9.34 and 9.35). By doing so, the software programs

and the surrounding infrastructure are more scalable. State management databases are commonly used by cloud services, especially those involved in long-running runtime activities.

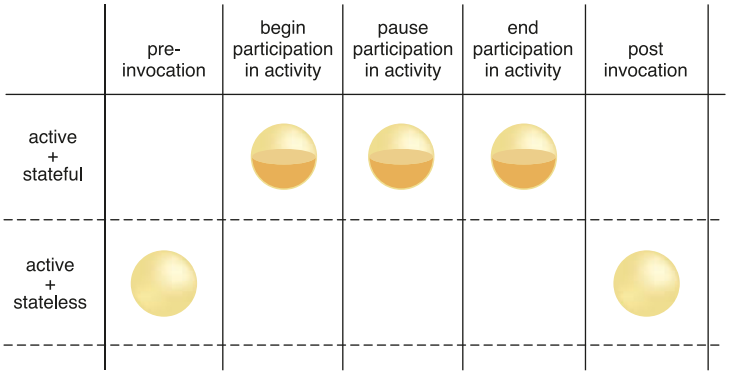


Figure 9.34
During the lifespan of a cloud service instance, it may be required to remain stateful and keep state data cached in memory, even when idle.

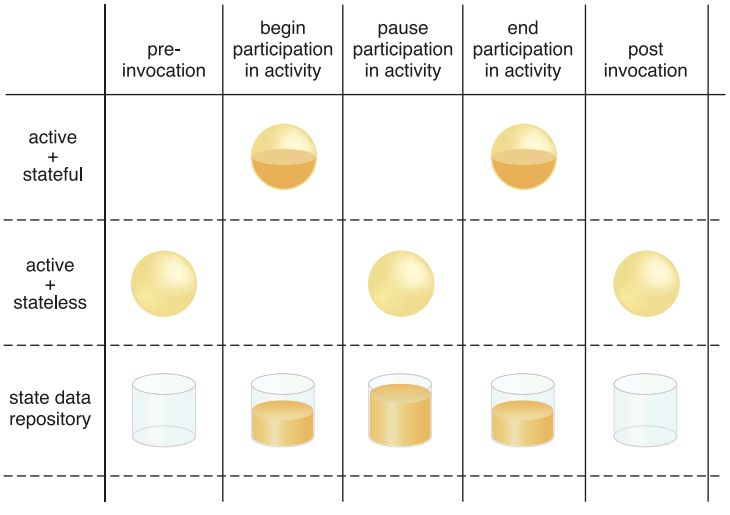


Figure 9.35
By deferring state data to a state data repository, the cloud service is able to transition to a stateless condition (or a partially stateless condition), thereby temporarily freeing system resources.

CASE STUDY EXAMPLE

ATN is expanding its ready-made environment architecture to allow for the deferral of state information for extended periods by utilizing the state management database mechanism. Figure 9.36 demonstrates how a cloud service consumer working with a ready-made environment pauses activity, causing the environment to off-load cached state data.

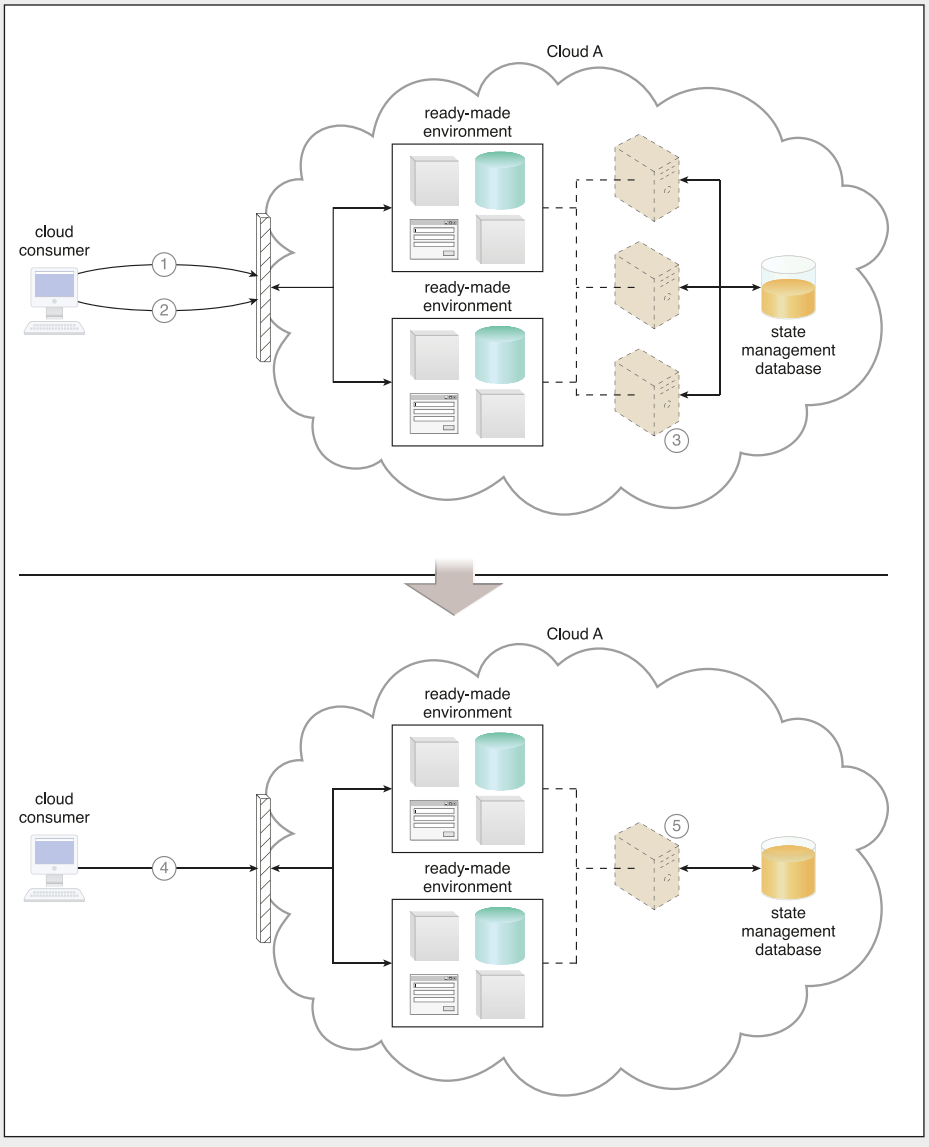


Figure 9.36

The cloud consumer accesses the ready-made environment and requires three virtual servers to perform all activities (1). The cloud consumer pauses activity. All the state data needs to be preserved for future access to the ready-made environment (2). The underlying infrastructure is automatically scaled in by reducing the number of virtual servers. State data is saved in the state management database, and one virtual server remains active to allow for future logins by the cloud consumer (3). At a later point, the cloud consumer logs in and accesses the ready-made environment to continue activity (4). The underlying infrastructure is automatically scaled out by increasing the number of virtual servers and by retrieving the state data from the state management database (5).

This page intentionally left blank

Chapter 10

Cloud Security and Cybersecurity Access-Oriented Mechanisms

This section describes the following mechanisms that are focused on establishing cloud access controls that cloud access monitoring functions.

- Encryption
- Hashing
- Digital Signature
- Cloud-Based Security Groups
- Public Key Infrastructure (PKI) System
- Single Sign-On (SSO) System
- Hardened Virtual Server Image
- Firewall
- Virtual Private Network (VPN)
- Biometric Scanner
- Multi-Factor Authentication (MFA) System
- Identity and Access Management (IAM) System
- Intrusion Detection System (IDS)
- Penetration Testing Tool
- User Behavior Analytics (UBA) System
- Third-Party Software Update Utility
- Network Intrusion Monitor
- Authentication Log Monitor
- VPN Monitor

10.1 Encryption

Data, by default, is coded in a readable format known as *plaintext*. When transmitted over a network, plaintext is vulnerable to unauthorized and potentially malicious access. The *encryption* mechanism is a digital coding system dedicated to preserving the confidentiality and integrity of data. It is used for encoding plaintext data into a protected and unreadable format.

Encryption technology commonly relies on a standardized algorithm called a *cipher* to transform original plaintext data into encrypted data, referred to as *ciphertext*. Access to ciphertext does not divulge the original plaintext data, apart from some forms of metadata, such as message length and creation date. When encryption is applied to plaintext data, the data is paired with a string of characters called an *encryption key*, a secret message that is established by and shared among authorized parties. The encryption key is used to decrypt the ciphertext back into its original plaintext format.

The encryption mechanism can help counter the traffic eavesdropping, malicious intermediary, insufficient authorization, and overlapping trust boundaries security threats. For example, malicious service agents that attempt traffic eavesdropping are unable to decrypt messages in transit if they do not have the encryption key (Figure 10.1).

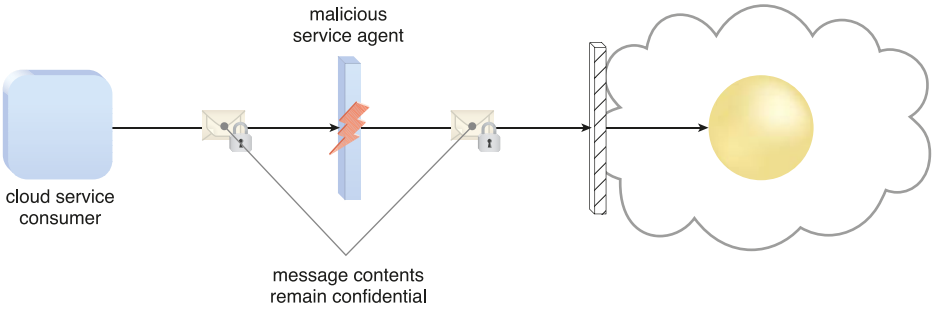


Figure 10.1

A malicious service agent is unable to retrieve data from an encrypted message. The retrieval attempt may furthermore be revealed to the cloud service consumer. (Note the use of the lock symbol to indicate that a security mechanism has been applied to the message contents.)

There are two common forms of encryption known as symmetric encryption and asymmetric encryption.

Symmetric Encryption

Symmetric encryption (also known as secret key cryptography) uses the same key for both encryption and decryption, both of which are performed by authorized parties that use the one shared key. Messages that are encrypted with a specific key can be decrypted by only that same key. Parties that rightfully decrypt the data are provided with evidence that the original encryption was performed by parties that rightfully possess the key. A basic authentication check is always performed, because only authorized parties that own the key can create messages. This maintains and verifies data confidentiality.

Note that symmetric encryption does not have the characteristic of non-repudiation, since determining exactly which party performed the message encryption or decryption is not possible if more than one party is in possession of the key.

Asymmetric Encryption

Asymmetric encryption relies on the use of two different keys—namely, a private key and a public key. With asymmetric encryption (which is also referred to as *public key cryptography*), the private key is known only to its owner, whereas the public key is commonly available. A document that was encrypted with a private key can only be correctly decrypted with the corresponding public key. Conversely, a document that was encrypted with a public key can be decrypted only using its private key counterpart. As a result of two different keys being used instead of just the one, asymmetric encryption is almost always computationally slower than symmetric encryption.

The level of security that is achieved is dictated by whether a private key or a public key was used to encrypt the plaintext data. As every asymmetrically encrypted message has its own private–public key pair, messages that were encrypted with a private key can be correctly decrypted by any party with the corresponding public key. This method of encryption does not offer any confidentiality protection, even though successful decryption proves that the text was encrypted by the rightful private key owner. Private key encryption therefore offers integrity protection in addition to authenticity and non-repudiation. A message that was encrypted with a public key can only be decrypted by the rightful private key owner, which provides confidentiality protection. However, any party that has the public key can generate the ciphertext, meaning this method provides neither message integrity nor authenticity protection due to the communal nature of the public key.

NOTE

The encryption mechanism, when used to secure web-based data transmissions, is most commonly applied via HTTPS, which refers to the use of SSL/TLS as an underlying encryption protocol for HTTP. TLS (transport layer security) is the successor to the SSL (secure sockets layer) technology. Because asymmetric encryption is usually more time-consuming than symmetric encryption, TLS uses the former only for its key exchange method. TLS systems then switch to symmetric encryption once the keys have been exchanged.

Most TLS implementations primarily support RSA as the chief asymmetric encryption cipher, while ciphers such as RC4, Triple-DES, and AES are supported for symmetric encryption.

CASE STUDY EXAMPLE

Innovartus has recently learned that users who access their User Registration Portal via public Wi-Fi hot zones and unsecured LANs may be transmitting personal user profile details via plaintext. Innovartus immediately remedies this vulnerability by applying the encryption mechanism to its web portal via the use of HTTPS (Figure 10.2).

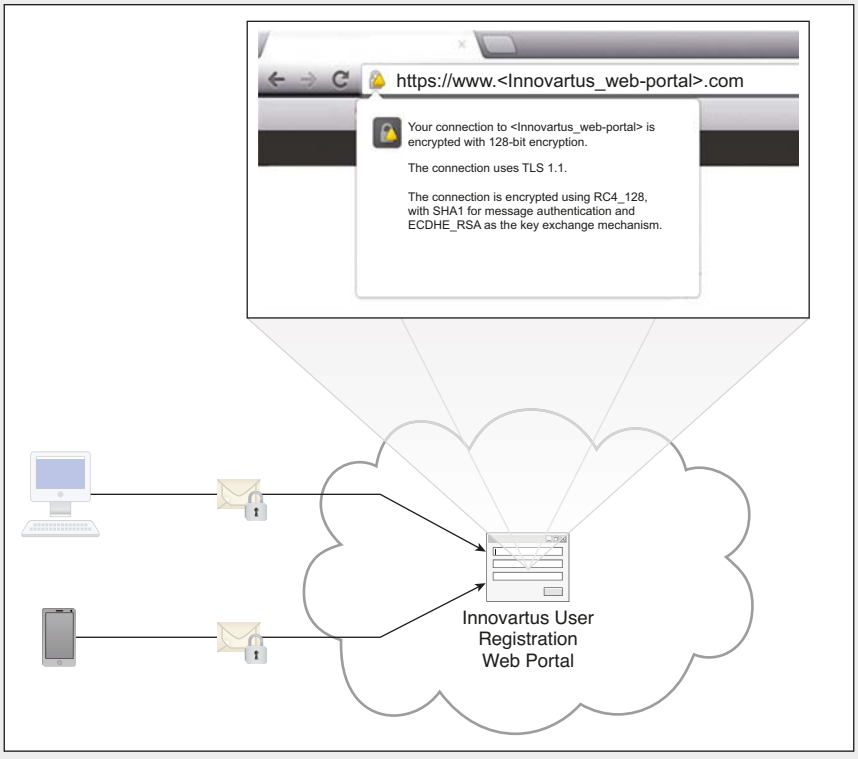


Figure 10.2
The encryption mechanism is added to the communication channel between outside users and Innovartus's User Registration Portal. This safeguards message confidentiality via the use of HTTPS.

10.2 Hashing

The *hashing* mechanism is used when a one-way, nonreversible form of data protection is required. Once hashing has been applied to a message, it is locked and no key is provided for the message to be unlocked. A common application of this mechanism is the storage of passwords.

Hashing technology can be used to derive a hashing code or *message digest* from a message, which is often of a fixed length and smaller than the original message. The message sender can then utilize the hashing mechanism to attach the message digest to the message. The recipient applies the same hash function to the message to verify that the

produced message digest is identical to the one that accompanied the message. Any alteration to the original data results in an entirely different message digest and clearly indicates that tampering has occurred.

In addition to its utilization for protecting stored data, the cloud threats that can be mitigated by the hashing mechanism include malicious intermediary and insufficient authorization. An example of the former is illustrated in Figure 10.3.

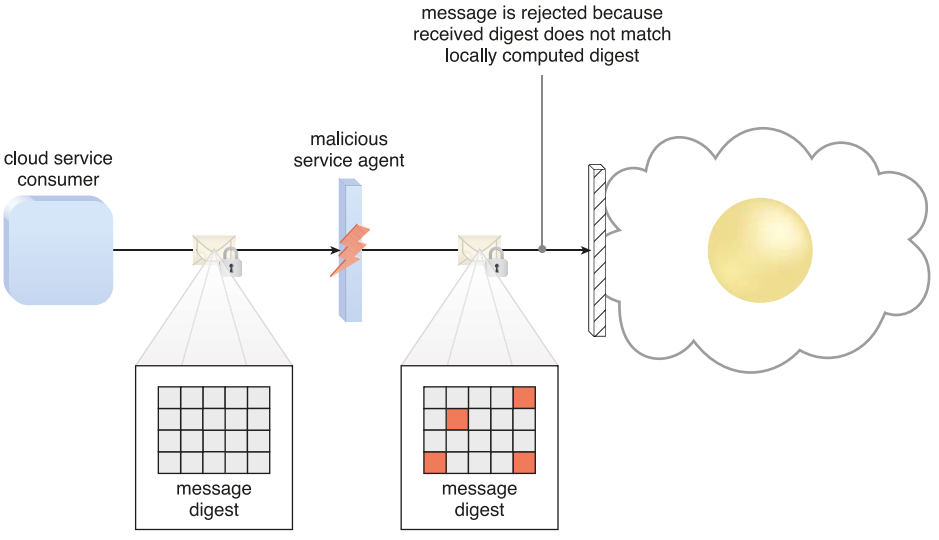


Figure 10.3

A hashing function is applied to protect the integrity of a message that is intercepted and altered by a malicious service agent, before it is forwarded. The firewall can be configured to determine that the message has been altered, thereby enabling it to reject the message before it can proceed to the cloud service.

CASE STUDY EXAMPLE

A subset of the applications that have been selected to be ported to ATN's PaaS platform allow users to access and alter highly sensitive corporate data. This information is being hosted on a cloud to enable access by trusted partners, who may use it for critical calculation and assessment purposes. Concerned that the data could be tampered with, ATN decides to apply the hashing mechanism as a means of protecting and preserving the data's integrity.

ATN cloud resource administrators work with the cloud provider to incorporate a digest-generating procedure with each application version that is deployed in the cloud. Current values are logged to a secure database on premises and the procedure is regularly repeated with the results analyzed. Figure 10.4 illustrates how ATN implements hashing to determine whether any non-authorized actions have been performed against the ported applications.

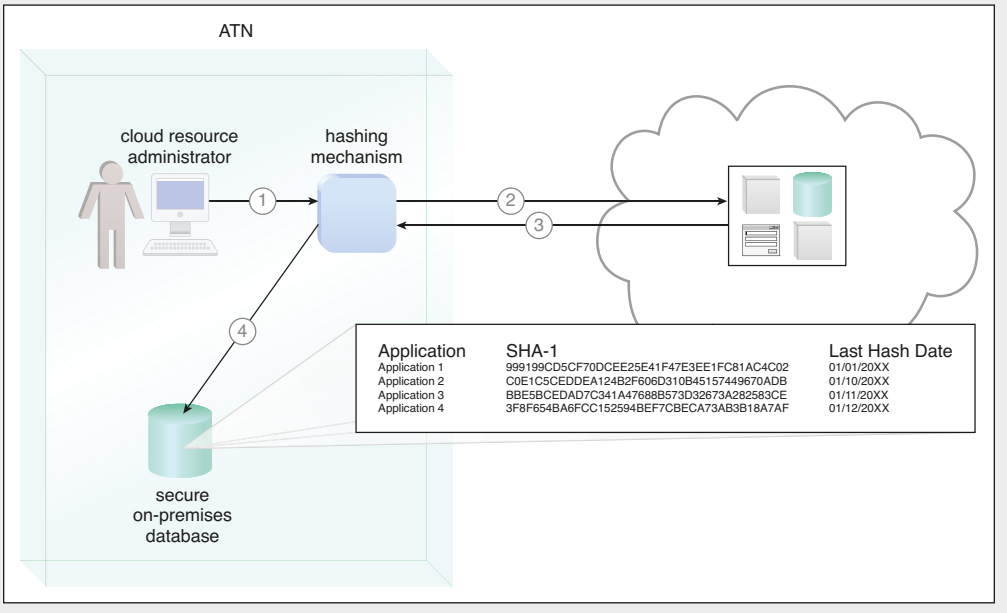


Figure 10.4

A hashing procedure is invoked when the PaaS environment is accessed (1). The applications that were ported to this environment are checked (2) and their message digests are calculated (3). The message digests are stored in a secure database on premises (4), and a notification is issued if any of their values are not identical to the ones in storage.

10.3 Digital Signature

The *digital signature* mechanism is a means of providing data authenticity and integrity through authentication and non-repudiation. A message is assigned a digital signature prior to transmission, which is then rendered invalid if the message experiences any subsequent, unauthorized modifications. A digital signature provides evidence that the message received is the same as the one created by its rightful sender.

Both hashing and asymmetric encryption are involved in the creation of a digital signature, which essentially exists as a message digest that was encrypted by a private key and appended to the original message. The recipient verifies the signature validity and uses the corresponding public key to decrypt the digital signature, which produces the message digest. The hashing mechanism can also be applied to the original message to produce this message digest. Identical results from the two different processes indicate that the message maintained its integrity.

The digital signature mechanism helps mitigate the malicious intermediary, insufficient authorization, and overlapping trust boundaries security threats (Figure 10.5).

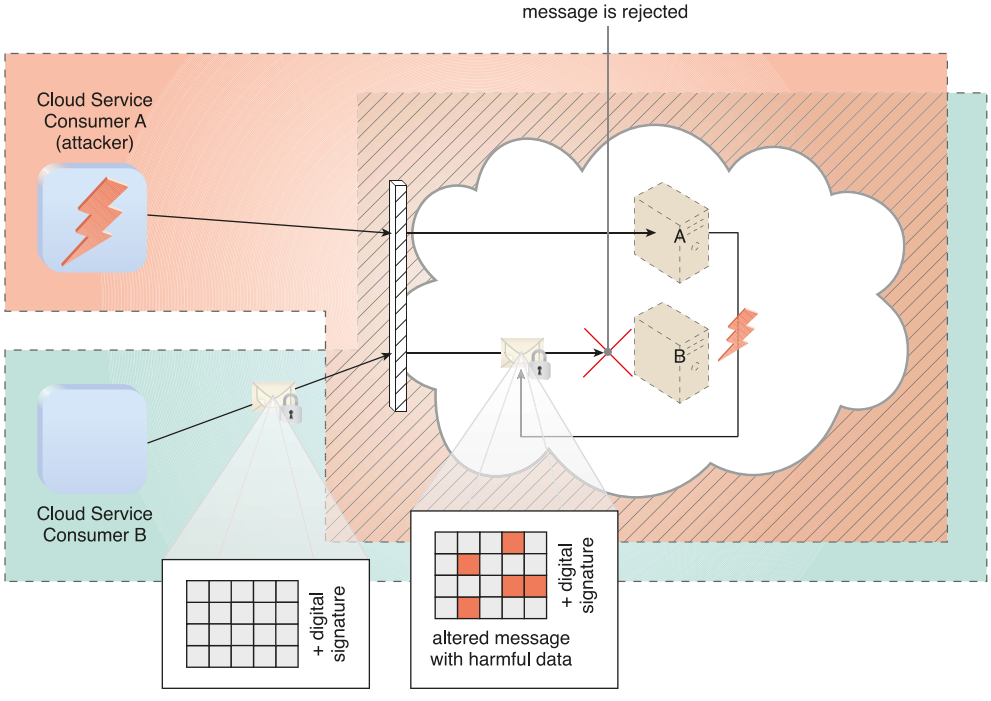


Figure 10.5

Cloud Service Consumer B sends a message that was digitally signed but was altered by trusted attacker Cloud Service Consumer A. Virtual Server B is configured to verify digital signatures before processing incoming messages even if they are within its trust boundary. The message is revealed as illegitimate due to its invalid digital signature, and is therefore rejected by Virtual Server B.

CASE STUDY EXAMPLE

With DTGOV's client portfolio expanding to include public-sector organizations, many of its cloud computing policies have become unsuitable and require modification. Considering that public-sector organizations frequently handle strategic information, security safeguards need to be established to protect data manipulation and to establish a means of auditing activities that may impact government operations.

DTGOV proceeds to implement the digital signature mechanism specifically to protect its web-based management environment (Figure 10.6). Virtual server self-provisioning inside the IaaS environment and the tracking functionality of realtime SLA and billing are all performed via web portals. As a result, user error or malicious actions could result in legal and financial consequences.

Digital signatures provide DTGOV with the guarantee that every action performed is linked to its legitimate originator. Unauthorized access is expected to become highly improbable, since digital signatures are only accepted if the encryption key is identical to the secret key held by the rightful owner. Users will not have grounds to deny attempts at message adulteration because the digital signatures will confirm message integrity.

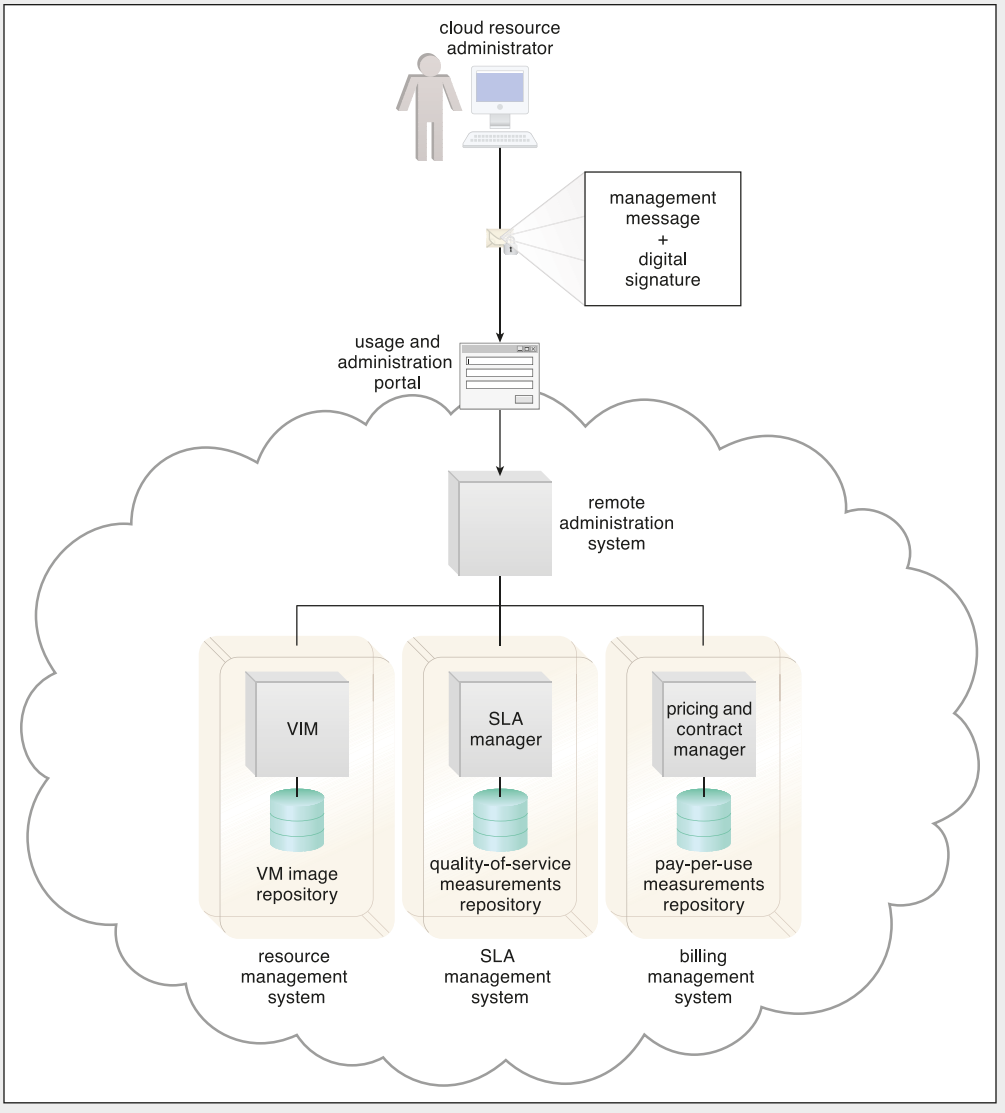


Figure 10.6

Whenever a cloud consumer performs a management action that is related to IT resources provisioned by DTGOV, the cloud service consumer program must include a digital signature in the message request to prove the legitimacy of its user.

10.4 Cloud-Based Security Groups

Similar to constructing dikes and levees that separate land from water, data protection is increased by placing barriers between IT resources. Cloud resource segmentation is a process by which separate physical and virtual IT environments are created for different users and groups. For example, an organization's WAN can be partitioned according to individual network security requirements. One network can be established with a resilient firewall for external internet access, while a second is deployed without a firewall because its users are internal and unable to access the internet.

Resource segmentation is used to enable virtualization by allocating a variety of physical IT resources to virtual machines. It needs to be optimized for public cloud environments, since organizational trust boundaries from different cloud consumers overlap when sharing the same underlying physical IT resources.

The cloud-based resource segmentation process creates *cloud-based security group* mechanisms that are determined through security policies. Networks are segmented into logical cloud-based security groups that form logical network perimeters. Each cloud-based IT resource is assigned to at least one logical cloud-based security group. Each logical cloud-based security group is assigned specific rules that govern the communication between the security groups.

Multiple virtual servers running on the same physical server can become members of different logical cloud-based security groups (Figure 10.7). Virtual servers can further be separated into public–private groups, development–production groups, or any other designation configured by the cloud resource administrator.

Cloud-based security groups delineate areas where different security measures can be applied. Properly implemented cloud-based security groups help limit unauthorized access to IT resources in the event of a security breach. This mechanism can be used to help counter the denial of service, insufficient authorization, overlapping trust boundaries, virtualization attack, and containerization attack threats, and is closely related to the logical network perimeter mechanism.

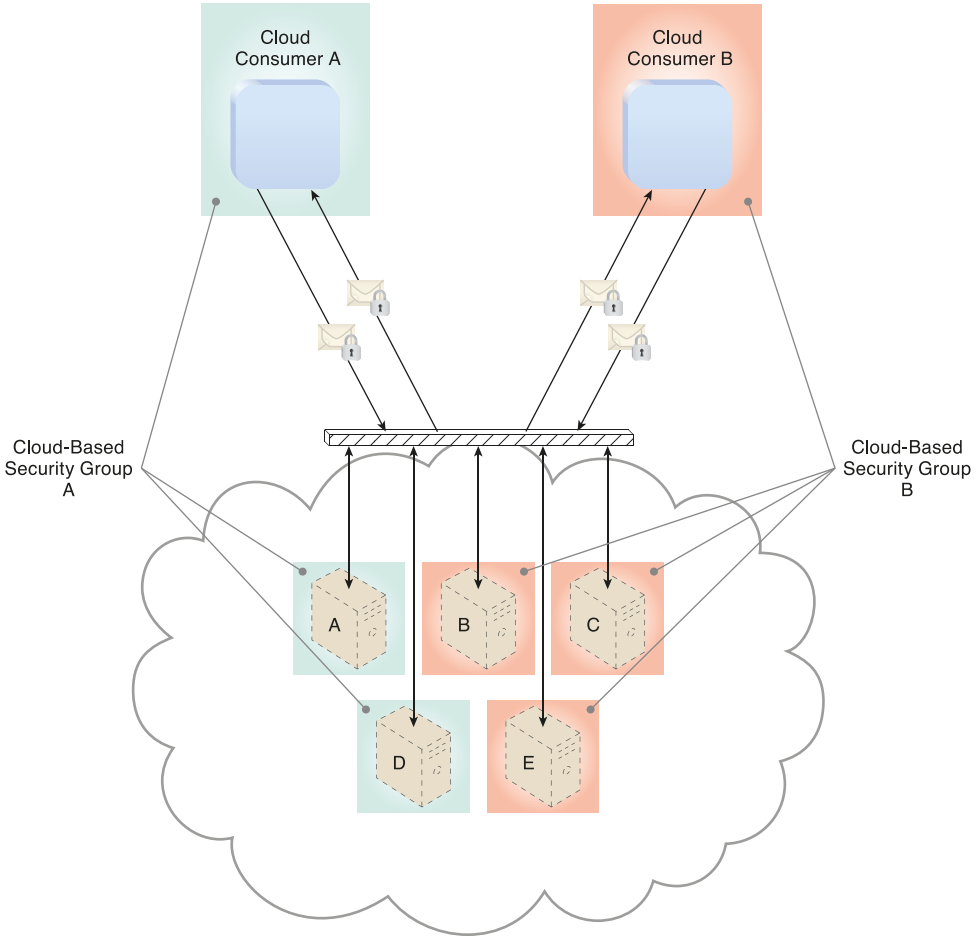


Figure 10.7

Cloud-Based Security Group A encompasses Virtual Servers A and D and is assigned to Cloud Consumer A. Cloud-Based Security Group B is comprised of Virtual Servers B, C, and E and is assigned to Cloud Consumer B. If Cloud Service Consumer A's credentials are compromised, the attacker would only be able to access and damage the virtual servers in Cloud-Based Security Group A, thereby protecting Virtual Servers B, C, and E.

CASE STUDY EXAMPLE

Now that DTGOV has itself become a cloud provider, security concerns are raised pertaining to its hosting of public-sector client data. A team of cloud security specialists is brought in to define cloud-based security groups together with the digital signature and PKI mechanisms.

Security policies are classified into levels of resource segmentation before being integrated into DTGOV's web portal management environment. Consistent with the security requirements guaranteed by its SLAs, DTGOV maps IT resource allocation to the appropriate logical cloud-based security group (Figure 10.8), which has its own security policy that clearly stipulates its IT resource isolation and control levels.

DTGOV informs its clients about the availability of these new security policies. Cloud consumers can optionally choose to utilize them and doing so results in increased fees.

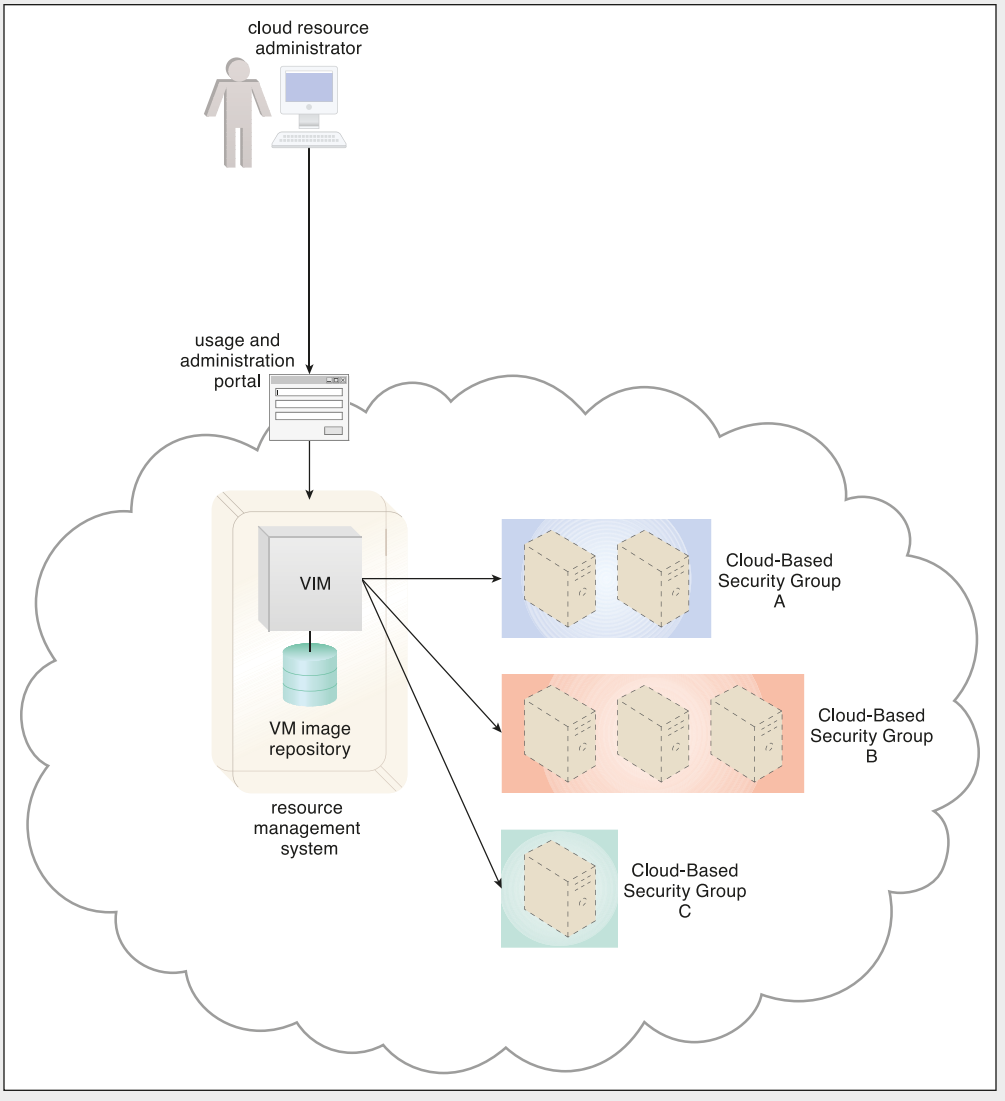


Figure 10.8

When an external cloud resource administrator accesses the web portal to allocate a virtual server, the requested security credentials are assessed and mapped to an internal security policy that assigns a corresponding cloud-based security group to the new virtual server.

10.5 Public Key Infrastructure (PKI) System

A common approach for managing the issuance of asymmetric keys is based on the *public key infrastructure (PKI) system* mechanism, which exists as a system of protocols, data formats, rules, and practices that enable large-scale systems to securely use public key cryptography. This system is used to associate public keys with their corresponding key owners (known as *public key identification*) while enabling the verification of key validity. PKI systems rely on the use of digital certificates, which are digitally signed data structures that bind public keys to certificate owner identities, as well as to related information, such as validity periods. Digital certificates are usually digitally signed by a third-party certificate authority (CA), as illustrated in Figure 10.9.

Other methods of generating digital signatures can be employed, even though the majority of digital certificates are issued by only a handful of trusted CAs like VeriSign and Comodo. Larger organizations, such as Microsoft, can act as their own CA and issue certificates to their clients and the public, since even individual users can generate certificates as long as they have the appropriate software tools.

Building up an acceptable level of trust for a CA is time-intensive but necessary. Rigorous security measures, substantial infrastructure investments, and stringent operational processes all contribute to establishing the credibility of a CA. The higher its level of trust and reliability, the more esteemed and reputable its certificates. The PKI system is a dependable method for implementing asymmetric encryption, managing cloud consumer and cloud provider identity information, and helping to defend against the malicious intermediary and insufficient authorization threats.

The PKI system mechanism is primarily used to counter the insufficient authorization threat.

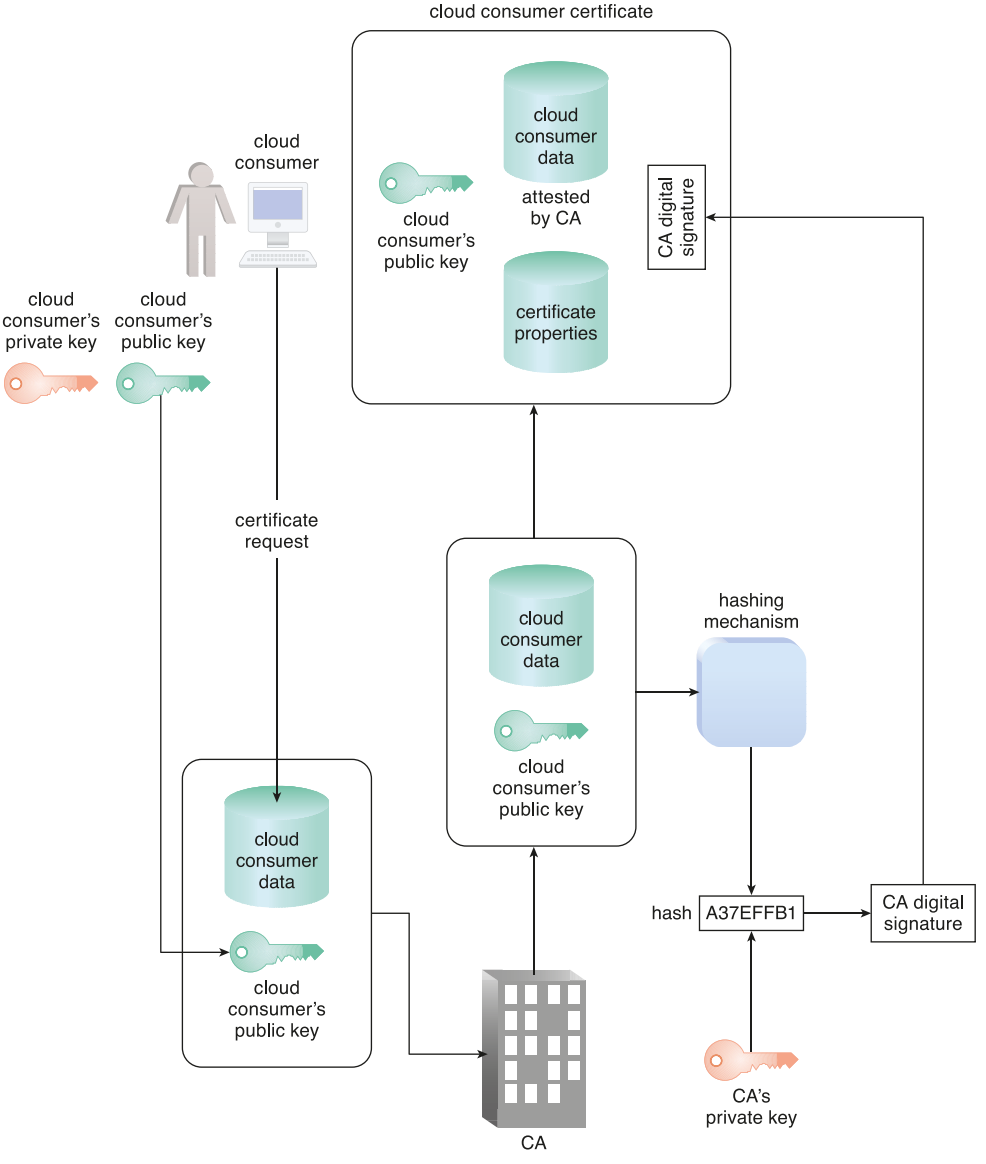


Figure 10.9

The common steps involved during the generation of certificates by a certificate authority.

CASE STUDY EXAMPLE

DTGOV requires that its clients use digital signatures to access its web-based management environment. These are to be generated from public keys that have been certified by a recognized CA (Figure 10.10).

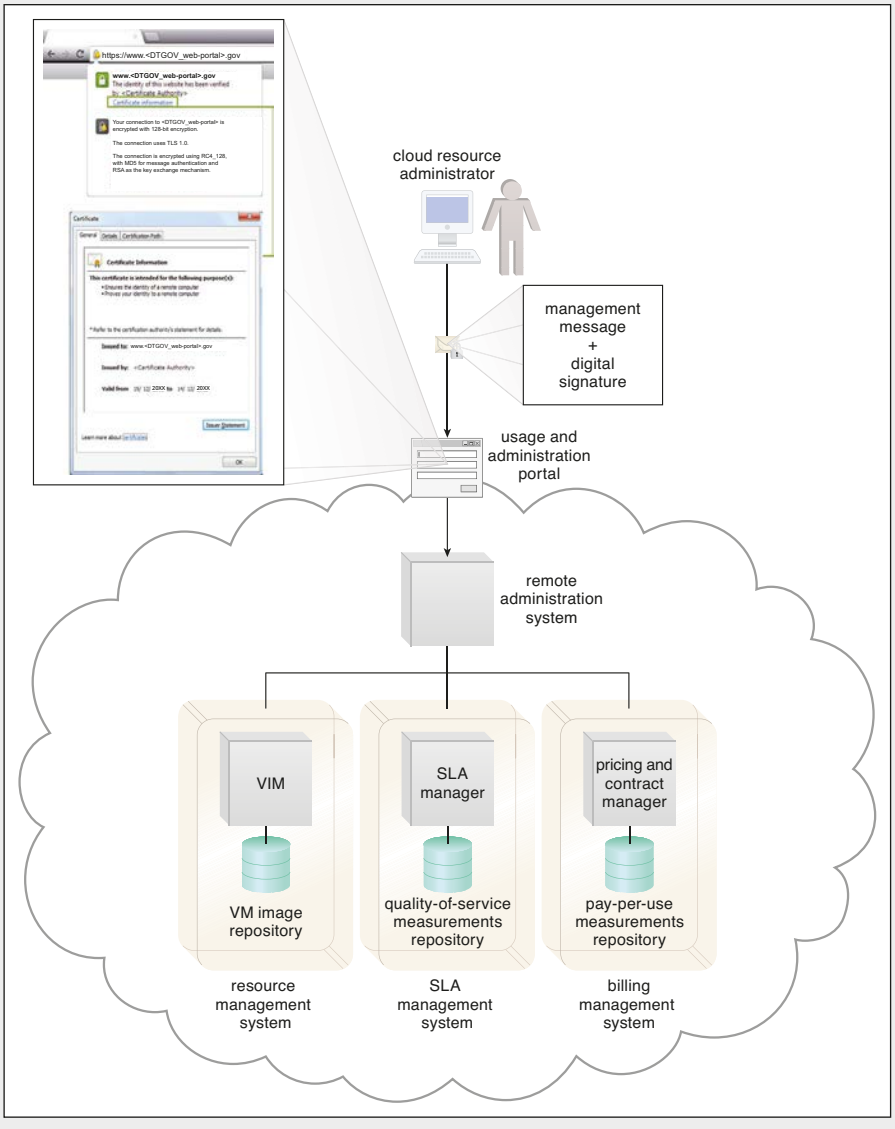


Figure 10.10

An external cloud resource administrator uses a digital certificate to access the web-based management environment. DTGOV's digital certificate is used in the HTTPS connection and then signed by a trusted CA.

10.6 Single Sign-On (SSO) System

Propagating the authentication and authorization information for a cloud service consumer across multiple cloud services can be a challenge, especially if numerous cloud services or cloud-based IT resources need to be invoked as part of the same overall runtime activity. The *single sign-on (SSO) system* mechanism enables one cloud service consumer to be authenticated by a security broker, which establishes a security context that is persisted while the cloud service consumer accesses other cloud services or cloud-based IT resources. Otherwise, the cloud service consumer would need to reauthenticate itself with every subsequent request.

The SSO system mechanism essentially enables mutually independent cloud services and IT resources to generate and circulate runtime authentication and authorization credentials. The credentials initially provided by the cloud service consumer remain valid for the duration of a session, while its security context information is shared (Figure 10.11). The SSO system mechanism's security broker is especially useful when a cloud service consumer needs to access cloud services residing on different clouds (Figure 10.12).

The SSO system mechanism does not directly counter any of the cloud security threats listed in Chapter 7. It primarily enhances the usability of cloud-based environments for access and management of distributed IT resources and solutions.

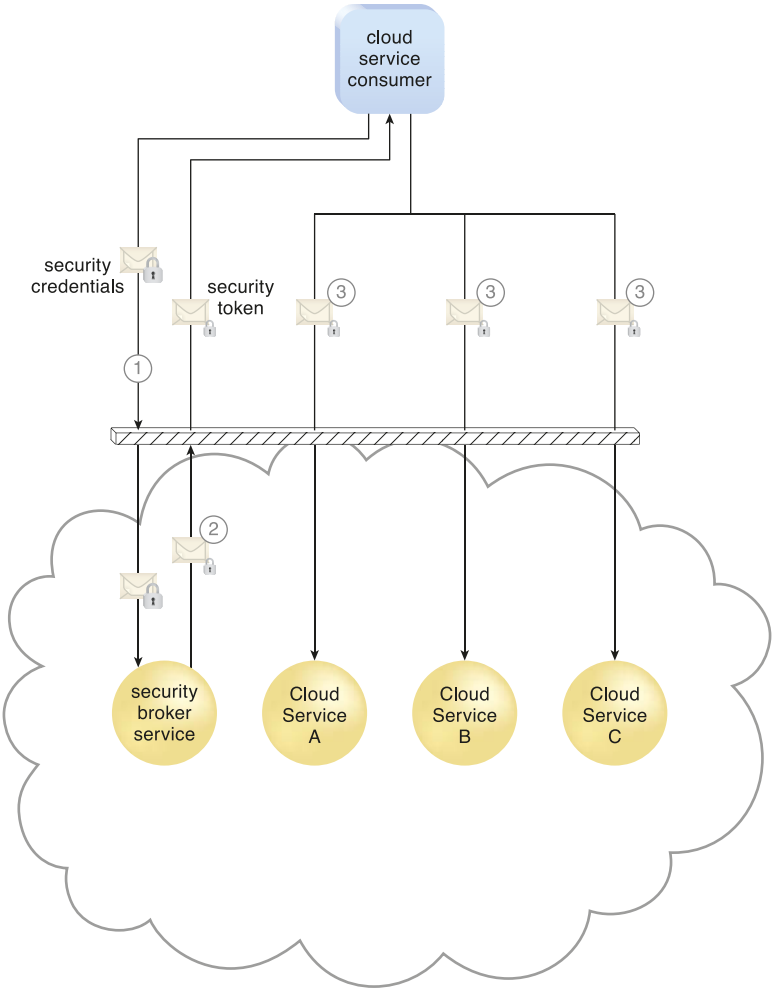


Figure 10.11

A cloud service consumer provides the security broker with login credentials (1). The security broker responds with an authentication token (message with small lock symbol) upon successful authentication, which contains cloud service consumer identity information (2) that is used to automatically authenticate the cloud service consumer across Cloud Services A, B, and C (3).

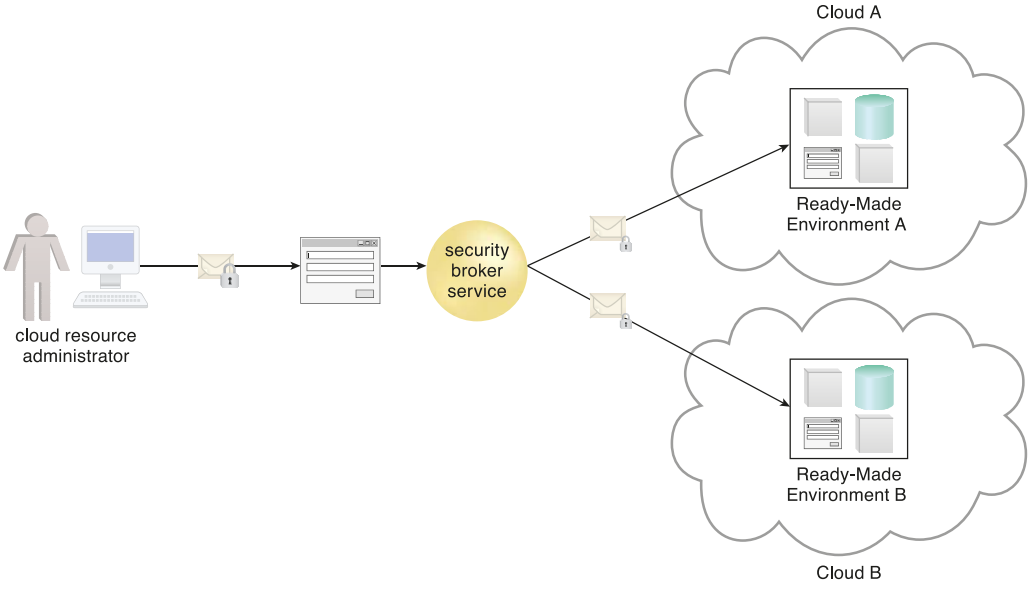


Figure 10.12

The credentials received by the security broker are propagated to ready-made environments across two different clouds. The security broker is responsible for selecting the appropriate security procedure with which to contact each cloud.

CASE STUDY EXAMPLE

The migration of applications to ATN's new PaaS platform was successful, but also raised a number of new concerns pertaining to the responsiveness and availability of PaaS-hosted IT resources. ATN intends to move more applications to a PaaS platform, but decides to do so by establishing a second PaaS environment with a different cloud provider. This will allow them to compare cloud providers during a three-month assessment period.

To accommodate this distributed cloud architecture, the SSO system mechanism is used to establish a security broker capable of propagating login credentials across both clouds (Figure 10.12). This enables a single cloud resource administrator to access IT resources on both PaaS environments without having to log in separately to each one.

10.7 Hardened Virtual Server Image

As previously discussed, a virtual server is created from a template configuration called a virtual server image (or virtual machine image). Hardening is the process of stripping unnecessary software from a system to limit potential vulnerabilities that can be exploited by attackers. Removing redundant programs, closing unnecessary server ports, and disabling unused services, internal root accounts, and guest access are all examples of hardening.

A *hardened virtual server image* is a template for virtual service instance creation that has been subjected to a hardening process (Figure 10.13). This generally results in a virtual server template that is significantly more secure than the original standard image.

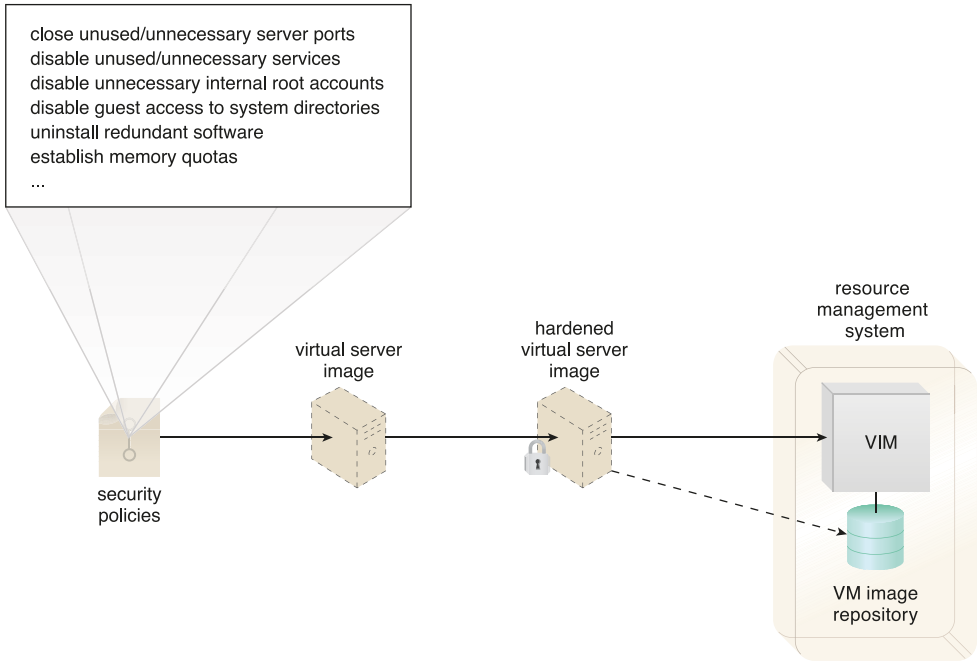


Figure 10.13

A cloud provider applies its security policies to harden its standard virtual server images. The hardened image template is saved in the VM images repository as part of a resource management system.

Hardened virtual server images help counter the denial of service, insufficient authorization, and overlapping trust boundaries threats.

CASE STUDY EXAMPLE

One of the security features made available to cloud consumers as part of DTGOV's adoption of cloud-based security groups is an option to have some or all virtual servers within a given group hardened (Figure 10.14). Each hardened virtual server image results in an extra fee but spares cloud consumers from having to carry out the hardening process themselves.

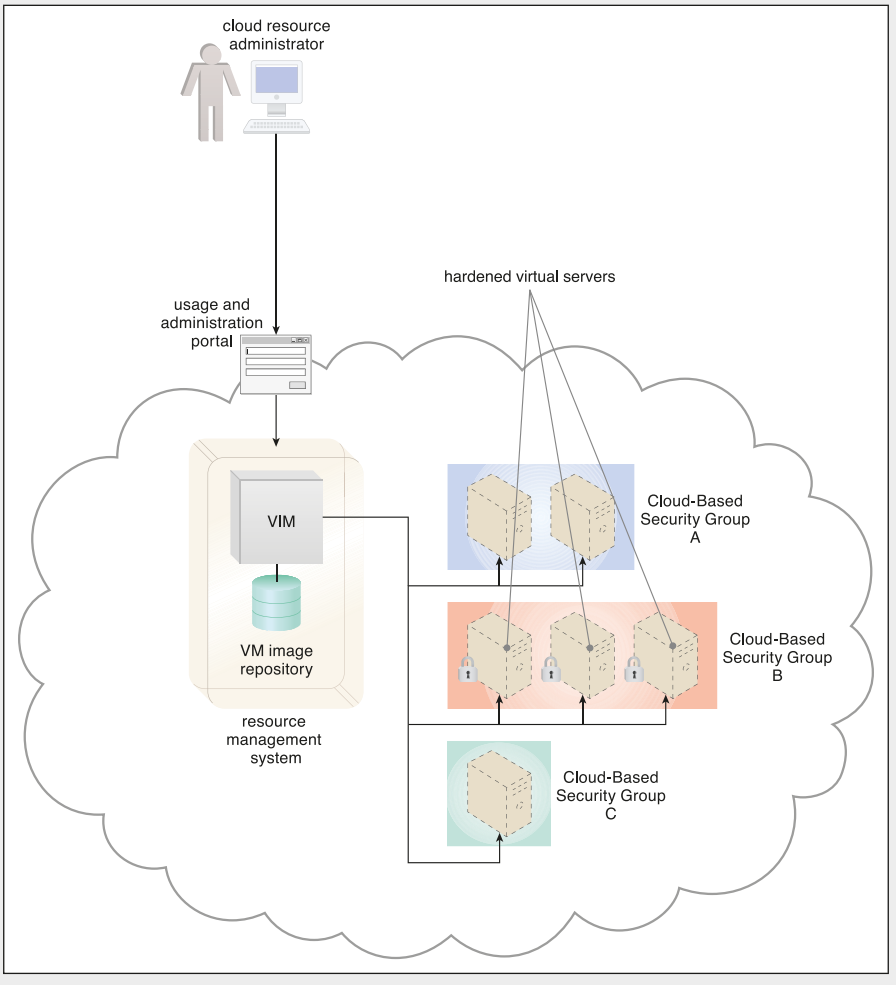


Figure 10.14
The cloud resource administrator chooses the hardened virtual server image option for the virtual servers provisioned for Cloud-Based Security Group B.

10.8 Firewall

A *firewall* is a network gateway that limits access between networks in accordance with an established security policy. It acts as the interface of a network to one or more external networks and regulates the network traffic passing through it by accepting or rejecting packets in accordance with a set of criteria. Both physical and virtual firewalls exist (Figure 10.15).

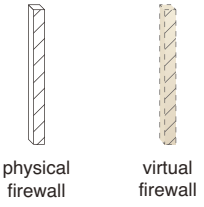
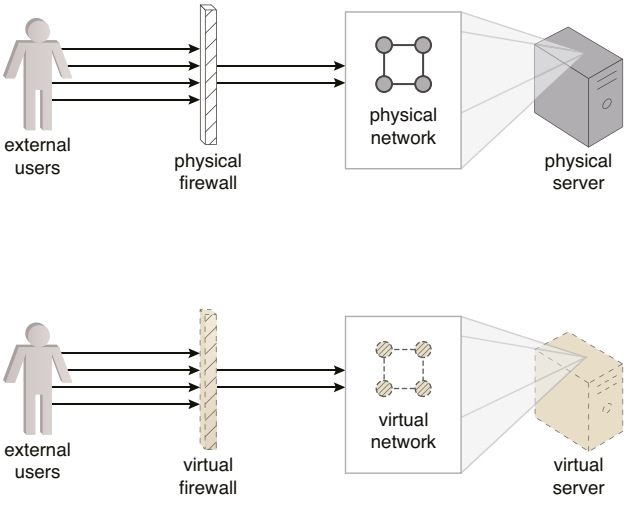


Figure 10.15
The icons used to represent physical and virtual firewalls.

Firewalls are used to protect the attack surface of an organization by intercepting all traffic that goes in and out and identifying whether any of that traffic matches predefined rules that can be preconfigured to control the traffic flow.

A physical firewall protects physical connections to network devices. However, physical firewalls are not capable of filtering out traffic that belongs to virtual networks, which exist only within virtualized hosts in networking environments that are not represented by physical connections between physical devices. In such an environment, a virtual firewall can be deployed to provide the same kind of protection for the virtual network (Figure 10.16). It is very common for virtual and physical firewalls to be integrated so as to provide coordinated protection that encompasses physical and virtual networks.

Figure 10.16
A physical firewall filters traffic for a physical network, whereas a virtual firewall filters traffic for a virtual network.



Some firewall implementations will also rely on firewall agents, which are programs deployed to run on individual software programs. These agents can provide a more individualized level of protection. A firewall with agents can be referred to as a *distributed firewall* because the overall firewall capabilities are provided collectively by the central firewall and its agents.

NOTE

Contemporary firewall products can encompass the capabilities of other mechanisms, such as features from the intrusion detection system (IDS) and digital virus scanning and decryption system. Some firewall products utilize data science technologies, such as machine learning and artificial intelligence (AI), to enable the firewall to evolve in its ability to protect network traffic.

CASE STUDY EXAMPLE

As part of DTGOV's cloud migration strategy, deploying virtual firewalls as part of every individual client network is fundamental to ensuring that all of them are protected against unauthorized access not only from the internet but also from each other. A virtual firewall for each client will allow DTGOV to customize network access individually as required by each different government organization.

10.9 Virtual Private Network (VPN)

A *virtual private network (VPN)* (Figure 10.17) is a mechanism that exists as an encrypted connection that allows remote users to access devices on a firewall-protected network. This mechanism provides a secure communications tunnel for data to be transmitted between networks. It is commonly used to establish an encrypted extension of a private network across an untrusted network, such as the internet. VPNs are implemented as virtual networks.

virtual private network (VPN)

Figure 10.17
The icon used to represent the virtual private network (VPN) mechanism.

VPNs protect access to internal information assets by allowing only authorized parties with the required security clearance to

remotely access data, while blocking other parties. VPNs are commonly built using cryptographic technologies to authenticate, authorize, and cipher all the traffic that passes through the VPN connection.

There are two types of VPNs:

- *Secure VPN* – This type of VPN sends and receives traffic in an encrypted and authenticated manner. Both the server and the client agree on security properties, and no one outside of the VPN can modify these agreed-upon properties.
- *Trusted VPN* – This type of VPN may not use encryption, but instead, users trust the VPN provider to ensure that no one else is using the same IP address in the pathway of that VPN. In a trusted VPN, only the provider can change, inject, or delete data in the VPN's communication channel.

Hybrid VPNs exist that combine the encryption property of a secure VPN and the dedicated connection property of a trusted VPN.

NOTE

Common VPN protocols include Open VPN, L2TP/IPSec, SSTP, IKEv2, PPTP, and Wireguard. Each offers different levels of speed, security, and ease of setup.

CASE STUDY EXAMPLE

DTGOV has identified certain client government organizations that need to be able to access protected data stored in cloud-based storage servers from remote locations in a secure manner. Dedicated physical connections are not available everywhere, requiring many of their clients to use the internet to access such data. By implementing VPN connections, DTGOV can guarantee secure access to the protected data via the internet.

10.10 Biometric Scanner

Biometrics is a technology used to determine a person's identity based on their physiological or behavioral characteristics. Since biometric data is directly derived from these types of unique user characteristics, it cannot be lost or forgotten by the user, nor can it be easily forged by attackers. This overcomes some of the problems users may have with passwords and tokens, which can be lost, forgotten, stolen, or otherwise compromised by attackers.

A *biometric scanner* (Figure 10.18) is a mechanism capable of validating a human's identity by scanning or capturing a physiological or behavioral characteristic, such as handwriting, signatures, fingerprints, eyes, voice, or facial recognition.

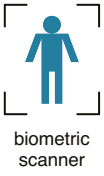


Figure 10.18
The icon used to represent the biometric scanner mechanism.

There are two primary types of identifiers that can be validated using biometric scanners:

- *Physiological Identifiers* – These can be either biological or morphological. Biological identifiers include DNA, blood, saliva, and urine tests, which are commonly used by medical teams and police forensics and do not really apply to cybersecurity protection mechanisms. Morphological identifiers include fingerprints, hand shape or vein pattern, eyes (including iris and retina), and face shape.
- *Behavioral Identifiers* – These include voice recognition, signature dynamics (including the speed of movement of the pen, accelerations, pressure exerted, and inclination), keystroke dynamics, the way we use certain objects, gait (the sound of a person's steps when they walk), and other types of gestures.

Different types of identifiers and measurements do not always have the same level of reliability. Physiological measurements usually offer the benefit of staying stable throughout the lifetime of a person and are not subjected to stress, whereas behavioral measurements can change with different life stages and stress levels (Figure 10.19).

Some biometric scanner mechanisms combine different types of biometric scanners to increase the range of security validations and the accuracy of identifications. These types of systems are referred to as *multimodal biometric scanners*, and they require at least two biometric credentials to perform identification. For example, a multimodal biometric scanner system may require both facial and fingerprint recognition to validate a user.

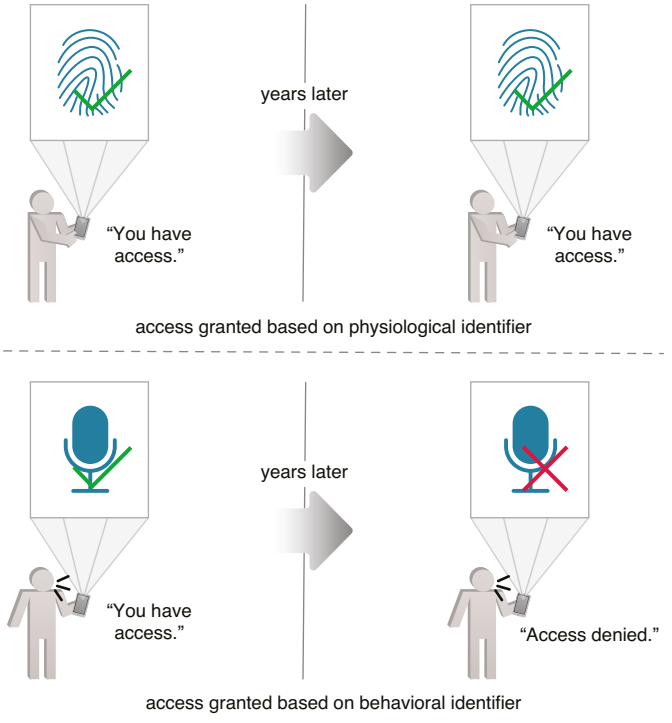


Figure 10.19
Over time, a person's fingerprint will not normally change, making it a reliable physiological identifier. However, a person's voice may change, making it a less reliable behavioral identifier.

Biometric scanners that are limited to verifying one identifier can also be referred to as unimodal biometric scanners.

CASE STUDY EXAMPLE

Recognizing how critical it is to ensure that only authorized guardians of children using their products should be allowed to access a cloud-based account, Innovartus decides to support the option for parents to require that access is only allowed via the use of a thumb scan. To enable this, Innovartus uses the biometric scanner mechanism and makes it available for users of mobile devices. This can help guarantee that only parents and other authorized guardians gain access to the cloud account that stores private data.

10.11 Multi-Factor Authentication (MFA) System

A *multi-factor authentication (MFA) system* (Figure 10.20) uses two or more factors (verifiers) to achieve authentication. It works by requesting one form of verification from a user during a sign-in process, and then requesting a second form of verification to complete the sign-in. The types of authentication methods are kept independent of each other, thereby making it difficult for malicious users to gain unauthorized access.

multi-factor authentication (MFA) system

Figure 10.20

The icon used to represent the multi-factor authentication (MFA) system mechanism.

Factors used in MFA systems typically include:

- something a user knows, such as a password or PIN (Figure 10.21)
- something a user has, such as a digital signature or token
- some part of a user, such as a biometric identifier or measurement

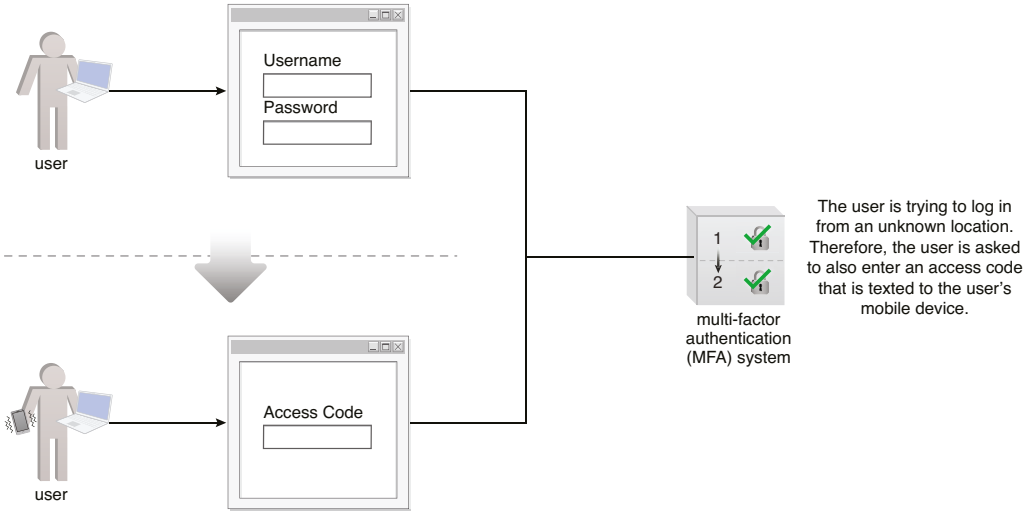


Figure 10.21

An MFA system is used to require multi-factor authentication steps by a user after it is determined that the user is attempting access from a new geographical location.

MFA systems may also support:

- *Location-Based Authentication* – A more advanced type of MFA that performs verification based on a user's IP address and geolocation.
- *Risk-Based Authentication* – A type of verification based on an analysis of context or behavior when a user is trying to access an account, such as when or from where the user is trying to sign in, whether sign-in is being performed by a known or new device, how many failed sign-in attempts have occurred, and so on. This is also known as *adaptive authentication*.

MFA systems are commonly used together with VPNs within organizations to enable employees to access corporate servers remotely.

CASE STUDY EXAMPLE

Some of the parents of children using Innovartus Technologies products have requested specific persons to access their children's accounts in the cloud on their behalf. To ensure that the alternate "guardian" is truly the one requesting access, Innovartus provides the option for cloud accounts to be only accessible via multi-factor authentication, such as a one-time password (OTP) sent via text to the authorized party's mobile device.

10.12 Identity and Access Management (IAM) System

The *identity and access management (IAM) system* mechanism encompasses the components and policies necessary to control and track user identities and access privileges for IT resources, environments, and systems.

Specifically, IAM system mechanisms exist as systems comprised of four main components:

- *Authentication* – Username and password combinations remain the most common forms of user authentication credentials managed by the IAM system, which can also support digital signatures, digital certificates, biometric hardware (fingerprint readers), specialized software (such as voice analysis programs), and locking user accounts to registered IP or MAC addresses.

- *Authorization* – The authorization component defines the correct granularity for access controls and oversees the relationships between identities, access control rights, and IT resource availability.
- *User Management* – Related to the administrative capabilities of the system, the user management program is responsible for creating new user identities and access groups, resetting passwords, defining password policies, and managing privileges.
- *Credential Management* – The credential management system establishes identities and access control rules for defined user accounts, which mitigates the threat of insufficient authorization.

Although its objectives are similar to those of the PKI system mechanism, the IAM system mechanism's scope of implementation is distinct because its structure encompasses access controls and policies in addition to assigning specific levels of user privileges.

The IAM system mechanism is primarily used to counter the insufficient authorization, denial of service, overlapping trust boundaries threats, virtualization attack, and containerization attack threats.

An IAM system (Figure 10.22) is an established mechanism used to identify, authenticate, and authorize users based on predefined user roles and access privileges.

Figure 10.22
The icon used to represent the identity and access management (IAM) system mechanism.

An IAM system can:

- verify users
- assign roles to users
- assign levels of access to users or groups of users (Figure 10.23)

An IAM system can carry out the identification, authentication, and authorization of a user by utilizing:

- *Unique Passwords* – Traditionally, the most common type of digital authentication an IAM system uses.
- *Pre-Shared Key (PSK)* – A type of digital authentication whereby the password is shared among users authorized to access the same IT resources. It provides convenience but is less secure than the use of individual passwords.

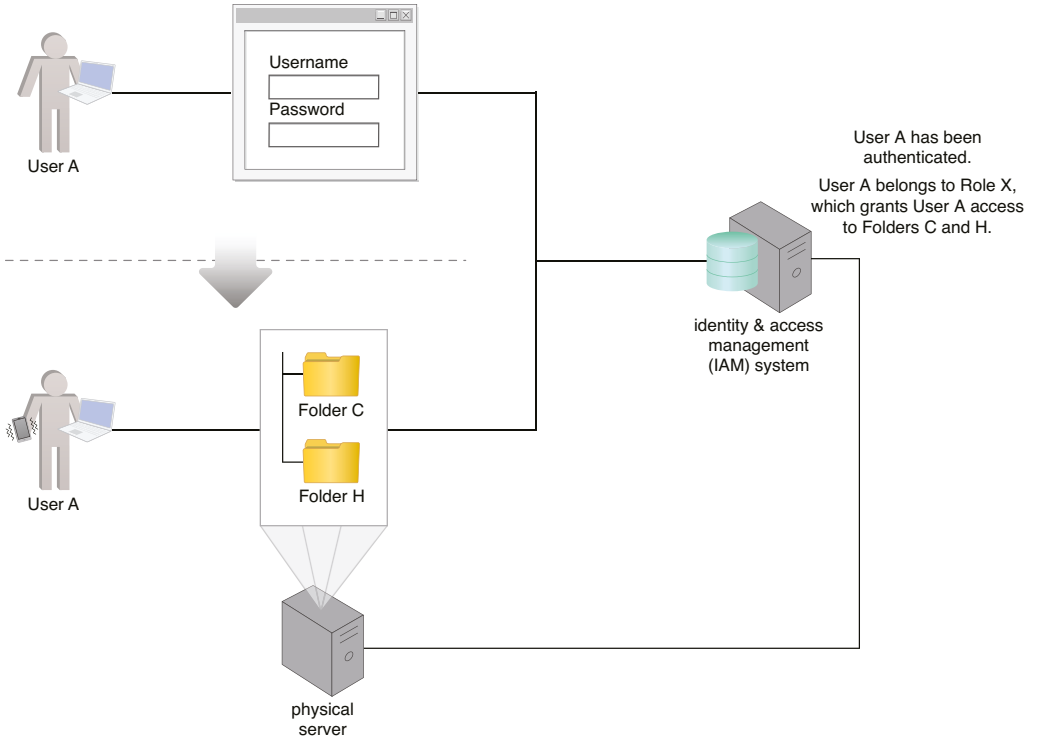


Figure 10.23

The IAM system authenticates User A and identifies the user as belonging to Role X. Based on the user's role, the IAM system authorizes the user to access two specific folders on a physical file server.

- *Behavioral Authentication* – For access to sensitive information or critical systems, the IAM can encompass or be used together with a biometric scanner to provide behavioral authentication. For example, it may analyze keystroke dynamics or mouse-use characteristics to instantly determine whether a user's sign-in behavior falls outside of the norm.
- *Other Biometrics* – IAM systems can use other biometric identifiers for more precise authentication.

Contemporary IAM systems can include AI technology to help assess user patterns and behaviors. The system may collect historical user access data that the AI system can use to learn about users and as a reference when comparing recent user behavior to historically recorded user behavior.

CASE STUDY EXAMPLE

As a result of several past corporate acquisitions, ATN's legacy landscape has become complex and heterogeneous. Maintenance costs have increased due to redundant and similar applications and databases running concurrently. Legacy repositories of user credentials are just as assorted.

Now that ATN has ported several applications to a PaaS environment, new identities are created and configured to grant users access. The CloudEnhance consultants suggest that ATN capitalize on this opportunity by starting a pilot IAM system initiative, especially since a new group of cloud-based identities is needed.

ATN agrees, and a specialized IAM system is designed specifically to regulate the security boundaries within their new PaaS environment. With this system, the identities assigned to cloud-based IT resources differ from corresponding on-premises identities, which were originally defined according to ATN's internal security policies.

10.13 Intrusion Detection System (IDS)

The *intrusion detection system (IDS)* mechanism (Figure 10.24) detects unauthorized or intrusion activity. It is the first line of defense for many networks. IDSs reference a database of known attack data to help recognize suspicious activity. Contemporary systems utilize machine learning and AI technology to help recognize activity associated with new attacks or being carried out by new attackers.

Figure 10.24
The icon used to represent the intrusion detection system (IDS) mechanism.

Based on the type of machine learning or AI technology used, different forms of intrusion detection can be carried out.

For example, an anomaly-based detection system works by creating a baseline for each information asset that represents a profile of "normal behavior." This profile considers usage bandwidth and other metrics for each device in the organization's attack surface, generating an alert for any activity that deviates from this baseline. Since each information asset is unique, these customized profiles can be created to make it more difficult for an attacker to know which specific activity can be carried out without setting off an alarm.

Features like this can help detect zero-day attacks, due to the fact that such a system does not depend on an established database of prior known intrusions, but instead focuses on the deviations from the established baselines.

There are two primary types of IDS mechanisms:

- *Passive* – The previously described scenario is an example of a passive IDS, as its main responsibilities are to detect intrusions and raise alerts.
- *Dynamic* – A dynamic IDS (also known as an intrusion detection and prevention system) is additionally designed to take action when a suspected intrusion has been detected.

In general, an intrusion detection and prevention system is a combination of a passive IDS and access control devices that the system executes to block an intruder.

CASE STUDY EXAMPLE

Given that secret or confidential data is not allowed outside of the secure perimeter of each client organization, some of DTGOV's law-enforcement clients have been attacked with the purpose of acquiring this type of data, such as data about open cases. Therefore, DTGOV decides to install an IDS that will help enable it to take immediate action whenever it is detected that attackers are attempting to penetrate a secure perimeter established by DTGOV for one of those clients.

10.14 Penetration Testing Tool

The *penetration testing tool* mechanism (Figure 10.25) is used to carry out penetration testing, also known as *pentesting*, which is the practice of testing a network or system to expose security vulnerabilities. It helps organizations understand the current capabilities of their cybersecurity environments, provides insight into which attacks can more successfully occur and allows for security professionals to carry out simulations of actual attacks.

Figure 10.25
The icon used to represent the penetration testing tool mechanism.

Contemporary attack vectors need to be examined thoroughly for potential vulnerabilities using modern and improved penetration testing techniques, which include:

- automated pentesting
- cloud-based pentesting
- social engineering pentesting (to evaluate how humans may react to threats, such as phishing)

Figure 10.26 illustrates several pentesting scenarios.

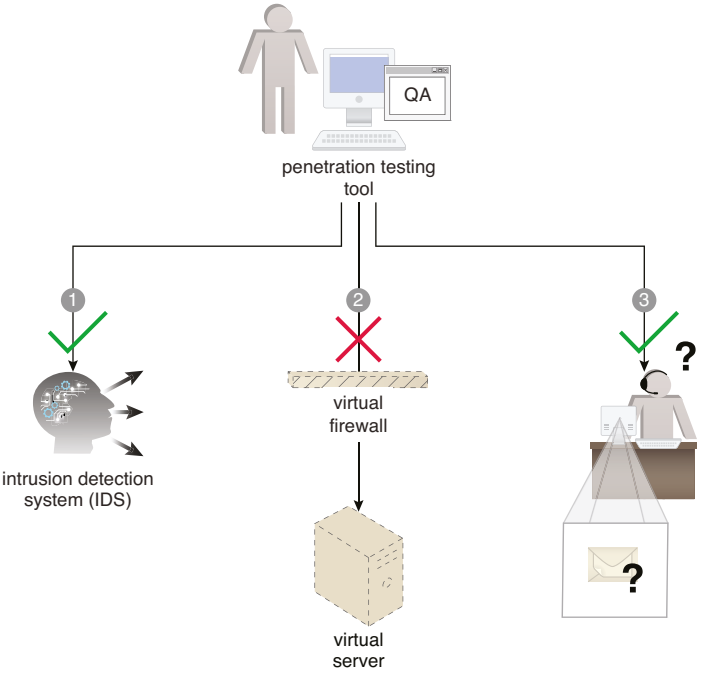


Figure 10.26
A security professional uses the penetration testing tool to verify that an intrusion detection system (IDS) (1) is working correctly. The penetration testing tool then exposes a vulnerability in a virtual firewall (2). Finally, it tries (unsuccessfully) to trick a human worker into opening a phishing email (3).

Penetration testing can be performed in a fully automated manner that allows for more frequent tests to be performed on the organization's security infrastructure. This can help establish a continuous assessment of a cybersecurity environment's effectiveness.

CASE STUDY EXAMPLE

DTGOV has implemented many security measures and controls to protect its clients' data and IT resources. However, the effectiveness of all those mechanisms, individually or as a group, has never been assessed. DTGOV employs a penetration testing tool to periodically test its security controls to ensure their effectiveness.

Specifically, the penetration testing tool is used in special exercises designed and scheduled to test and verify how well certain cloud security mechanisms are performing. This helps DTGOV make further adjustments and improvements in its security architecture.

10.15 User Behavior Analytics (UBA) System

A *user behavior analytics (UBA) system* (Figure 10.27) monitors the behavior of users in realtime to establish a baseline for "normal user activity" with the purpose of identifying abnormal user behavior that could indicate malicious activity. Monitored behaviors can include attempts to open, view, delete, and modify files; modifying critical system settings; and initiating network communications. A UBA system can block suspicious behaviors in realtime and/or terminate offending software. Some advanced UBA solutions focus on network and perimeter system activity such as logins and application- and system-level events. Others may focus on more granular metadata in the system itself, such as user activity on files and emails.

user behavior analytics (UBA) system

Figure 10.27
The icon used to represent the user behavior analytics (UBA) system mechanism.

A UBA system utilizes data science practices and technologies. The system needs to be trained to identify normal behavior by processing activity logs, file access, logins, network activity, and other types of historical activity. Through a variety of machine learning analysis techniques and the use of AI and neural networks, the system can establish a baseline from which it will be possible to predict what is and is not normal (Figure 10.28).

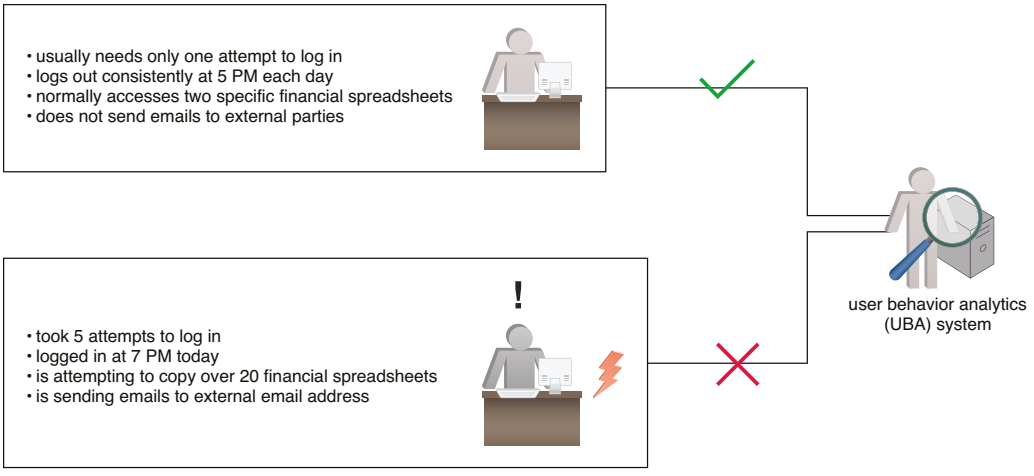


Figure 10.28

A UBA system detects suspicious activity when a user demonstrates abnormal behavior.

Other features of UBA systems include:

- *Processing High User Activity Volume* – File systems can be enormous and sensitive data can be spread out scarcely. To be able to identify attackers, a UBA system needs to be able to search through and analyze key metadata and activity of many users across potentially massive amounts of data.
- *Realtime Alerts* – A UBA system's attacker detection algorithms must be able to raise alerts in near realtime since the time window in which attackers access and copy sensitive data can be very short.

CASE STUDY EXAMPLE

DTGOV recognizes that its users cannot be effectively trained in cloud security awareness as quickly and effectively as it would want. It utilizes a UBA system that allows it to monitor and recognize user behavior to identify when a given user might actually be an attacker or intruder.

The UBA system analyzes the behavior of all of DTGOV's clients' end users and learns their common behavior in preparation for any unauthorized user that may attempt to use an account that belongs to a legitimate user.

10.16 Third-Party Software Update Utility

Cybersecurity-related software vulnerabilities commonly show up after a new version of third-party software has been released. When this happens, developers try to patch the vulnerability as fast as possible by releasing an upgrade or a patch that needs to be applied on all implementations of that software to remediate the encountered vulnerability. The longer it takes for the systems administrator to apply the update or patch, the higher the probability of being attacked through said vulnerability. The *third-party software update utility* mechanism (Figure 10.29) can help administrators automate the process of patching or updating their third-party software programs.

Figure 10.29
The icon used to represent the third-party software update utility mechanism.

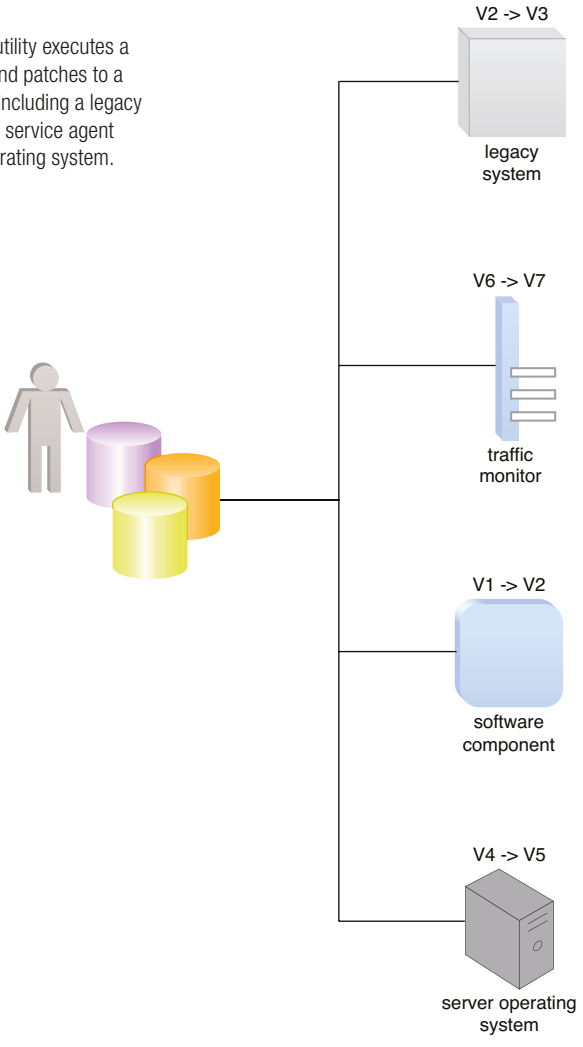
This mechanism typically works as follows:

- The administrator defines a baseline that determines the level of update and patching that is required.
- All related third-party software programs are reviewed against this baseline and the required updating and patching for each are identified.
- Patches and updates are downloaded from a central repository, usually through a secure channel to ensure that the software has not been tampered with. They are stored locally for further remediation purposes.
- The remediation process (the updating, upgrading, or patching activity that is performed automatically by the tool) is scheduled and/or carried out when required (Figure 10.30).

NOTE

This mechanism is specific to third-party software programs. Custom-developed software and applications can be more effectively updated and patched by an organization's development team using methodologies such as DevOps.

Figure 10.30
The third-party software update utility executes a series of prescheduled updates and patches to a number of third-party programs, including a legacy system, a software component, a service agent program, and a virtual server operating system.



CASE STUDY EXAMPLE

DTGOV manages a very large amount of cloud resources, including thousands of virtual servers with operating systems that need to be updated periodically to ensure that any vulnerabilities have been patched as soon as they are fixed by the operating system developer. Performing this task manually is not feasible due to the sheer amount of cloud-based virtual servers that have to be updated periodically.

DTGOV enlists the use of a third-party software update utility for each different operating system installed on the virtual servers it manages. This allows it to ensure that the operating systems are updated with all necessary security vulnerability patches and fixes as soon as they become available.

10.17 Network Intrusion Monitor

The *network intrusion monitor* mechanism (Figure 10.31) is dedicated to monitoring network packets across different subnetworks in order to find any suspicious activity. It reports back its findings to a centralized network IDS that coordinates its activity.

This mechanism can be signature-based or anomaly detection-based. The former is reactive, while the latter is proactive, autonomous, and can be configured to respond automatically to identified threats.

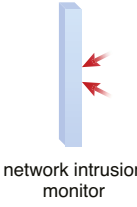


Figure 10.31
The icon used to represent the network intrusion monitor mechanism.

CASE STUDY EXAMPLE

As a provider of networking equipment itself for the telecommunications industry, ATN is concerned about the security of the virtual networks that connect all of its cloud-based resources. ATN knows how networks can be breached and wants to ensure that if that happens to its own cloud-based network, ATN can take immediate action to deal with the intruder appropriately.

ATN recognizes in the network intrusion monitor a mechanism that will notify the interested parties within the organization when a network has been breached. It will even provide sufficient information about the breach to allow IT security specialists from ATN to respond in time and avert all potential damages to the organization.

10.18 Authentication Log Monitor

The *authentication log monitor* mechanism (Figure 10.32) scans historical logs that include information about authentication events that occurred when users attempted to access protected network resources. This information may be used to solve access difficulties and to change authentication policy rules.

Among the data collected by this monitor is also authentication rule data, such as timeouts that represent the period of time during which a user can access a resource after first being authenticated for its access.

Figure 10.32
The icon used to represent the authentication log monitor mechanism.

CASE STUDY EXAMPLE

DTGOV needs to manage access for a very large number of users. This is a burdensome responsibility that has been performed manually by the administrators of the different cloud-based resources that DTGOV manages on behalf of its clients. Manual data entry has been prone to human error, which has resulted in complaints about possible unauthorized access to users' accounts.

DTGOV decided to use an authentication log monitor to regularly analyze access-related information for users that complain about their access privileges being used improperly. This helps them determine when actual intrusion events may have occurred. With this information, DTGOV can proceed to review the access privileges given to affected users to see if access was carried out in compliance with the originally requested privileges.

10.19 VPN Monitor

A *VPN monitor* (Figure 10.33) tracks and collects information about VPN connections, such as which users have connected (or are currently connected), the kinds of connections used, and the volume of data exchanged over a certain period. In case of failed connection attempts, it records connection issues and sends notifications to administrators. This mechanism helps identify network anomalies.

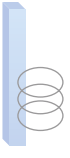


Figure 10.33
The icon used to represent the VPN monitor mechanism.

CASE STUDY EXAMPLE

Several of DTGOV's clients that allow remote access to their cloud-based data and systems via a VPN have complained that their data may have been accessed by unauthorized parties. To be able to verify this behavior, DTGOV uses a VPN monitor. DTGOV analyzes the information collected by the VPN monitor to identify potential unauthorized access through the VPN.

10.20 Additional Cloud Security Access-Oriented Practices and Technologies

The following is a list of further third-party cloud security access-oriented practices and technologies:

- *Cloud Access Security Brokers (CASB)* – Security solutions designed to protect cloud-based applications and services. These solutions are typically deployed between cloud service consumers and providers, allowing organizations to enforce security policies and gain visibility into cloud usage.
- *Secure Access Service Edge (SASE)* – A network architecture that combines network security and wide-area networking capabilities to deliver secure access to cloud-based applications and resources.
- *Cloud Security Posture Management (CSPM)* – A cloud security solution that provides continuous monitoring and management of an organization's cloud infrastructure to ensure compliance with security policies and regulations.
- *Cloud Workflow Protection Platforms (CWPP)* – A type of cloud security tool that is designed to protect and secure the various workflows and processes that occur within cloud-based environments. These platforms help to ensure that these workflows are safe from unauthorized access, data breaches, and other security threats.
- *Cloud Infrastructure Entitlement Management (CIEM)* – A type of security solution designed to manage and monitor access to cloud resources, such as servers, databases, and applications. CIEM helps organizations ensure that only authorized personnel have access to their cloud infrastructure, reducing the risk of data breaches and unauthorized modifications. This solution provides visibility into user access permissions and activity, allowing organizations to detect and respond to suspicious behavior in realtime.

Chapter 11

Cloud Security and Cybersecurity Data-Oriented Mechanisms

This chapter describes the following mechanisms that are focused on establishing data access controls and cloud data monitoring functions.

- Digital Virus Scanning and Decryption System
- Malicious Code Analysis System
- Data Loss Prevention (DLP) System
- Trusted Platform Module (TPM)
- Data Backup and Recovery System
- Activity Log Monitor
- Traffic Monitor
- Data Loss Protection Monitor

11.1 Digital Virus Scanning and Decryption System

The *digital virus scanning and decryption system* (Figure 11.1) is essentially an advanced antivirus system comprised of client-side and server-side components. The client-side component detects viruses by scanning files using detection methods that include specific pattern matches within executable files or heuristic methods to detect viral activity. It attempts to clean an identified virus infection by removing the virus's code and restoring the original file's contents.

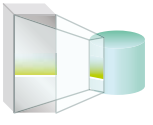


Figure 11.1
The icon used to represent the digital virus scanning and decryption system mechanism.

The server-side component is responsible for maintaining a database of collected virus information and using data science technologies to analyze and learn from the available information to help identify and counter new potential viruses or virus variants. The client-side component periodically receives updated intelligence from the server-side component to improve its ability to detect and remove viruses.

The digital virus scanning and decryption system commonly also provides the following features.

Generic Decryption

This feature enables the system to detect highly complex viruses while maintaining fast scanning speeds. Executable files are run through a generic decryption scanner, which consists of three fundamental elements:

- *CPU Emulator* – A software-based virtual computer where an executable file is run rather than executing it on the underlying processor.
- *Virus Signature Scanner* – Software that scans the executable file looking for known virus signatures.
- *Emulation Control Module* – Software that controls the execution of the executable file.

Digital Immune System

This feature enables the system to capture a virus, strip it of confidential information, and then automatically submit it to a central virus analysis center, where the virus is examined and a virus signature is created. The virus signature is then tested against the original sample, and if successful, it is sent back to the server to be deployed on the client-side mechanism component (Figure 11.2).

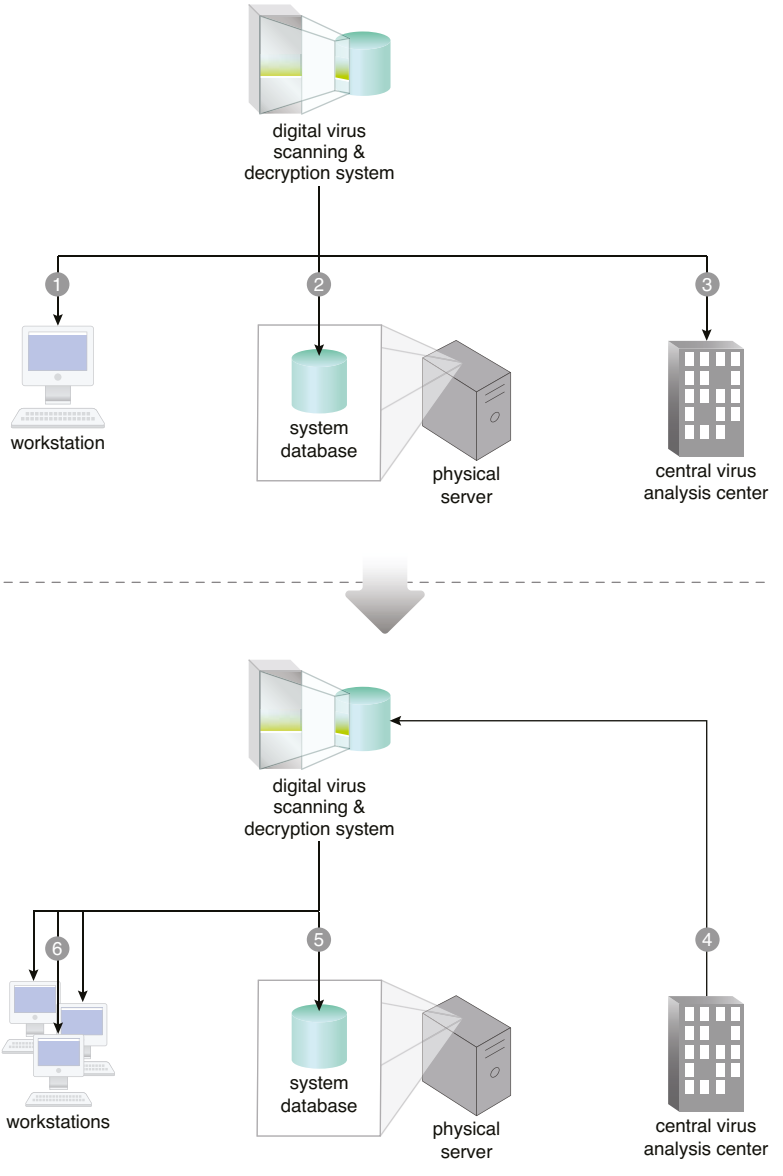


Figure 11.2

The client-side component of the digital virus scanning and decryption system detects a virus on a workstation (1). The server-side component logs the virus information in a central database (2) and further forwards it to a central virus analysis center (3), where it is assigned a virus signature. The virus signature is returned (4), logged by the server-side component (5), and distributed to all workstations under the system's protection (6).

CASE STUDY EXAMPLE

Numerous users working for DTGOV's clients connect to systems and IT resources that DTGOV deploys and manages in the cloud on their behalf. Unfortunately, several viruses have successfully attacked parts of that infrastructure since the migration to the cloud first began.

DTGOV employs a digital virus scanning and decryption system as part of their new cloud security defense strategy. This system significantly reduces the spread of viruses throughout its cloud-based resources.

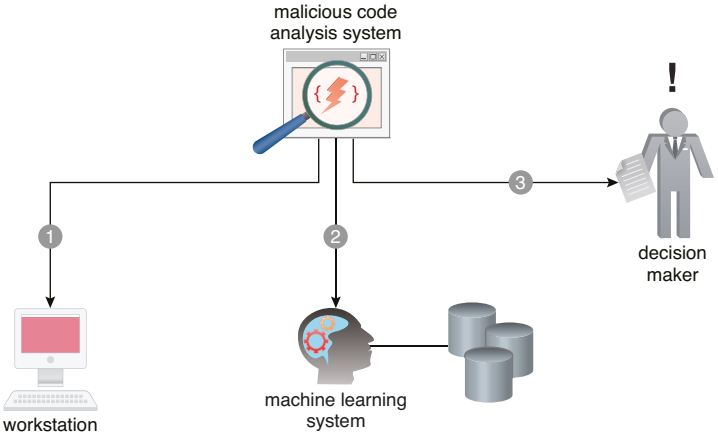
11.2 Malicious Code Analysis System

The *malicious code analysis system* (Figure 11.3) is a mechanism that is able to perform analysis of massive volumes of malicious code quickly and produce a report that a human analyst can use to determine which actions the malicious code took. Contemporary malicious code analysis systems rely on machine learning technology to carry out and constantly improve their malware detection capabilities.

Figure 11.3
The icon used to represent the malicious code analysis system mechanism.

The large processing capabilities of these systems enable them to accelerate security investigations using detailed data about workload-related events, application logs, infrastructure metrics, audits, and other sources of malicious code behavior information. The malicious code analysis system is further able to issue alerts about malicious or anomalous patterns (Figure 11.4).

Figure 11.4
An automated malicious code analysis system detects malicious code on a workstation (1) and analyzes the code in realtime (2) to alert and provide a report to a security professional to review (3).



The use of this mechanism can help an organization defend against zero-day attacks since the intelligence gathered is not necessarily based on historical intrusion detection, but rather on the results of data analysis provided by models capable of identifying new malware in realtime.

There are two primary types of malicious code analysis systems:

- *Static* – This type of system is capable of executing malicious code in a safe and isolated environment called a sandbox—a controlled environment that enables security professionals to watch the malware in action without risking its potential effects on the organization's business environment.
- *Dynamic* – This type of system can provide deep insight into the capabilities of malicious code. It utilizes automated sandboxing, which eliminates the time that it would take to reverse engineer a file after malicious code has performed actions on it.

Some attackers develop malicious programs that will remain dormant while running in a sandbox environment. Therefore, a hybrid combination of static and dynamic malicious code analysis systems can be used to provide a reliable means of detecting more sophisticated malicious code by hiding the presence of a sandbox.

CASE STUDY EXAMPLE

Innovartus has been the target of multiple different virus attacks and has therefore decided to implement a malicious code analysis system to prevent such attacks in the future.

This system has especially helped Innovartus identify the more sophisticated attacks that require specialized and profound code analysis to identify.

11.3 Data Loss Prevention (DLP) System

A *data loss prevention (DLP) system* (Figure 11.5) is a tool that enables security professionals to manage the security of and configure access to distributed information assets, which becomes more difficult the more remote the workforce. It is commonly used to avoid the unauthorized or accidental sharing of confidential data by internal staff.

Figure 11.5
The icon used to represent the data loss prevention (DLP) system mechanism.

The DLP mechanism's capabilities can include:

- *Device Control* – This allows the administrator to control which devices users can store or copy data on. For example, it can be used to block users from storing potentially confidential data on USB drives or SD cards.
- *Content-Aware Protection* – This allows the administrator to monitor and control files, emails, and other kinds of artifacts that can hold data, primarily to ensure that no confidential information can be extracted from them.
- *Data Scanning* – This function can scan files, emails, and digital documents across different devices to mark those that can be considered confidential. Information assets can be labeled as confidential for future reference by other mechanisms.
- *Forced Encryption* – This is used to ensure that any content that is allowed to leave the organization is encrypted to ensure that it will be accessed only by authorized parties.

Figure 11.6 demonstrates some of these capabilities.

DLP systems can exist as cloud-based services used to monitor cloud-based file-sharing applications and sites.

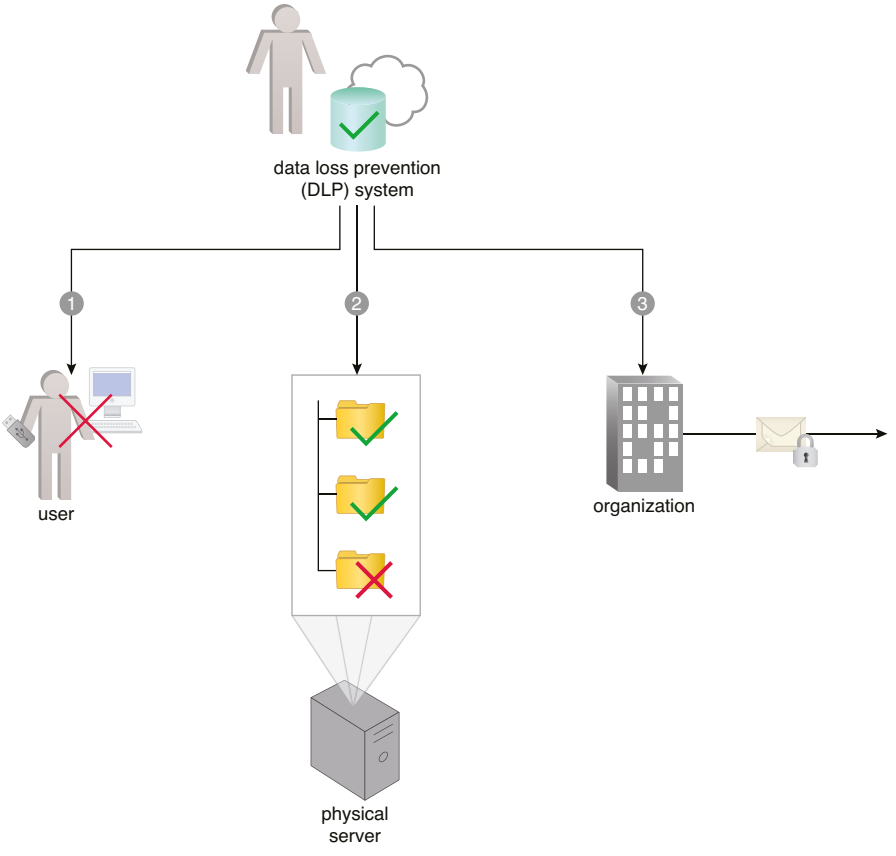


Figure 11.6

A security professional with a DLP system blocks a user from storing company data on a USB drive (1), scans a corporate server with files in folders to identify the ones with confidential data (2), and forces an email going outside of the organization boundary to be encrypted (3).

CASE STUDY EXAMPLE

Some of DTGOV's clients are law-enforcement government organizations that need to keep certain data confidential and secret. To prevent classified data from being shared in an unauthorized manner, a cloud-based data loss prevention system is established specifically for those clients.

This system ensures that any data copied or moved is checked to see if it is confidential or secret. If that is the case, the data will not be allowed outside of a specified perimeter within the client's cloud environment.

11.4 Trusted Platform Module (TPM)

A *trusted platform module (TPM)* (Figure 11.7) is a mechanism that stores artifacts that are used to authenticate devices ("platforms") such as PCs, laptops, mobile phones, and tablets. A TPM can exist as a chip that has a unique and secret key burned in during production.

trusted platform module (TPM)

Figure 11.7

The icon used to represent a trusted platform module (TPM) mechanism.

The TPM chip performs certain measurements each time the device starts. These measurements include taking hashes of the BIOS code, BIOS settings, TPM settings bootloader, and OS kernel so that alternative versions of the measured modules cannot be easily produced, and so that the hashes lead to identical measurements. These measurements are then used to validate against known good values.

During boot-up, the mechanism verifies the characteristics of the hardware components connected to the processor against the device information stored in the TPM. If they differ, then it is confirmed that the hardware has been compromised (Figure 11.8).

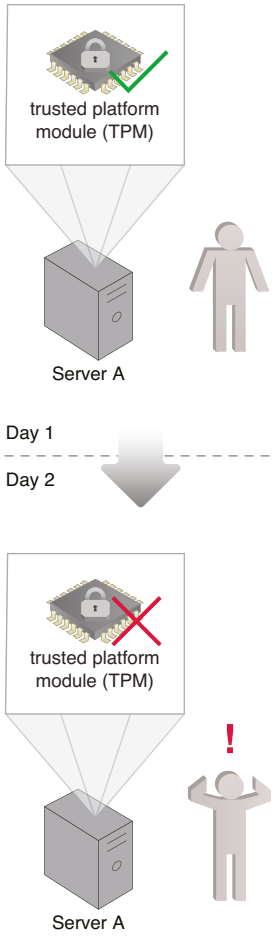


Figure 11.8

On Day 1, an administrator starts up a physical server. The TPM mechanism verifies that the hardware is okay. On Day 2, the administrator starts up the same server, only this time the TPM mechanism indicates that the hardware confirmation does not match its measurements. The administrator is made aware that the server could have been tampered with.

CASE STUDY EXAMPLE

The security of children while using the virtual toys provided by Innovartus is of critical importance to their parents. It is therefore important for Innovartus to ensure that no malicious code can run on any of its cloud-based virtual servers.

To achieve this, they install a TPM on each of the physical servers that host their cloud-based virtual servers. It uses the TPM to verify that the hypervisor and every operating system instance running on those servers is verified for authenticity before it is loaded into memory. This guarantees that no tampering with the physical hardware firmware or any other logic that runs on those servers can occur. This, in turn, helps eliminate the possibility of malware running together with their virtual toy products.

11.5 Data Backup and Recovery System

The *data backup and recovery system* (Figure 11.9) is a mechanism that is used to provide fast data recovery in the event of data loss or corruption resulting from cyber attacks, cyber theft, physical theft, or hardware or software failure.

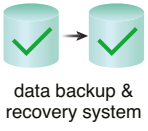


Figure 11.9
The icon used to represent a data backup and recovery system mechanism.

The data backup and recovery system essentially copies important data to separate storage repositories to provide a constant fallback from which an organization can recover data (Figure 11.10).

Many variations of this mechanism rely on placing backup data in clouds. Cloud providers often provide backup as a service (BaaS) offerings, which can simplify data backup and recovery because they do not require the installation and configuration of a storage device and additional software, such as an operating system.

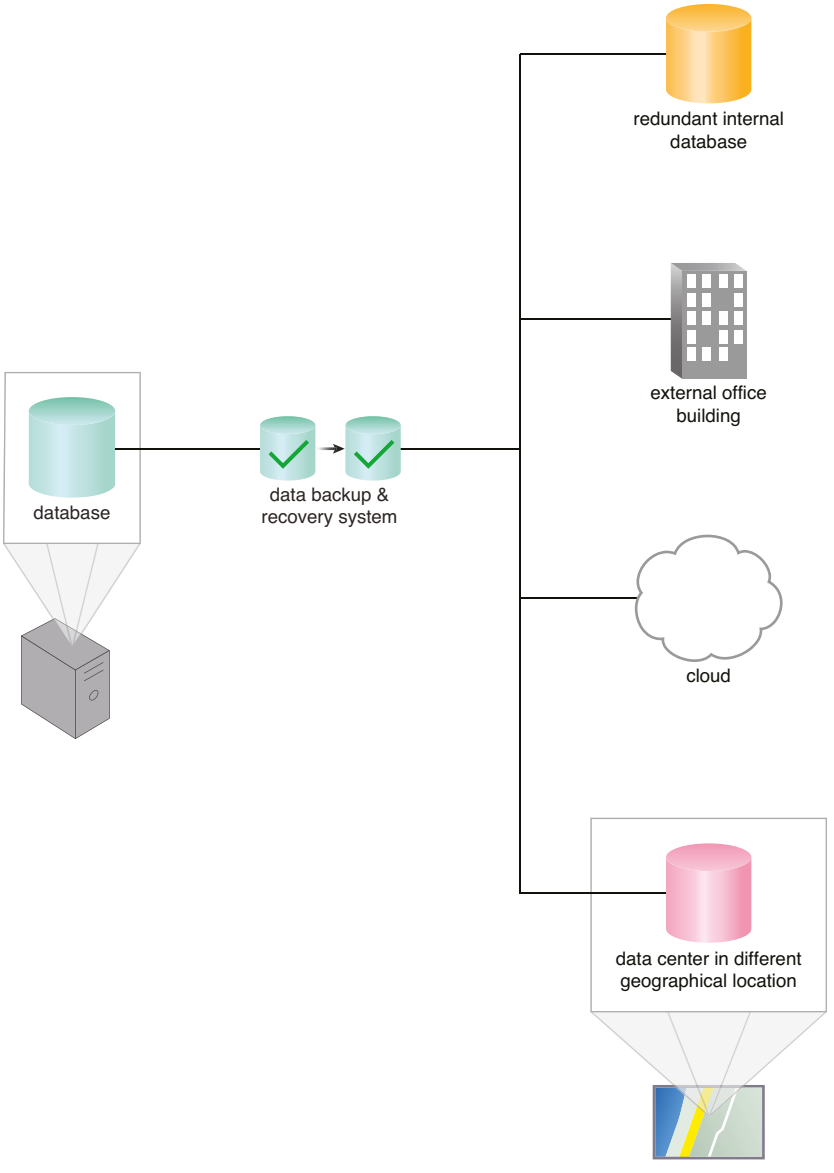


Figure 11.10

A common technique for using the data backup and recovery system mechanism is known as the "3-2-1 approach," which requires that data be kept in three separate locations, utilizing two different storage formats, and with one extra copy kept elsewhere, in a different geographical region.

CASE STUDY EXAMPLE

The sheer volume of data that DTGOV is storing and processing in the cloud on behalf of its many clients places a significant responsibility on DTGOV to ensure that the data is always made available to its clients, regardless of any failure conditions or disruptions that may occur in the cloud environment.

A data backup and recovery system helps DTGOV ensure that the most critical data it stores and processes on behalf of its clients is copied in a safe and available medium, in a location that is not subject to the same environmental or operational hazards to which the original data may be exposed. This way, in case the original data becomes unavailable, the copy can be used to restore that data.

11.6 Activity Log Monitor

The *activity log monitor* (Figure 11.11) scans historical log files or databases in an attempt to detect patterns of activity on the network that may indicate possible security breaches. Activity log data can come from event logs, device configuration logs, operating system logs, and other sources.

Figure 11.11
The icon used to represent the activity log monitor mechanism.

CASE STUDY EXAMPLE

When parents complain to Innovartus about possible unauthorized access to their cloud-based accounts, the company needs to be able to verify such claims.

For this purpose, they use an activity log monitor to search through all recorded access attempts, whether successful or not, to the account in question. This monitor provides information about any activity patterns that may indicate malicious behavior, which Innovartus can then study to verify the legitimacy of each complaint.

11.7 Traffic Monitor

The *traffic monitor* mechanism (Figure 11.12) is responsible for monitoring network traffic to review and analyze traffic activity in search of abnormalities that may be adversely affecting network performance, availability, and/or security. This mechanism provides network administrators with realtime data and long-term usage trends for network devices.

Figure 11.12
The icon used to represent the traffic monitor mechanism.

CASE STUDY EXAMPLE

Multiple types of security incidents trigger specific network-related events within the virtual networks that interconnect cloud-based resources. Therefore, as a complement to the network intrusion monitor, ATN installs a traffic monitor mechanism to gather data about the behavior of the network, which can be correlated with information from the network intrusion monitor to identify more specifically the type of intrusion or network breach that happened, allowing ATN to take the most effective action against the intrusion.

11.8 Data Loss Protection Monitor

A *data loss protection monitor* (Figure 11.13) is designed to safeguard vital data by utilizing capture technology that acts as a digital recorder and replays after-the-fact data loss incidents. These recordings can be used for subsequent investigations. This mechanism can streamline remediation by alerting senders, recipients, content owners, and system administrators to a breach.

The data loss protection monitor is commonly used to safeguard an organization's most important information assets, such as source code, internal memos, and patent applications.

Figure 11.13
The icon used to represent the data loss protection monitor mechanism.

It detects many different content types traversing any port or protocol to uncover unknown threats. The monitor can find and analyze sensitive information traveling across the network and apply rules to prevent future risks. This mechanism can further provide input for reports that explain who sent data, where it went, and how it was sent.

NOTE

A data loss protection monitor can help an organization meet data loss monitoring regulatory requirements, such as PCI, GLBA, HIPAA, and SOX.

CASE STUDY EXAMPLE

In support of the strict data requirements of its law enforcement clients, DTGOV relies on a data loss protection monitor to keep it informed when any activity, such as copying or moving of data, occurs that is not in compliance with their regulations and policies.

Chapter 12

Cloud Management Mechanisms

Cloud-based IT resources need to be set up, configured, maintained, and monitored. The systems covered in this chapter are mechanisms that encompass and enable these types of management tasks. They form key parts of cloud technology architectures by facilitating the control and evolution of the IT resources that form cloud platforms and solutions.

The following management-related mechanisms are described in this chapter:

- Remote Administration System
- Resource Management System
- SLA Management System
- Billing Management System

These systems typically provide integrated APIs and can be offered as individual products, custom applications, or combined into various product suites or multifunction applications.

12.1 Remote Administration System

The *remote administration system* mechanism (Figure 12.1) provides tools and user interfaces for external cloud resource administrators to configure and administer cloud-based IT resources.

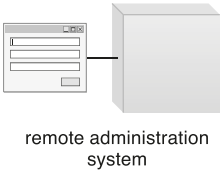


Figure 12.1
The symbol used in this book for the remote administration system. The displayed user interface will typically be labeled to indicate a specific type of portal.

A remote administration system can establish a portal for access to administration and management features of various underlying systems, including the resource management, SLA management, and billing management systems described in this chapter (Figure 12.2).

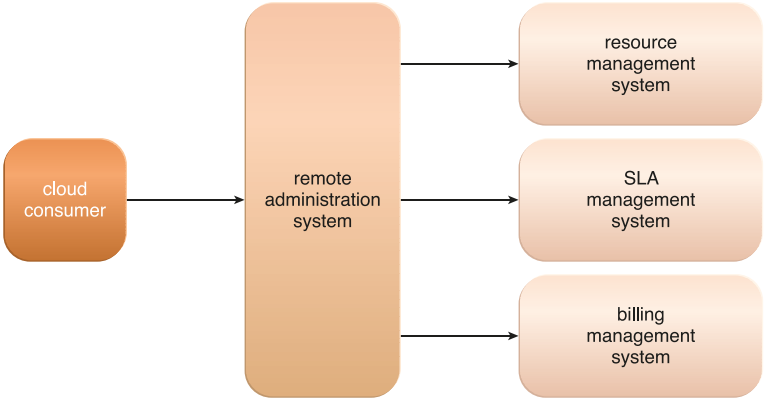


Figure 12.2

The remote administration system abstracts underlying management systems to expose and centralize administration controls to external cloud resource administrators. The system provides a customizable user console, while programmatically interfacing with underlying management systems via their APIs.

The tools and APIs provided by a remote administration system are generally used by the cloud provider to develop and customize online portals that provide cloud consumers with a variety of administrative controls.

The following are the two primary types of portals that are created with the remote administration system:

usage and administration portal

- *Usage and Administration Portal* – A general-purpose portal that centralizes management controls to different cloud-based IT resources and can further provide IT resource usage reports. This portal is part of numerous cloud technology architectures covered in Chapters 13 to 15.

self-service portal

- *Self-Service Portal* – This is essentially a shopping portal that allows cloud consumers to search an up-to-date list of cloud services and IT resources that are available from a cloud provider (usually for lease). The cloud consumer submits its chosen items to the cloud provider for provisioning. This portal is primarily associated with the rapid provisioning architecture described in Chapter 14.

Figure 12.3 illustrates a scenario involving a remote administration system and both the usage and administration and self-service portals.

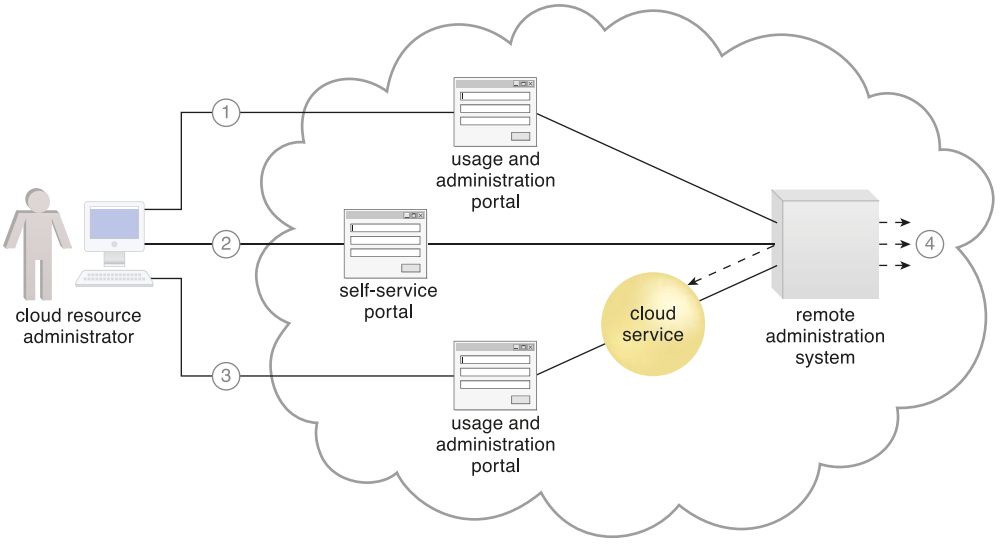


Figure 12.3

A cloud resource administrator uses the usage and administration portal to configure an already leased virtual server (not shown) to prepare it for hosting (1). The cloud resource administrator then uses the self-service portal to select and request the provisioning of a new cloud service (2). The cloud resource administrator then accesses the usage and administration portal again to configure the newly provisioned cloud service that is hosted on the virtual server (3). Throughout these steps, the remote administration system interacts with the necessary management systems to perform the requested actions (4).

Depending on:

- the type of cloud product or cloud delivery model the cloud consumer is leasing or using from the cloud provider,
- the level of access control granted by the cloud provider to the cloud consumer, and
- which underlying management systems the remote administration system interfaces with,

…the following tasks can commonly be performed by cloud consumers via a remote administration console:

- configuring and setting up cloud services
- provisioning and releasing IT resource for on-demand cloud services
- monitoring cloud service status, usage, and performance
- monitoring QoS and SLA fulfillment
- managing leasing costs and usage fees
- managing user accounts, security credentials, authorization, and access control
- tracking internal and external access to leased services
- planning and assessing IT resource provisioning
- capacity planning

While the user interface provided by the remote administration system will tend to be proprietary to the cloud provider, there is a preference among cloud consumers to work with remote administration systems that offer standardized APIs. This allows a cloud consumer to invest in the creation of its own front-end with the foreknowledge that it can reuse this console if it decides to move to another cloud provider that supports the same standardized API. Additionally, the cloud consumer would be able to further leverage standardized APIs if it is interested in leasing and centrally administering IT resources from multiple cloud providers and/or IT resources residing in cloud and on-premises environments (Figure 12.4).

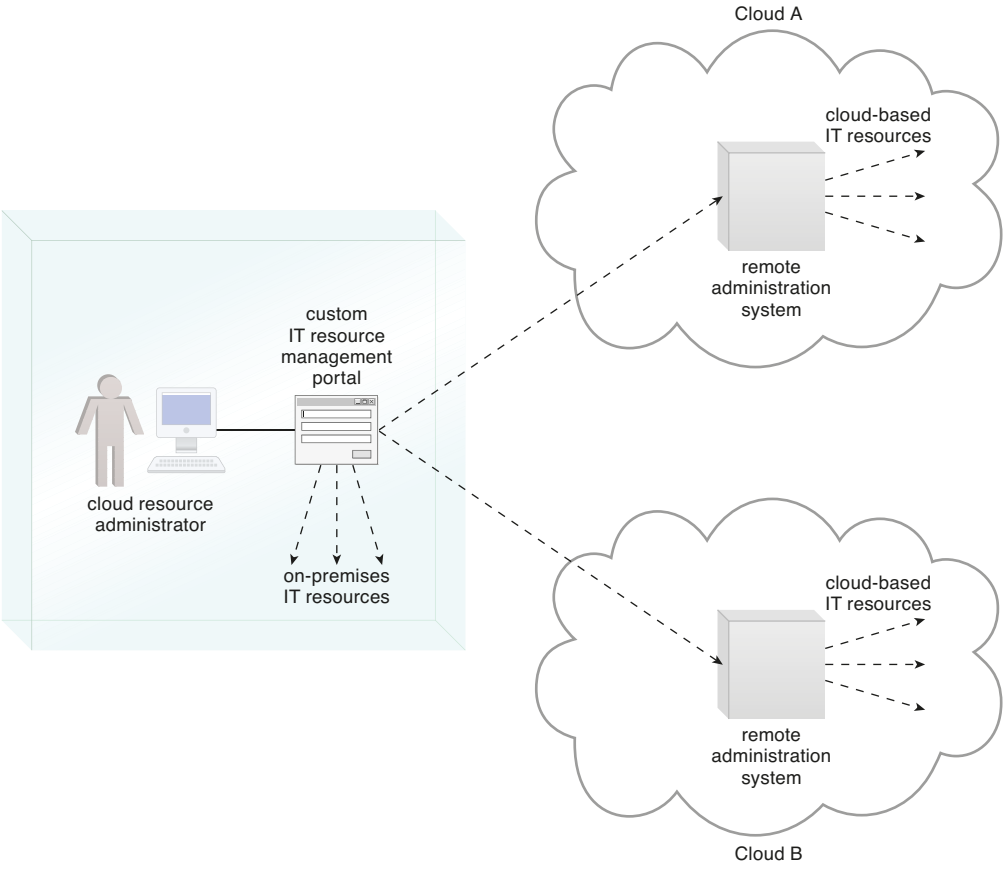


Figure 12.4

Standardized APIs published by remote administration systems from different clouds enable a cloud consumer to develop a custom portal that centralizes a single IT resource management portal for both cloud-based and on-premises IT resources.

CASE STUDY EXAMPLE

DTGOV has been offering its cloud consumers a user-friendly remote administration system for some time and recently determined that upgrades are required to accommodate the growing number of cloud consumers and the increasing diversity of requests. DTGOV is planning a development project to extend the remote administration system to fulfill the following requirements:

- Cloud consumers need to be able to self-provision virtual servers and virtual storage devices. The system specifically needs to interoperate with the cloud-enabled VIM platform's proprietary API to enable self-provisioning capabilities.
- A single sign-on mechanism (described in Chapter 10) needs to be incorporated to centrally authorize and control cloud consumer access.
- An API that supports the provisioning, starting, stopping, releasing, up–down scaling, and replicating of commands for virtual servers and cloud storage devices needs to be exposed.

In support of these features, a self-service portal is developed and the feature set of DTGOV's existing usage and administration portal is extended.

12.2 Resource Management System

The *resource management system* mechanism helps coordinate IT resources in response to management actions performed by both cloud consumers and cloud providers (Figure 12.5). Core to this system is the virtual infrastructure manager (VIM) that coordinates the server hardware so that virtual server instances can be created from the most expedient underlying physical server. A VIM is a commercial product that can be used to manage a range of virtual IT resources across multiple physical servers. For example, a VIM can create and manage multiple instances of a hypervisor across different physical servers or allocate a virtual server on one physical server to another (or to a resource pool).

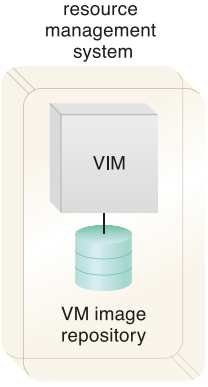


Figure 12.5
A resource management system encompassing a VIM platform and a virtual machine image repository. The VIM may have additional repositories, including one dedicated to storing operational data.

Tasks that are typically automated and implemented through the resource management system include:

- managing virtual IT resource templates that are used to create prebuilt instances, such as virtual server images
- allocating and releasing virtual IT resources into the available physical infrastructure in response to the starting, pausing, resuming, and termination of virtual IT resource instances
- coordinating IT resources in relation to the involvement of other mechanisms, such as resource replication, load balancer, and failover system
- enforcing usage and security policies throughout the lifecycle of cloud service instances
- monitoring operational conditions of IT resources

Resource management system functions can be accessed by cloud resource administrators employed by the cloud provider or cloud consumer. Those working on behalf of a cloud provider will often be able to directly access the resource management system's native console.

Resource management systems typically expose APIs that allow cloud providers to build remote administration system portals that can be customized to selectively offer resource management controls to external cloud resource administrators acting on behalf of cloud consumer organizations via usage and administration portals.

Both forms of access are depicted in Figure 12.6.

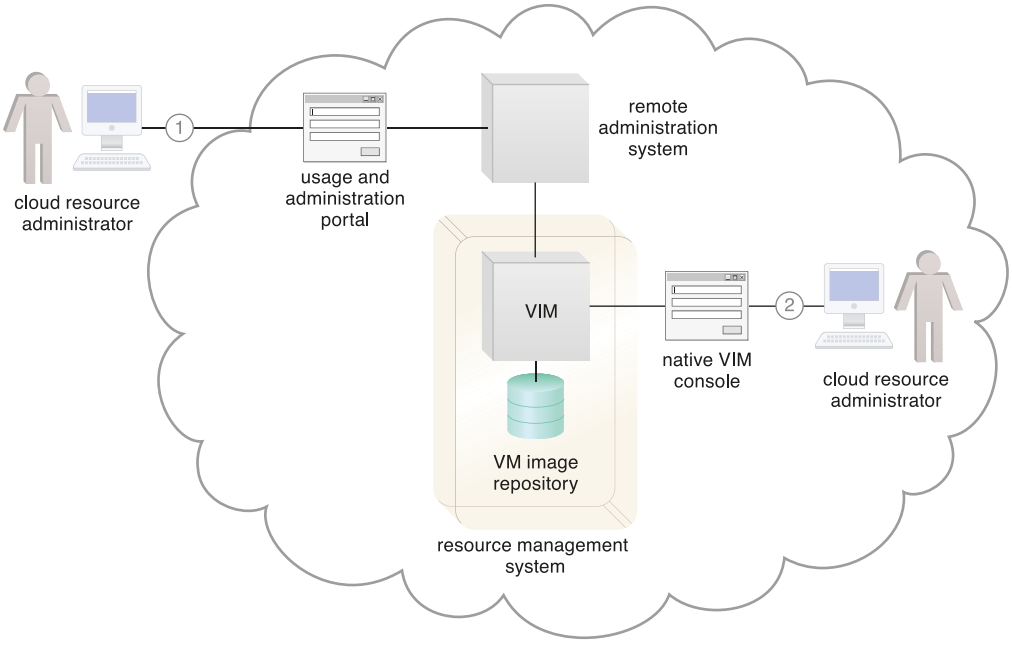


Figure 12.6

The cloud consumer's cloud resource administrator accesses a usage and administration portal externally to administer a leased IT resource (1). The cloud provider's cloud resource administrator uses the native user interface provided by the VIM to perform internal resource management tasks (2).

CASE STUDY EXAMPLE

The DTGOV resource management system is an extension of a new VIM product it purchased and provides the following primary features:

- management of virtual IT resources with a flexible allocation of pooled IT resources across different data centers
- management of cloud consumer databases
- isolation of virtual IT resources at logical perimeter networks
- management of a template virtual server image inventory available for immediate instantiation

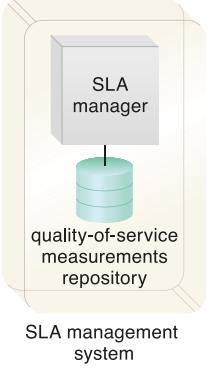
- automated replication ("snapshotting") of virtual server images for virtual server creation
- automated up–down scaling of virtual servers according to usage thresholds to enable live VM migration among physical servers
- an API for the creation and management of virtual servers and virtual storage devices
- an API for the creation of network access control rules
- an API for the up–down scaling of virtual IT resources
- an API for the migration and replication of virtual IT resources across multiple data centers
- interoperation with a single sign-on mechanism through an LDAP interface

Custom-designed SNMP command scripts are further implemented to interoperate with the network management tools to establish isolated virtual networks across multiple data centers.

12.3 SLA Management System

The *SLA management system* mechanism represents a range of commercially available cloud management products that provide features pertaining to the administration, collection, storage, reporting, and runtime notification of SLA data (Figure 12.7).

Figure 12.7
An SLA management system encompassing an SLA manager and QoS measurements repository.



An SLA management system deployment will generally include a repository used to store and retrieve collected SLA data based on predefined metrics and reporting parameters. It will further rely on one or more SLA monitor mechanisms to collect the SLA data that can then be made available in near-realtime to usage and administration portals to provide ongoing feedback regarding active cloud services (Figure 12.8). The metrics monitored for individual cloud services are aligned with the SLA guarantees in the corresponding cloud provisioning contracts.

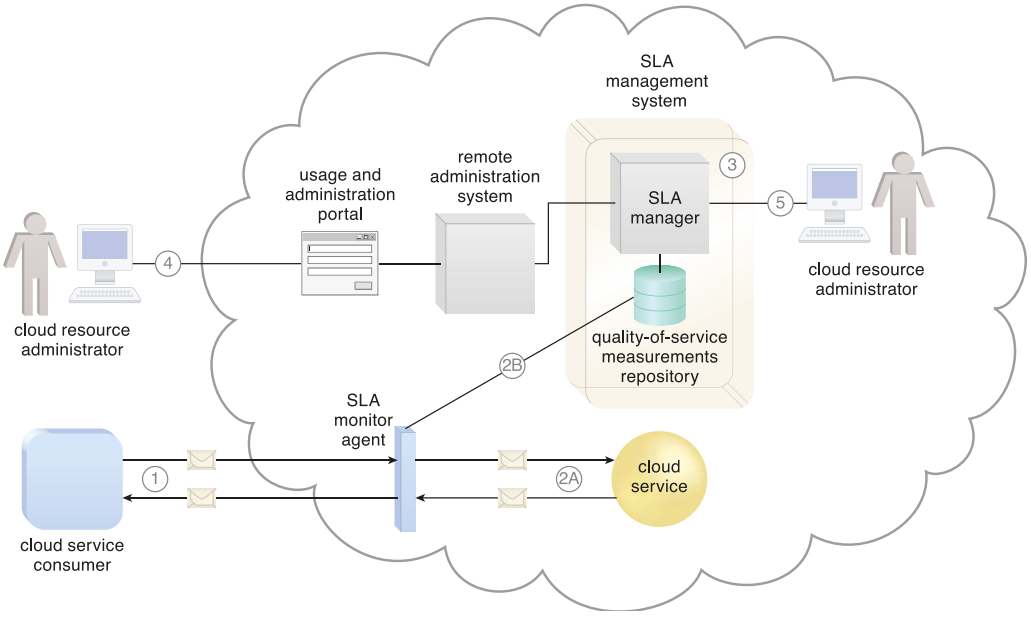


Figure 12.8

A cloud service consumer interacts with a cloud service (1). An SLA monitor intercepts the exchanged messages, evaluates the interaction, and collects relevant runtime data in relation to quality-of-service guarantees defined in the cloud service's SLA (2A). The data collected is stored in a repository (2B) that is part of the SLA management system (3). Queries can be issued and reports can be generated for an external cloud resource administrator via a usage and administration portal (4) or for an internal cloud resource administrator via the SLA management system's native user interface (5).

CASE STUDY EXAMPLE

DTGOV implements an SLA management system that interoperates with its existing VIM. This integration allows DTGOV cloud resource administrators to monitor the availability of a range of hosted IT resources via SLA monitors.

DTGOV works with the SLA management system's report design features to create the following predefined reports that are made available via custom dashboards:

- *Per-Data Center Availability Dashboard* – Publicly accessible through DTGOV's corporate cloud portal, this dashboard shows the overall operational conditions of each group of IT resources at each data center, in realtime.
- *Per-Cloud Consumer Availability Dashboard* – This dashboard displays realtime operational conditions of individual IT resources. Information about each IT resource can only be accessed by the cloud provider and the cloud consumer leasing or owning the IT resource.
- *Per-Cloud Consumer SLA Report* – This report consolidates and summarizes SLA statistics for cloud consumer IT resources, including downtimes and other time-stamped SLA events.

The SLA events generated by the SLA monitors represent the status and performance of physical and virtual IT resources that are controlled by the virtualization platform. The SLA management system interoperates with the network management tools through a custom-designed SNMP software agent that receives the SLA event notifications.

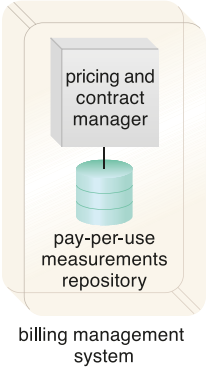
The SLA management system also interacts with the VIM through its proprietary API to associate each network SLA event with the affected virtual IT resource. The system includes a proprietary database used to store SLA events (such as virtual server and network downtimes).

The SLA management system exposes a REST API that DTGOV uses to interface with its central remote administration system. The proprietary API has a component service implementation that can be used for batch processing with the billing management system. DTGOV utilizes this to periodically provide downtime data that translates into credit applied to cloud consumer usage fees.

12.4 Billing Management System

The *billing management system* mechanism is dedicated to the collection and processing of usage data as it pertains to cloud provider accounting and cloud consumer billing. Specifically, the billing management system relies on pay-per-use monitors to gather runtime usage data that is stored in a repository that the system components then draw from for billing, reporting, and invoicing purposes (Figures 12.9 and 12.10).

Figure 12.9
A billing management system comprised of a pricing and contract manager and a pay-per-use measurements repository.



The billing management system allows for the definition of different pricing policies, as well as custom pricing models on a per-cloud-consumer and/or per-IT-resource basis. Pricing models can vary from the traditional pay-per-use models, to flat-rate or pay-per-allocation models, or combinations thereof.

Billing arrangements can be based on pre-usage and post-usage payments. The latter type can include predefined limits or it can be set up (with the mutual agreement of the cloud consumer) to allow for unlimited usage (and, consequently, no limit on subsequent billing). When limits are established, they are usually in the form of usage quotas. When quotas are exceeded, the billing management system can block further usage requests by cloud consumers.

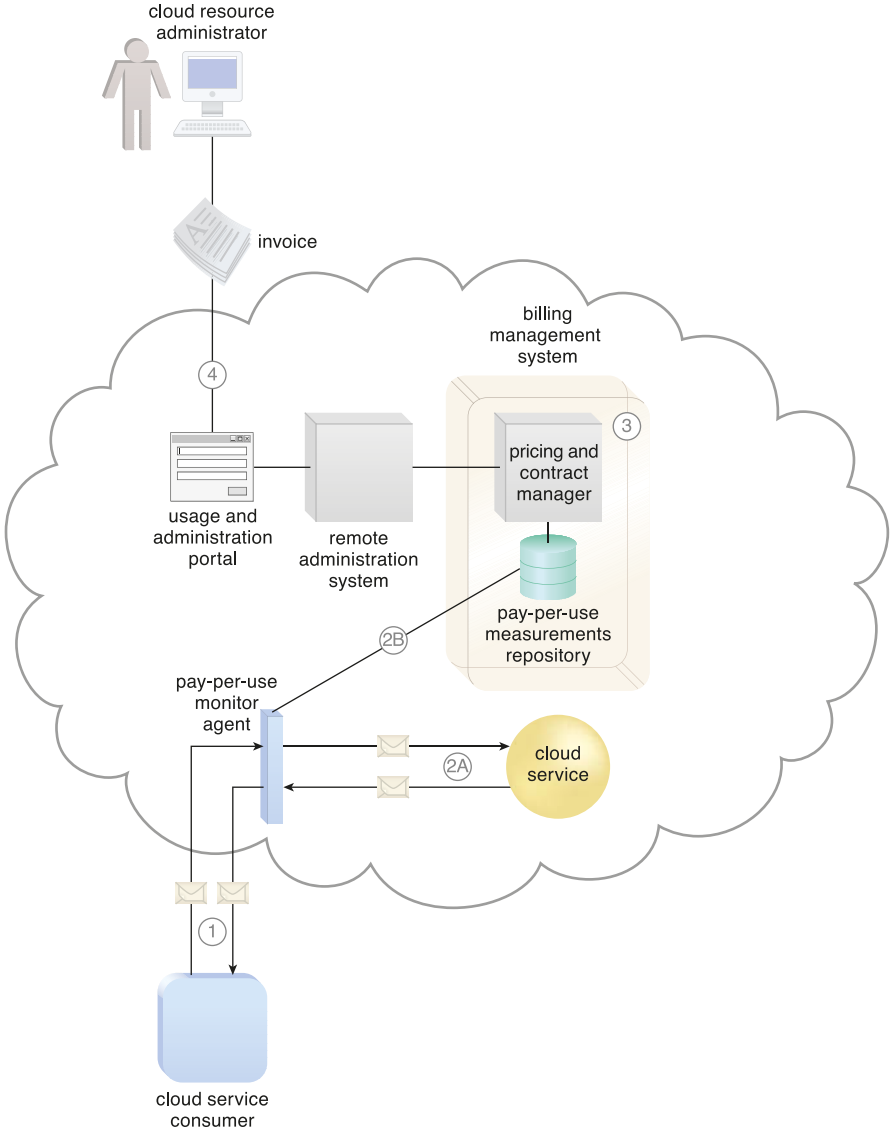


Figure 12.10

A cloud service consumer exchanges messages with a cloud service (1). A pay-per-use monitor keeps track of the usage and collects data relevant to billing (2A), which is forwarded to a repository that is part of the billing management system (2B). The system periodically calculates the consolidated cloud service usage fees and generates an invoice for the cloud consumer (3). The invoice may be provided to the cloud consumer through the usage and administration portal (4).

CASE STUDY EXAMPLE

DTGOV decides to establish a billing management system that enables them to create invoices for custom-defined billable events, such as subscriptions and IT resource volume usage. The billing management system is customized with the necessary events and pricing scheme metadata.

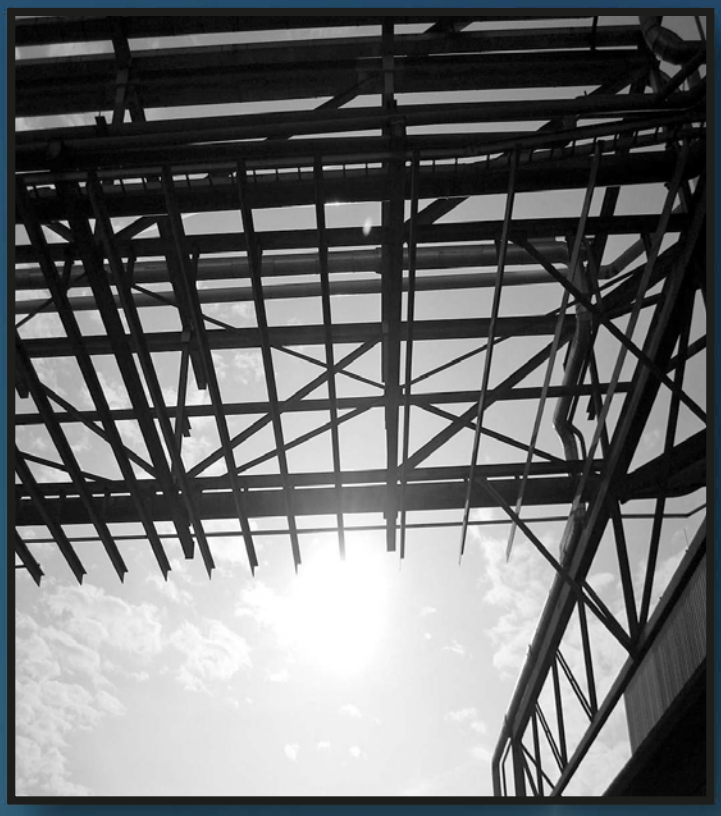
It includes the following two corresponding proprietary databases:

- billable event repository
- pricing scheme repository

Usage events are collected from pay-per-use monitors that are implemented as extensions to the VIM platform. Thin-granularity usage events, such as virtual server starting, stopping, up–down scaling, and decommissioning, are stored in a repository managed by the VIM platform.

The pay-per-use monitors further regularly supply the billing management system with the appropriate billable events. A standard pricing model is applied to most cloud consumer contracts, although it can be customized when special terms are negotiated.

This page intentionally left blank



Part III

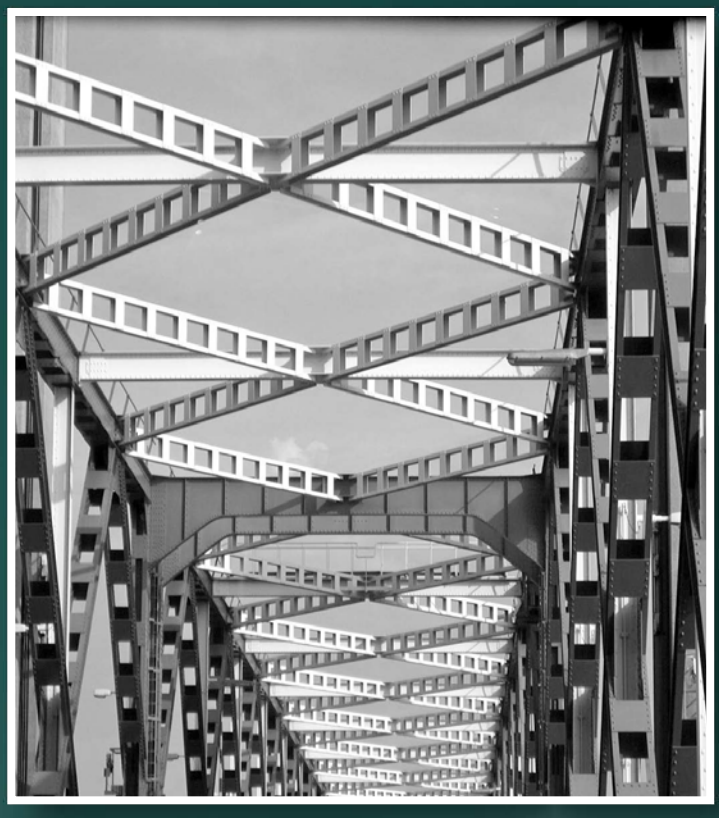
Cloud Computing Architecture

Cloud technology architectures formalize functional domains within cloud environments by establishing well-defined solutions comprised of interactions, behaviors, and distinct combinations of cloud computing mechanisms and other specialized cloud technology components.

The fundamental cloud architectural models covered in Chapter 13 establish foundational layers of technology architecture common to most clouds. Many of the advanced and specialized models described in Chapters 14 and 15 build upon these foundations to add complex and more narrowly focused solution architectures.

NOTE

Most of the cloud architectures described over the next three chapters are documented in greater detail in the book *Cloud Computing Design Patterns* (by Thomas Erl and Amin Naserpour), also part of the Pearson Digital Enterprise Series from Thomas Erl. Visit www.thomaserl.com/books for more information.



Chapter 13

Fundamental Cloud Architectures

This chapter describes the following foundational cloud architectural models:

- Workload Distribution
- Resource Pooling
- Dynamic Scalability
- Elastic Resource Capacity
- Service Load Balancing
- Cloud Bursting
- Elastic Disk Provisioning
- Redundant Storage
- Multicloud

For each architecture, the typical involvement of cloud computing mechanisms (previously covered in Part II) is documented.

13.1 Workload Distribution Architecture

IT resources can be horizontally scaled via the addition of one or more identical IT resources and a load balancer that provides runtime logic capable of evenly distributing the workload among the available IT resources (Figure 13.1). The resulting *workload distribution architecture* reduces both IT resource overutilization and underutilization to an extent dependent upon the sophistication of the load balancing algorithms and runtime logic.

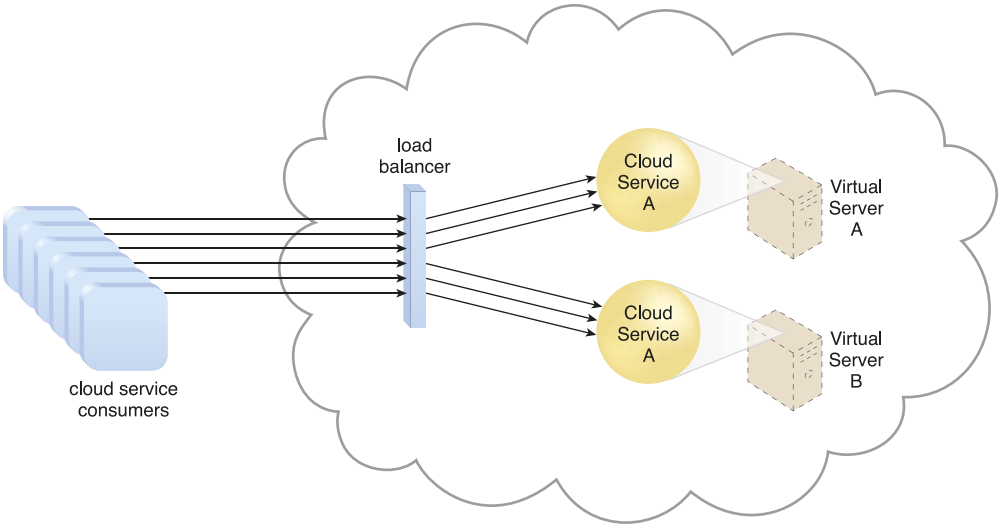


Figure 13.1

A redundant copy of Cloud Service A is implemented on Virtual Server B. The load balancer intercepts cloud service consumer requests and directs them to both Virtual Servers A and B to ensure even workload distribution.

This fundamental architectural model can be applied to any IT resource, with workload distribution commonly carried out in support of distributed virtual servers, cloud storage devices, and cloud services. Load balancing systems applied to specific IT resources usually produce specialized variations of this architecture that incorporate aspects of load balancing, such as:

- the service load balancing architecture explained later in this chapter
- the load balanced virtual server architecture covered in Chapter 14
- the load balanced virtual switches architecture described in Chapter 15

In addition to the base load balancer mechanism, and the virtual server and cloud storage device mechanisms to which load balancing can be applied, the following mechanisms can also be part of this cloud architecture:

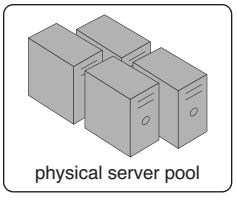
- *Audit Monitor* – When distributing runtime workloads, the type and geographical location of the IT resources that process the data can determine whether monitoring is necessary to fulfill legal and regulatory requirements.
- *Cloud Usage Monitor* – Various monitors can be involved to carry out runtime workload tracking and data processing.

- *Hypervisor* – Workloads between hypervisors and the virtual servers that they host may require distribution.
- *Logical Network Perimeter* – The logical network perimeter isolates cloud consumer network boundaries in relation to how and where workloads are distributed.
- *Resource Cluster* – Clustered IT resources in active–active mode are commonly used to support workload balancing between different cluster nodes.
- *Resource Replication* – This mechanism can generate new instances of virtualized IT resources in response to runtime workload distribution demands.

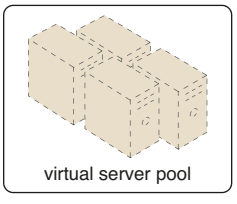
13.2 Resource Pooling Architecture

A *resource pooling architecture* is based on the use of one or more resource pools, in which identical IT resources are grouped and maintained by a system that automatically ensures that they remain synchronized.

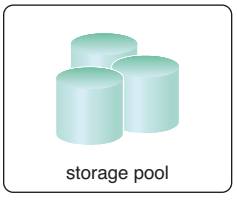
Provided here are common examples of resource pools:



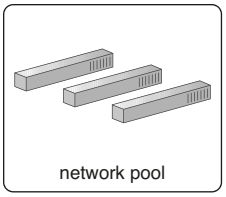
Physical server pools are composed of networked servers that have been installed with operating systems and other necessary programs and/or applications and are ready for immediate use.



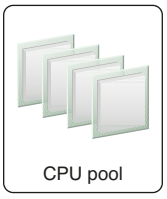
Virtual server pools are usually configured using one of several available templates chosen by the cloud consumer during provisioning. For example, a cloud consumer can set up a pool of mid-tier Windows servers with 4 GB of RAM or a pool of low-tier Ubuntu servers with 2 GB of RAM.



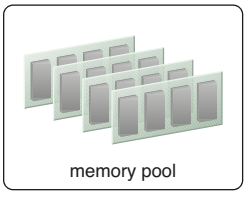
Storage pools, or cloud storage device pools, consist of file-based or block-based storage structures that contain empty and/or filled cloud storage devices.



Network pools (or interconnect pools) are composed of different preconfigured network connectivity devices. For example, a pool of virtual firewall devices or physical network switches can be created for redundant connectivity, load balancing, or link aggregation.



CPU pools are ready to be allocated to virtual servers and are typically broken down into individual processing cores.



Pools of physical RAM can be used in newly provisioned physical servers or to vertically scale physical servers.

Dedicated pools can be created for each type of IT resource, and individual pools can be grouped into a larger pool, in which case each individual pool becomes a sub-pool (Figure 13.2).

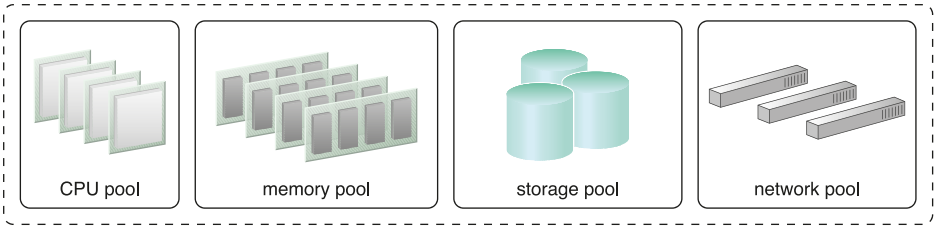


Figure 13.2

A sample resource pool that is comprised of four sub-pools of CPUs, memory, cloud storage devices, and virtual network devices.

Resource pools can become highly complex, with multiple pools created for specific cloud consumers or applications. A hierarchical structure can be established to form parent, sibling, and nested pools to facilitate the organization of diverse resource pooling requirements (Figure 13.3).

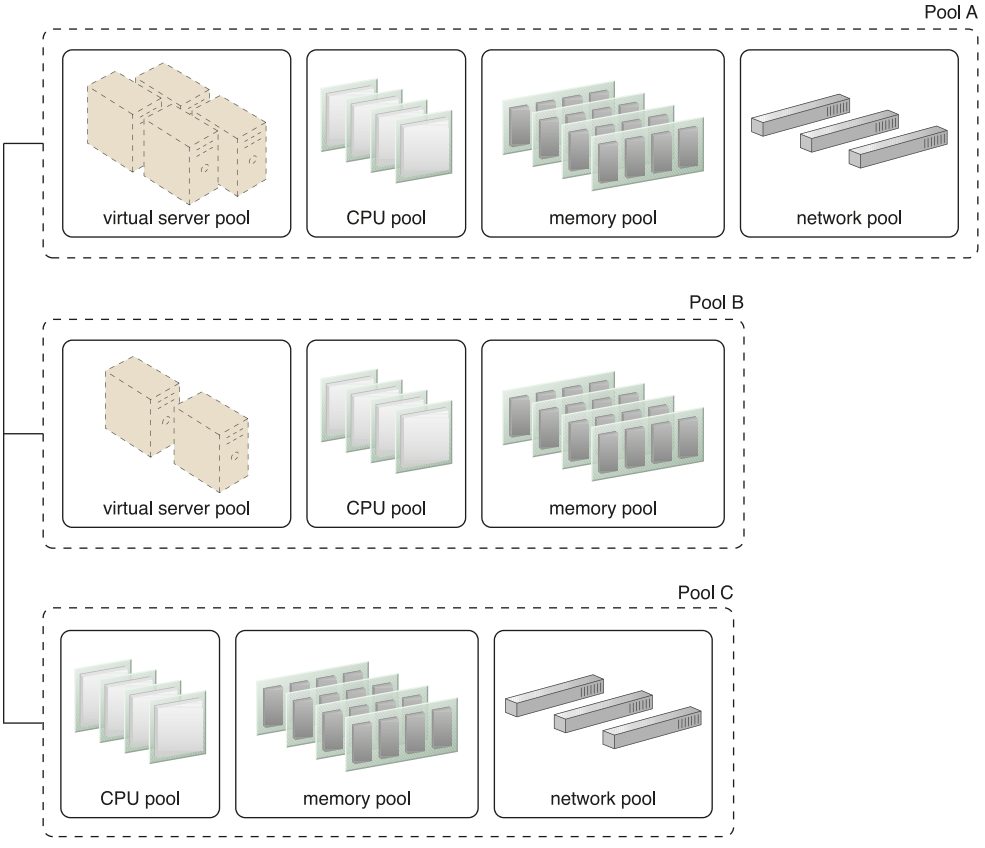


Figure 13.3

Pools B and C are sibling pools that are taken from the larger Pool A, which has been allocated to a cloud consumer. This is an alternative to taking the IT resources for Pool B and Pool C from a general reserve of IT resources that is shared throughout the cloud.

Sibling resource pools are usually drawn from physically grouped IT resources, as opposed to IT resources that are spread out over different data centers. Sibling pools are isolated from one another so that each cloud consumer is only provided access to its respective pool.

In the nested pool model, larger pools are divided into smaller pools that individually group the same type of IT resources together (Figure 13.4). Nested pools can be used to assign resource pools to different departments or groups in the same cloud consumer organization.

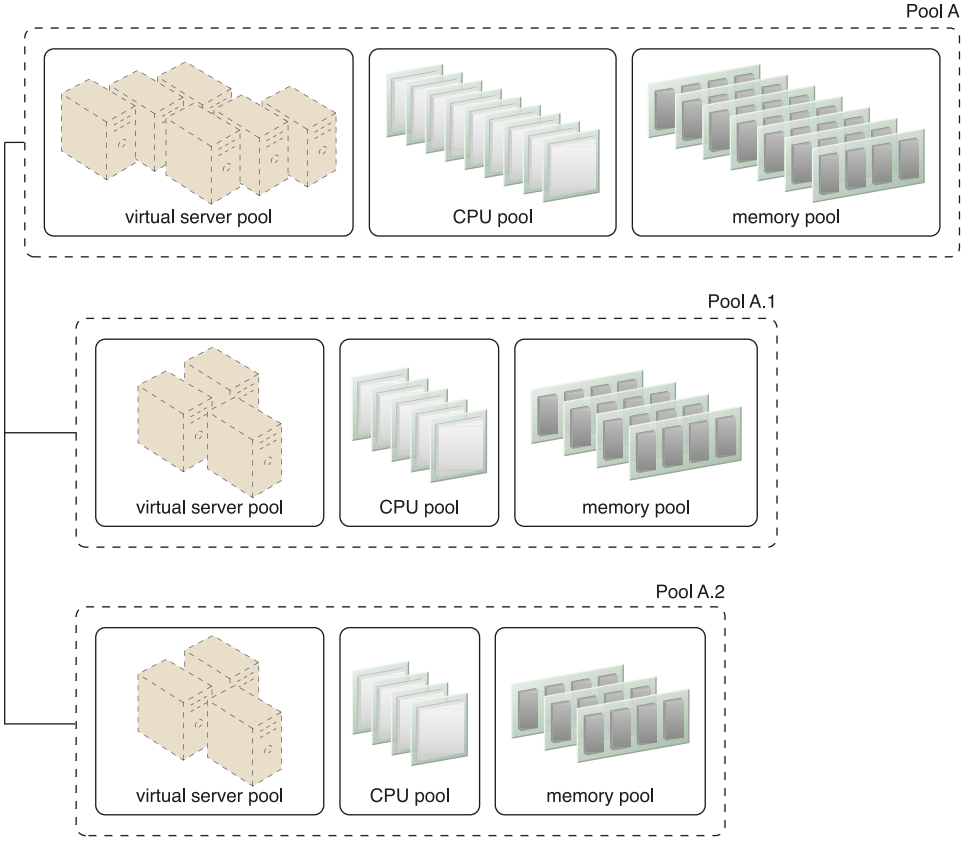


Figure 13.4

Nested Pools A.1 and Pool A.2 are comprised of the same IT resources as Pool A, but in different quantities. Nested pools are typically used to provision cloud services that need to be rapidly instantiated using the same type of IT resources with the same configuration settings.

After resource pools have been defined, multiple instances of IT resources from each pool can be created to provide an in-memory pool of "live" IT resources.

In addition to cloud storage devices and virtual servers, which are commonly pooled mechanisms, the following mechanisms can also be part of this cloud architecture:

- *Audit Monitor* – This mechanism monitors resource pool usage to ensure compliance with privacy and regulation requirements, especially when pools contain cloud storage devices or data loaded into memory.

- *Cloud Usage Monitor* – Various cloud usage monitors are involved in the runtime tracking and synchronization that are required by the pooled IT resources and any underlying management systems.
- *Hypervisor* – The hypervisor mechanism is responsible for providing virtual servers with access to resource pools, in addition to hosting the virtual servers and sometimes the resource pools themselves.
- *Logical Network Perimeter* – The logical network perimeter is used to logically organize and isolate resource pools.
- *Pay-Per-Use Monitor* – The pay-per-use monitor collects usage and billing information on how individual cloud consumers are allocated and use IT resources from various pools.
- *Remote Administration System* – This mechanism is commonly used to interface with back-end systems and programs to provide resource pool administration features via a front-end portal.
- *Resource Management System* – The resource management system mechanism supplies cloud consumers with the tools and permission management options for administering resource pools.
- *Resource Replication* – This mechanism is used to generate new instances of IT resources for resource pools.

13.3 Dynamic Scalability Architecture

The *dynamic scalability architecture* is an architectural model based on a system of predefined scaling conditions that trigger the dynamic allocation of IT resources from resource pools. Dynamic allocation enables variable utilization as dictated by usage demand fluctuations, since unnecessary IT resources are efficiently reclaimed without requiring manual interaction.

The automated scaling listener is configured with workload thresholds that dictate when new IT resources need to be added to the workload processing. This mechanism can be provided with logic that determines how many additional IT resources can be dynamically provided, based on the terms of a given cloud consumer's provisioning contract.

The following types of dynamic scaling are commonly used:

- *Dynamic Horizontal Scaling* – IT resource instances are scaled out and in to handle fluctuating workloads. The automated scaling listener monitors requests and signals the resource replication mechanism to initiate IT resource duplication, as per requirements and permissions.
- *Dynamic Vertical Scaling* – IT resource instances are scaled up and down when there is a need to adjust the processing capacity of a single IT resource. For example, a virtual server that is being overloaded can have its memory dynamically increased or it may have a processing core added.
- *Dynamic Relocation* – The IT resource is relocated to a host with more capacity. For example, a database may need to be moved from a tape-based SAN storage device with 4 GB per second I/O capacity to another disk-based SAN storage device with 8 GB per second I/O capacity.

Figures 13.5 to 13.7 illustrate the process of dynamic horizontal scaling.

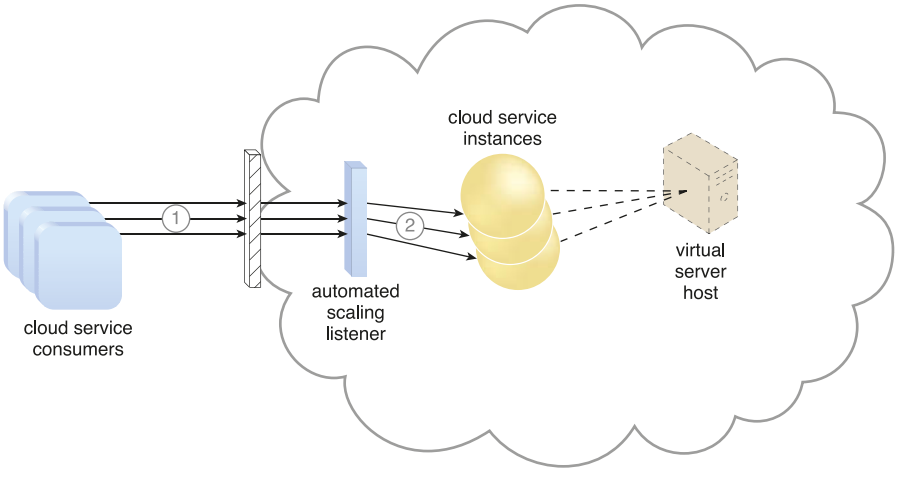


Figure 13.5

Cloud service consumers are sending requests to a cloud service (1). The automated scaling listener monitors the cloud service to determine if predefined capacity thresholds are being exceeded (2).

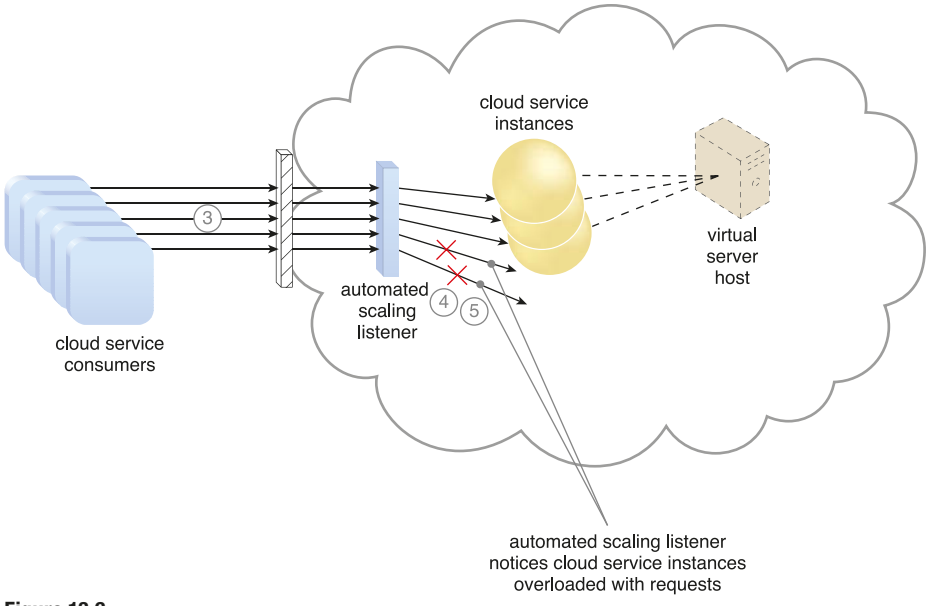


Figure 13.6

The number of requests coming from cloud service consumers increases (3). The workload exceeds the performance thresholds. The automated scaling listener determines the next course of action based on a predefined scaling policy (4). If the cloud service implementation is deemed eligible for additional scaling, the automated scaling listener initiates the scaling process (5).

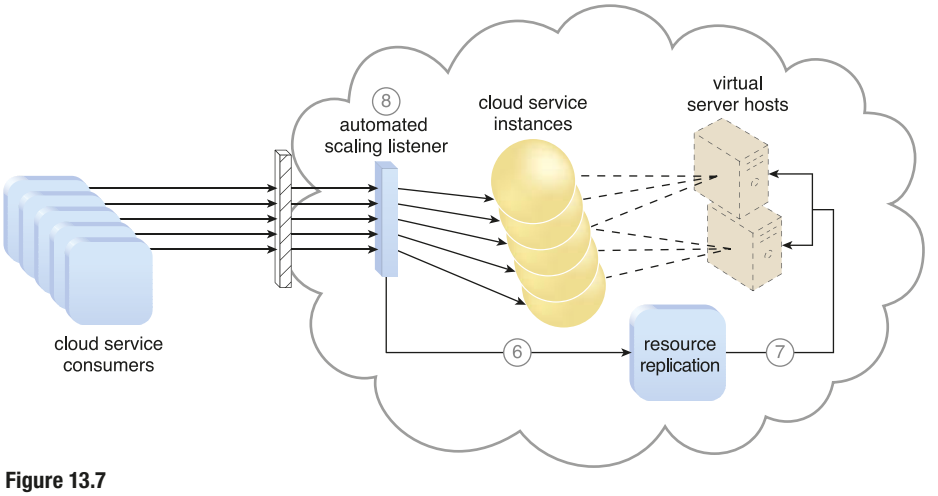


Figure 13.7

The automated scaling listener sends a signal to the resource replication mechanism (6), which creates more instances of the cloud service (7). Now that the increased workload has been accommodated, the automated scaling listener resumes monitoring and scaling IT resources, as required (8).

The dynamic scalability architecture can be applied to a range of IT resources, including virtual servers and cloud storage devices. In addition to the core automated scaling listener and resource replication mechanisms, the following mechanisms can also be used in this form of cloud architecture:

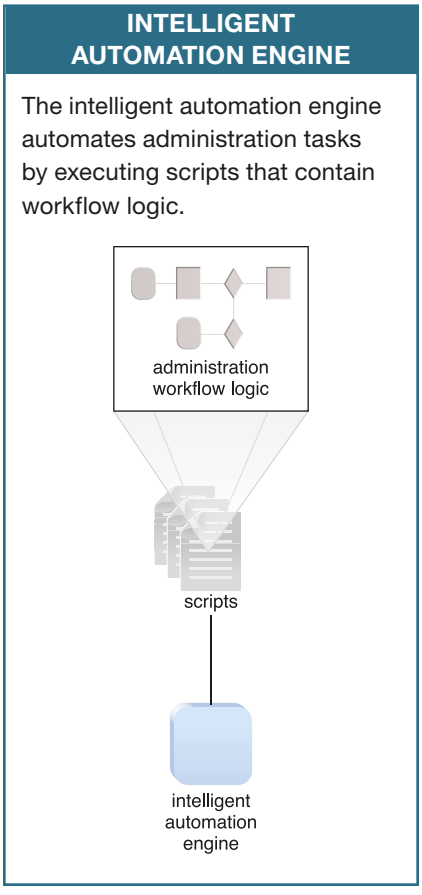
- *Cloud Usage Monitor* – Specialized cloud usage monitors can track runtime usage in response to dynamic fluctuations caused by this architecture.
- *Hypervisor* – The hypervisor is invoked by a dynamic scalability system to create or remove virtual server instances, or to be scaled itself.
- *Pay-Per-Use Monitor* – The pay-per-use monitor is engaged to collect usage cost information in response to the scaling of IT resources.

13.4 Elastic Resource Capacity Architecture

The *elastic resource capacity architecture* is primarily related to the dynamic provisioning of virtual servers, using a system that allocates and reclaims CPUs and RAM in immediate response to the fluctuating processing requirements of hosted IT resources (Figures 13.8 and 13.9).

Resource pools are used by scaling technology that interacts with the hypervisor and/or VIM to retrieve and return CPU and RAM resources at runtime. The runtime processing of the virtual server is monitored so that additional processing power can be leveraged from the resource pool via dynamic allocation, before capacity thresholds are met. The virtual server and its hosted applications and IT resources are vertically scaled in response.

This type of cloud architecture can be designed so that the intelligent automation engine script sends its scaling request via the VIM instead of to the hypervisor directly. Virtual servers that participate in elastic resource allocation systems may require rebooting for the dynamic resource allocation to take effect.



Some additional mechanisms that can be included in this cloud architecture are the following:

- *Cloud Usage Monitor* – Specialized cloud usage monitors collect resource usage information on IT resources before, during, and after scaling to help define the future processing capacity thresholds of the virtual servers.
- *Pay-Per-Use Monitor* – The pay-per-use monitor is responsible for collecting resource usage cost information as it fluctuates with the elastic provisioning.
- *Resource Replication* – Resource replication is used by this architectural model to generate new instances of the scaled IT resources.

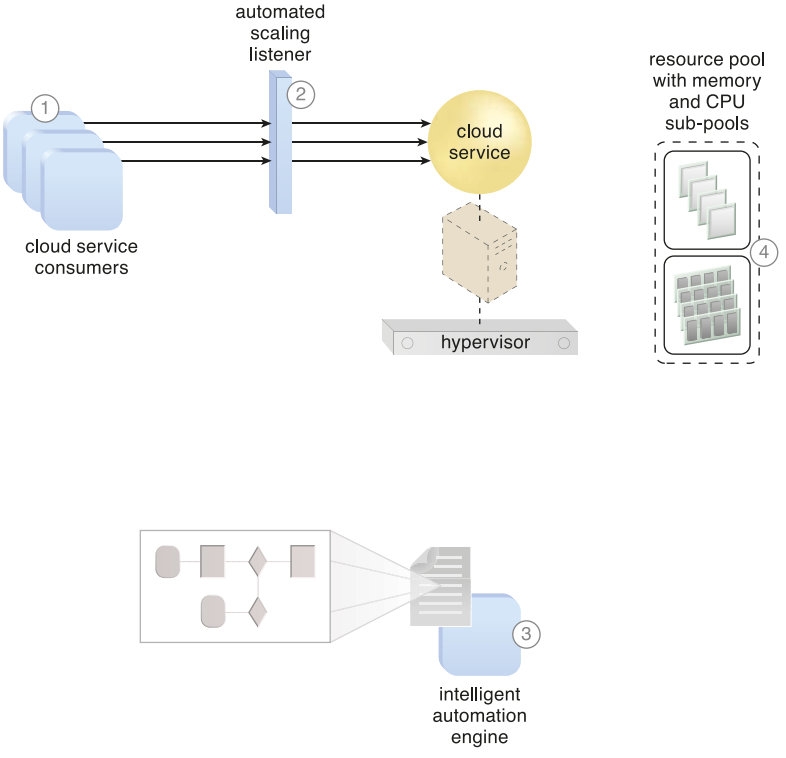


Figure 13.8

Cloud service consumers are actively sending requests to a cloud service (1), which are monitored by an automated scaling listener (2). An intelligent automation engine script is deployed with workflow logic (3) that is capable of notifying the resource pool using allocation requests (4).

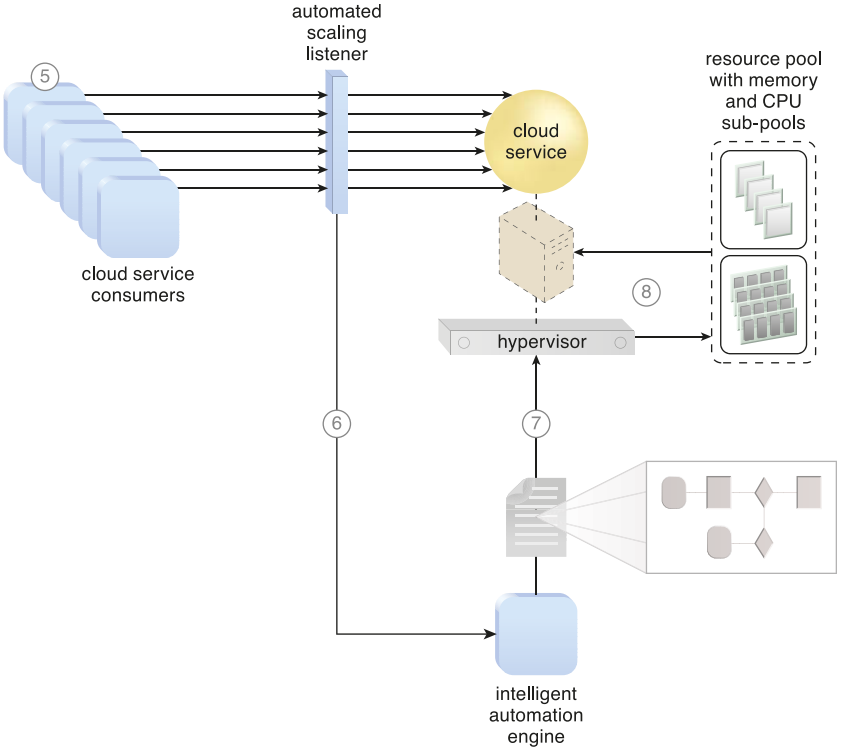


Figure 13.9

Cloud service consumer requests increase (5), causing the automated scaling listener to signal the intelligent automation engine to execute the script (6). The script runs the workflow logic that signals the hypervisor to allocate more IT resources from the resource pools (7). The hypervisor allocates additional CPU and RAM to the virtual server, enabling the increased workload to be handled (8).

13.5 Service Load Balancing Architecture

The *service load balancing architecture* can be considered a specialized variation of the workload distribution architecture that is geared specifically for scaling cloud service implementations. Redundant deployments of cloud services are created, with a load balancing system added to dynamically distribute workloads.

The duplicate cloud service implementations are organized into a resource pool, while the load balancer is positioned as either an external or built-in component to allow the host servers to balance the workloads themselves.

Depending on the anticipated workload and processing capacity of host server environments, multiple instances of each cloud service implementation can be generated as part of a resource pool that responds to fluctuating request volumes more efficiently.

The load balancer can be positioned either independent of the cloud services and their host servers (Figure 13.10), or built-in as part of the application or server's environment. In the latter case, a primary server with the load balancing logic can communicate with neighboring servers to balance the workload (Figure 13.11).

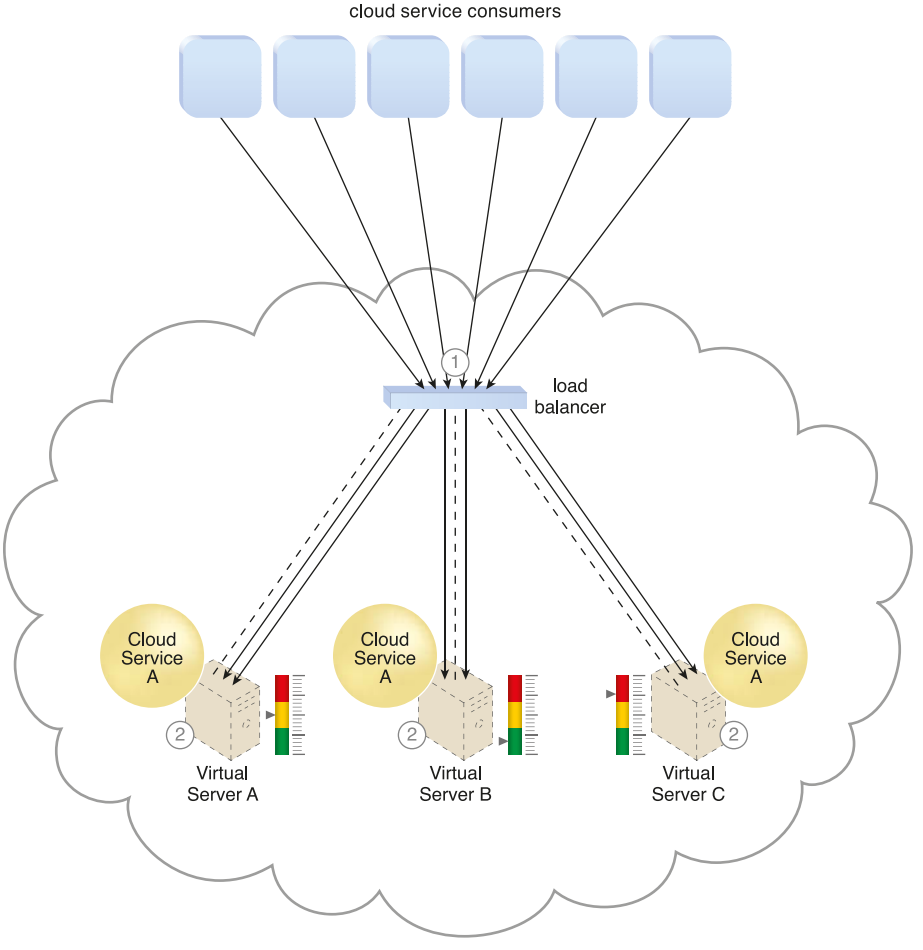


Figure 13.10

The load balancer intercepts messages sent by cloud service consumers (1) and forwards them to the virtual servers so that the workload processing is horizontally scaled (2).

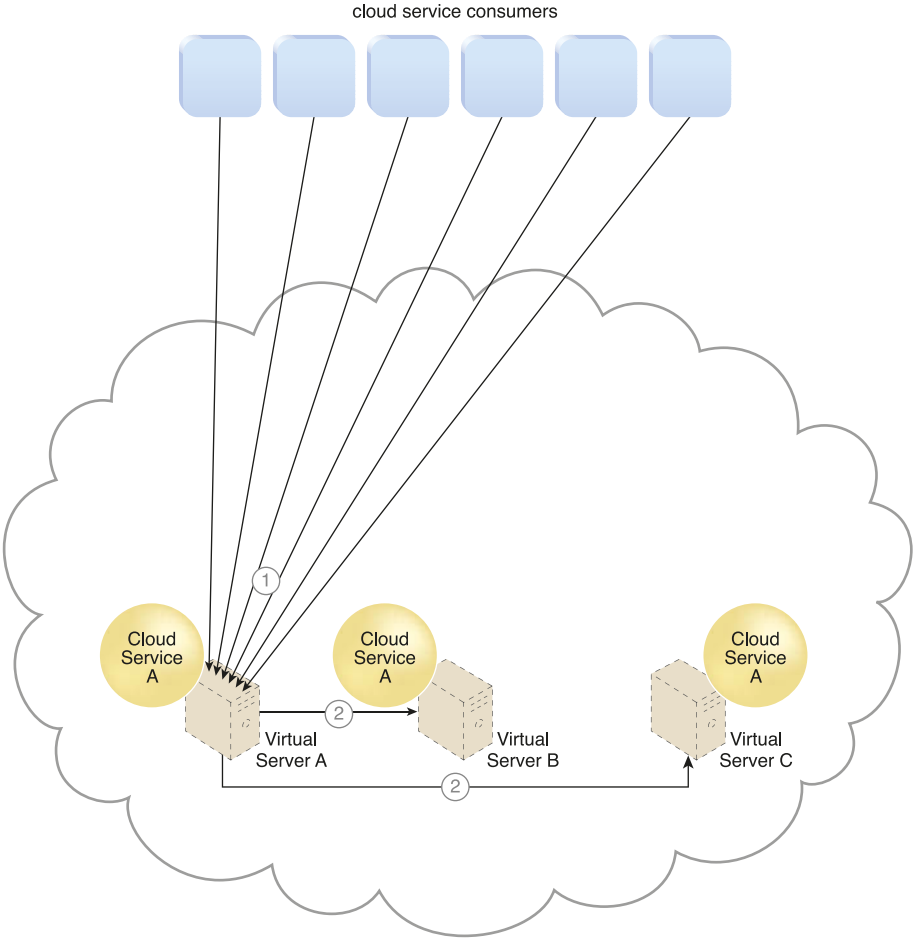


Figure 13.11

Cloud service consumer requests are sent to Cloud Service A on Virtual Server A (1). The cloud service implementation includes built-in load balancing logic that is capable of distributing requests to the neighboring Cloud Service A implementations on Virtual Servers B and C (2).

The service load balancing architecture can involve the following mechanisms in addition to the load balancer:

- *Cloud Usage Monitor* – Cloud usage monitors may be involved with monitoring cloud service instances and their respective IT resource consumption levels, as well as various runtime monitoring and usage data collection tasks.
- *Resource Cluster* – Active–active cluster groups are incorporated in this architecture to help balance workloads across different members of the cluster.
- *Resource Replication* – The resource replication mechanism is utilized to generate cloud service implementations in support of load balancing requirements.

13.6 Cloud Bursting Architecture

The *cloud bursting architecture* establishes a form of dynamic scaling that scales or "bursts out" on-premises IT resources into a cloud whenever predefined capacity thresholds have been reached. The corresponding cloud-based IT resources are redundantly pre-deployed but remain inactive until cloud bursting occurs. After they are no longer required, the cloud-based IT resources are released and the architecture "bursts in" back to the on-premises environment.

Cloud bursting is a flexible scaling architecture that provides cloud consumers with the option of using cloud-based IT resources only to meet higher usage demands. The foundation of this architectural model is based on the automated scaling listener and resource replication mechanisms.

The automated scaling listener determines when to redirect requests to cloud-based IT resources, and resource replication is used to maintain synchronicity between on-premises and cloud-based IT resources in relation to state information (Figure 13.12).

In addition to the automated scaling listener and resource replication, numerous other mechanisms can be used to automate the burst in and out dynamics for this architecture, depending primarily on the type of IT resource being scaled.

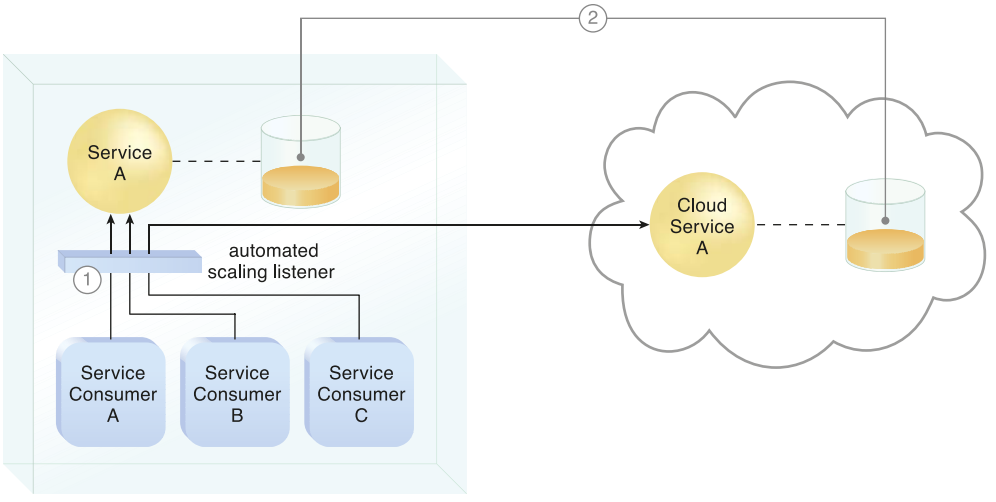


Figure 13.12

An automated scaling listener monitors the usage of on-premises Service A, and redirects Service Consumer C's request to Service A's redundant implementation in the cloud (Cloud Service A) once Service A's usage threshold has been exceeded (1). A resource replication system is used to keep state management databases synchronized (2).

13.7 Elastic Disk Provisioning Architecture

Cloud consumers are commonly charged for cloud-based storage space based on fixed-disk storage allocation, meaning the charges are predetermined by disk capacity and not aligned with actual data storage consumption. Figure 13.13 demonstrates this by illustrating a scenario in which a cloud consumer provisions a virtual server with the Windows Server operating system and three 150 GB hard drives. The cloud consumer is billed for using 450 GB of storage space after installing the operating system, even though it has not yet installed any software.

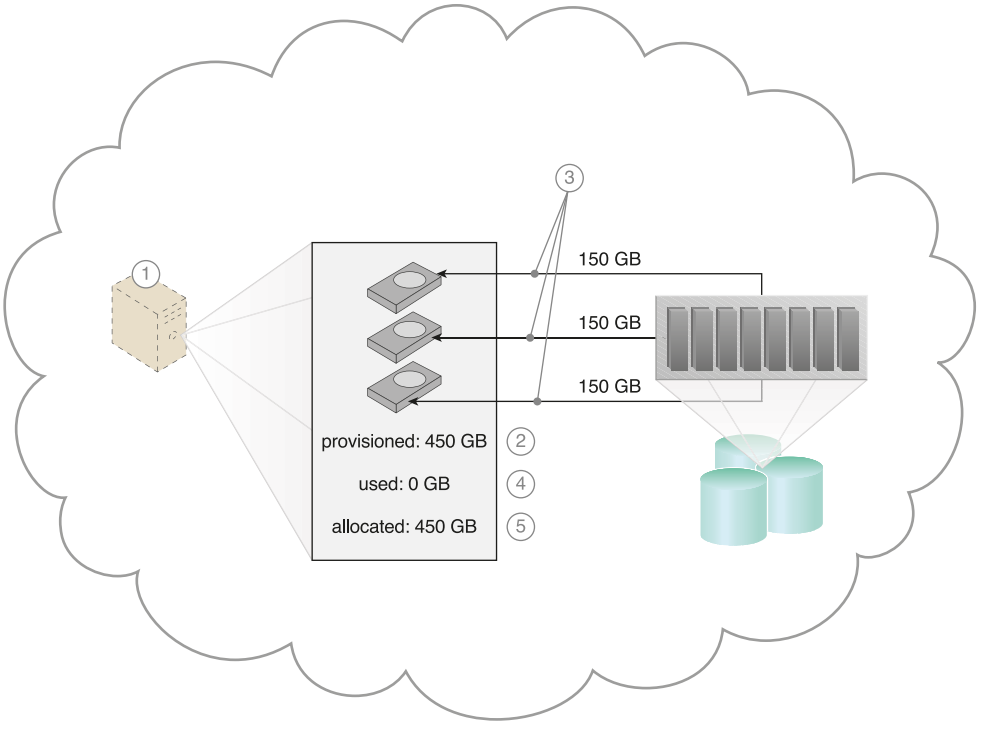


Figure 13.13

The cloud consumer requests a virtual server with three hard disks, each with a capacity of 150 GB (1). The virtual server is provisioned according to the elastic disk provisioning architecture, with a total of 450 GB of disk space (2). The 450 GB are allocated to the virtual server by the cloud provider (3). The cloud consumer has not installed any software yet, meaning the actual used space is currently 0 GB (4). Because the 450 GB are already allocated and reserved for the cloud consumer, it will be charged for 450 GB of disk usage as of the point of allocation (5).

The *elastic disk provisioning architecture* establishes a dynamic storage provisioning system that ensures that the cloud consumer is granularly billed for the exact amount of storage that it actually uses. This system uses thin-provisioning technology for the dynamic allocation of storage space and is further supported by runtime usage monitoring to collect accurate usage data for billing purposes (Figure 13.14).

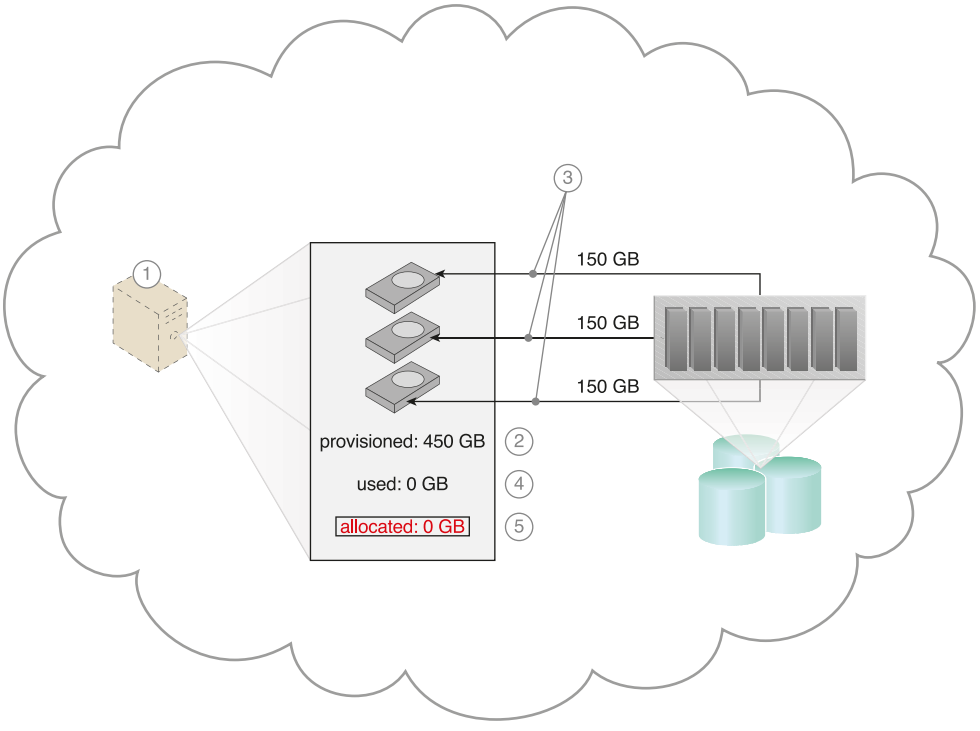


Figure 13.14

The cloud consumer requests a virtual server with three hard disks, each with a capacity of 150 GB (1). The virtual server is provisioned by this architecture with a total of 450 GB of disk space (2). The 450 GB are set as the maximum disk usage that is allowed for this virtual server, although no physical disk space has been reserved or allocated yet (3). The cloud consumer has not installed any software, meaning the actual used space is currently 0 GB (4). Because the allocated disk space is equal to the actual used space (which is currently zero), the cloud consumer is not charged for any disk space usage (5).

Thin-provisioning software is installed on virtual servers that process dynamic storage allocation via the hypervisor, while the pay-per-use monitor tracks and reports granular billing-related disk usage data (Figure 13.15).

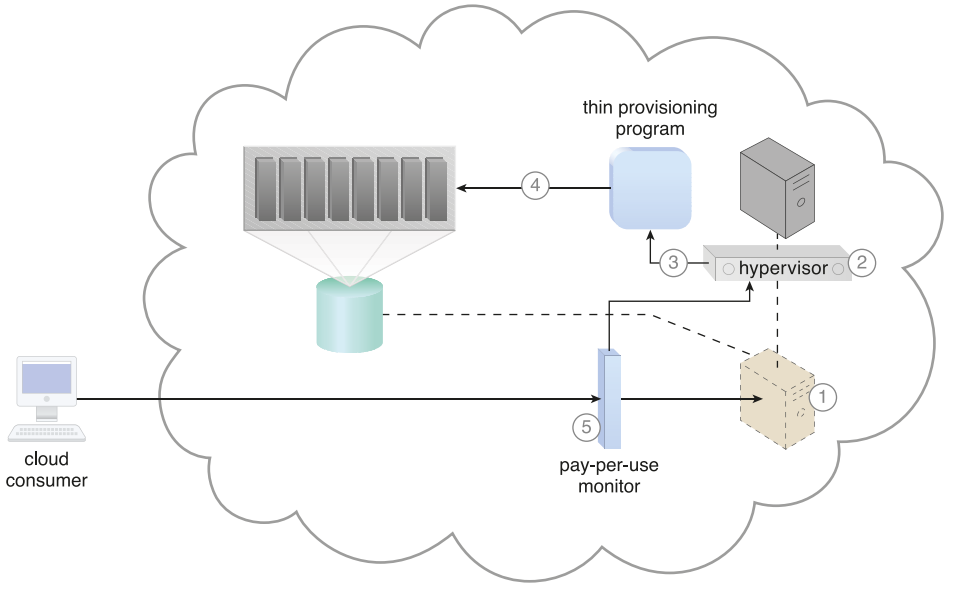


Figure 13.15

A request is received from a cloud consumer, and the provisioning of a new virtual server instance begins (1). As part of the provisioning process, the hard disks are chosen as dynamic or thin-provisioned disks (2). The hypervisor calls a dynamic disk allocation component to create thin disks for the virtual server (3). Virtual server disks are created via the thin-provisioning program and saved in a folder of near-zero size. The size of this folder and its files grow as operating applications are installed and additional files are copied onto the virtual server (4). The pay-per-use monitor tracks the actual dynamically allocated storage for billing purposes (5).

The following mechanisms can be included in this architecture in addition to the cloud storage device, virtual server, hypervisor, and pay-per-use monitor:

- *Cloud Usage Monitor* – Specialized cloud usage monitors can be used to track and log storage usage fluctuations.
- *Resource Replication* – Resource replication is part of an elastic disk provisioning system when conversion of dynamic thin-disk storage into static thick-disk storage is required.

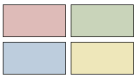
13.8 Redundant Storage Architecture

Cloud storage devices are occasionally subject to failures and disruptions that are caused by network connectivity issues, controller or general hardware failure, or security breaches. A compromised cloud storage device's reliability can have a ripple effect and cause impact failure across all the services, applications, and infrastructure components in the cloud that are reliant on its availability.

The *redundant storage architecture* introduces a secondary duplicate cloud storage device as part of a failover system that synchronizes its data with the data in the primary cloud storage device. A storage service gateway diverts cloud consumer requests to the secondary device whenever the primary device fails (Figures 13.16 and 13.17).

LUN

A logical unit number (LUN) is a logical drive that represents a partition of a physical drive.



LUNs

STORAGE SERVICE GATEWAY

The storage service gateway is a component that acts as the external interface to cloud storage services and is capable of automatically redirecting cloud consumer requests whenever the location of the requested data has changed.

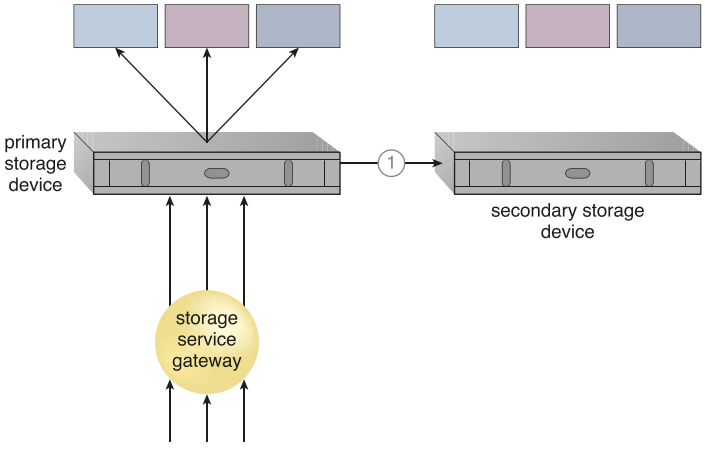


Figure 13.16

The primary cloud storage device is routinely replicated to the secondary cloud storage device (1).

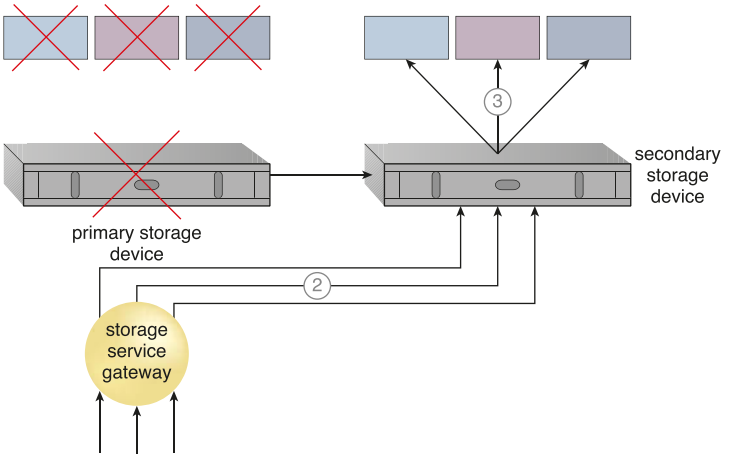


Figure 13.17

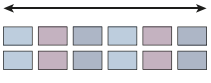
The primary storage becomes unavailable and the storage service gateway forwards the cloud consumer requests to the secondary storage device (2). The secondary storage device forwards the requests to the LUNs, allowing cloud consumers to continue to access their data (3).

This cloud architecture primarily relies on a storage replication system that keeps the primary cloud storage device synchronized with its duplicate secondary cloud storage devices (Figure 13.18).

Cloud providers may locate secondary cloud storage devices in a different geographical region than the primary cloud storage device, usually for economic reasons. However, this can introduce legal concerns for some types of data. The location of the secondary cloud storage devices can dictate the protocol and method used for synchronization, as some replication transport protocols have distance restrictions.

STORAGE REPLICATION

Storage replication is a variation of the resource replication mechanisms used to synchronously or asynchronously replicate data from a primary storage device to a secondary storage device. It can be used to replicate partial and entire LUNs.



storage replication

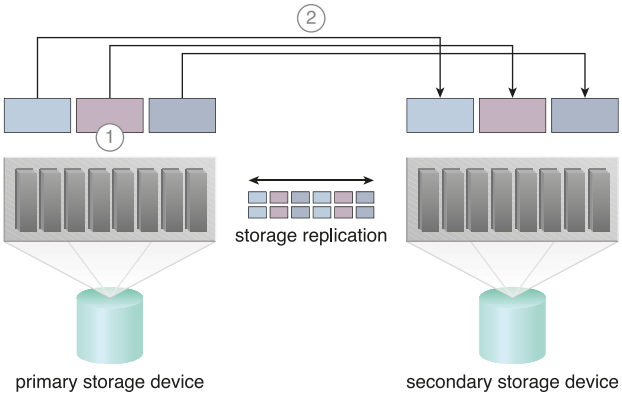


Figure 13.18

Storage replication is used to keep the redundant storage device synchronized with the primary storage device.

Some cloud providers use storage devices with dual array and storage controllers to improve device redundancy and place secondary storage devices in a different physical location for cloud balancing and disaster recovery purposes. In this case, cloud providers may need to lease a network connection via a third-party cloud provider to establish the replication between the two devices.

13.9 Multicloud Architecture

A cloud architecture that combines two or more public clouds is referred to as a *multicloud architecture* (Figure 13.19). The different clouds combined in this type of architecture may offer their resources through any of the cloud delivery models—namely, IaaS, PaaS, or SaaS. One of the fundamental reasons to utilize a multicloud architecture is avoiding vendor lock-in that is caused by forming dependencies on only a single cloud provider.

When using multicloud architectures, cloud consumers commonly select providers for specific resources or services, based on advantages or benefits they might have over others.

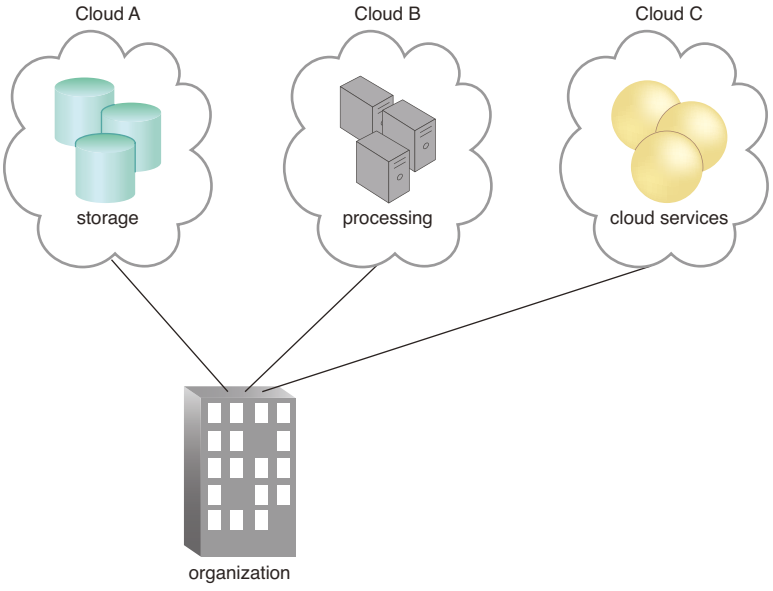


Figure 13.19
An organization uses different types of resources from different clouds, taking advantage of those resources that each cloud is better at and avoiding vendor lock-in.

Reasons for selecting one cloud provider over another can be any of the following:

- *Geographical* – when the physical locations of resources require cloud consumers to use local cloud providers for regulatory purposes
- *Economic* – prices or billing models
- *Operational* – seeking higher capacity, more resiliency, or better peformance
- *Functional* – looking for more features, specific capabilities required by the cloud consumer, or better quality in general

For cloud consumers to be able to make use of IT resources distributed across different clouds, the cloud resource administrator uses a centralized remote administration system mechanism that connects to the management systems of each individual cloud provider via their respective APIs (Figure 13.20). This allows the cloud consumer to manage all cloud-based IT resources from a central location and then use and access them as easily as if they were coming from a single cloud.

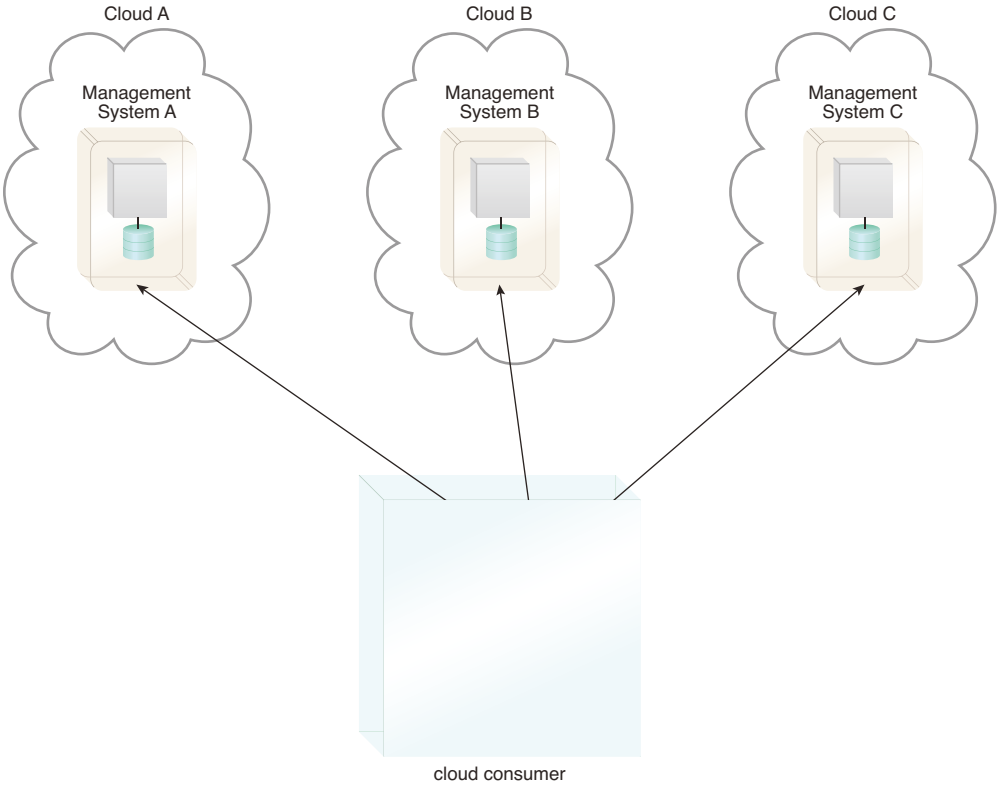


Figure 13.20
A cloud resource administrator utilizes a remote administration system mechanism to connect to the individual management systems of each different cloud provider in order to manage their resources from a central management location.

The ultimate business benefits resulting from the use of a multicloud architecture can be very different for each individual cloud consumer. Whether the goal of an organization is to maximize business agility, increase the speed at which it delivers new products, or optimize its cloud applications and automated operations, a multicloud architecture will give the organization the flexibility to mix and match cloud-based products, innovations, and services from multiple, competing cloud providers.

13.10 CASE STUDY EXAMPLE

An in-house solution that ATN did not migrate to the cloud is the Remote Upload Module, a program that is used by their clients to upload accounting and legal documents to a central archive on a daily basis. Usage peaks occur without warning, since the quantity of documents received on a day-by-day basis is unpredictable.

The Remote Upload Module currently rejects upload attempts when it is operating at capacity, which is problematic for users who need to archive certain documents before the end of a business day or prior to a deadline.

ATN decides to take advantage of its cloud-based environment by creating a cloud-bursting architecture around the on-premises Remote Upload Module service implementation. This enables it to burst out into the cloud whenever on-premises processing thresholds are exceeded (Figures 13.21 and 13.22).

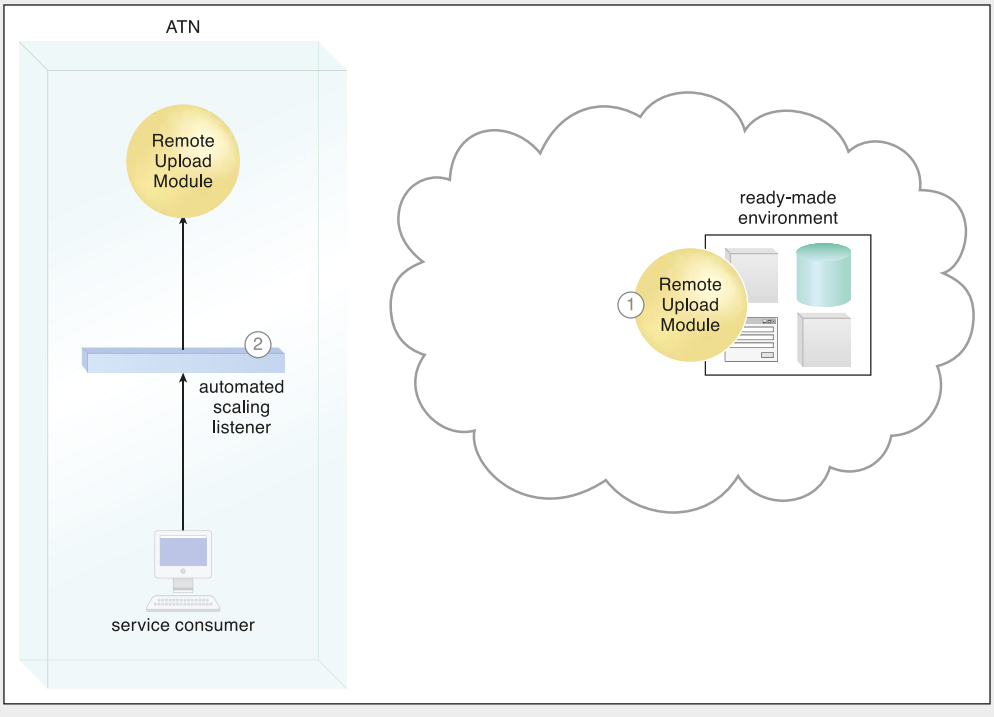


Figure 13.21
A cloud-based version of the on-premises Remote Upload Module service is deployed on ATN's leased ready-made environment (1). The automated scaling listener monitors service consumer requests (2).

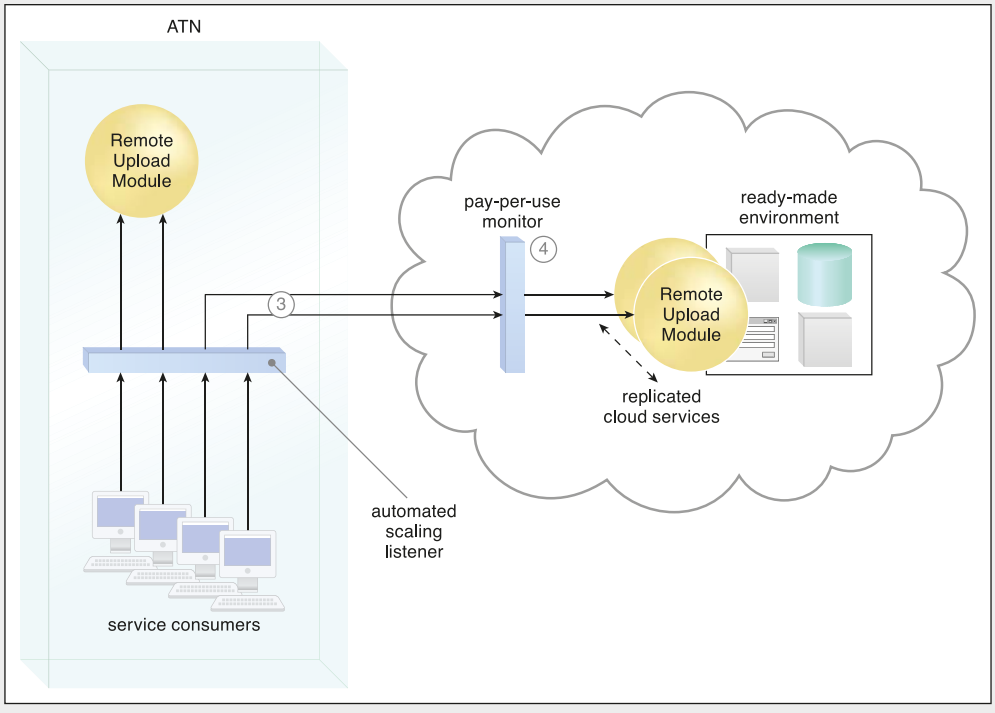


Figure 13.22

The automated scaling listener detects that service consumer usage has exceeded the local Remote Upload Module service's usage threshold and begins diverting excess requests to the cloud-based Remote Upload Module implementation (3). The cloud provider's pay-per-use monitor tracks the requests received from the on-premises automated scaling listener to collect billing data, and Remote Upload Module cloud service instances are created on-demand via resource replication (4).

A "burst in" system is invoked after the service usage has decreased enough so that service consumer requests can be processed by the on-premises Remote Upload Module implementation again. Instances of the cloud services are released, and no additional cloud-related usage fees are incurred.

This page intentionally left blank

Chapter 14

Advanced Cloud Architectures

The following cloud technology architectures are explored in this chapter:

- Hypervisor Clustering
- Virtual Server Clustering
- Load-Balanced Virtual Server Instances
- Nondisruptive Service Relocation
- Zero Downtime
- Cloud Balancing
- Resilient Disaster Recovery
- Distributed Data Sovereignty
- Resource Reservation
- Dynamic Failure Detection and Recovery
- Rapid Provisioning
- Storage Workload Management
- Virtual Private Cloud

These models represent distinct and sophisticated architectural layers, several of which can be built upon the more foundational environments established by the architectures covered in Chapter 13. For each architecture, the associated mechanisms are also documented.

14.1 Hypervisor Clustering Architecture

Hypervisors can be responsible for creating and hosting multiple virtual servers. Because of this dependency, any failure conditions that affect a hypervisor can cascade to its virtual servers (Figure 14.1).

HEARTBEATS

Heartbeats are system-level messages exchanged between hypervisors, between hypervisors and virtual servers, and between hypervisors and VIMs.

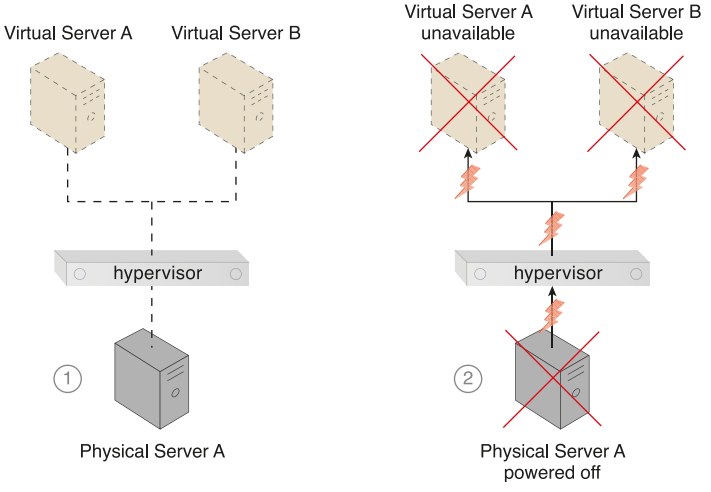


Figure 14.1

Physical Server A is hosting a hypervisor that hosts Virtual Servers A and B (1). When Physical Server A fails, the hypervisor and two virtual servers consequently fail as well (2).

The *hypervisor clustering architecture* establishes a high-availability cluster of hypervisors across multiple physical servers. If a given hypervisor or its underlying physical server becomes unavailable, the hosted virtual servers can be moved to another physical server or hypervisor to maintain runtime operations (Figure 14.2).

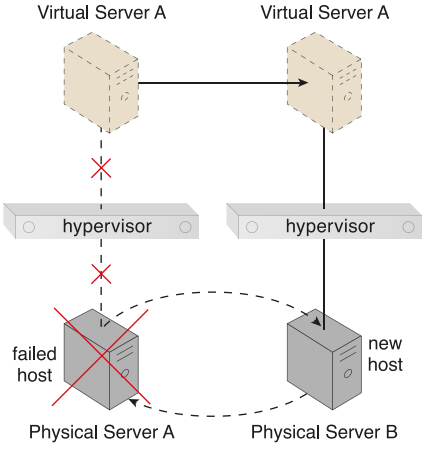
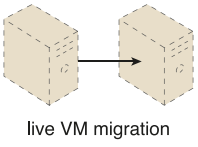


Figure 14.2
Physical Server A becomes unavailable and causes its hypervisor to fail. Virtual Server A is migrated to Physical Server B, which has another hypervisor that is part of the cluster to which Physical Server A belongs.

The hypervisor cluster is controlled via a central VIM, which sends regular heartbeat messages to the hypervisors to confirm that they are up and running. Unacknowledged heartbeat messages cause the VIM to initiate the live VM migration program to dynamically move the affected virtual servers to a new host.

LIVE VM MIGRATION

Live VM migration is a system that is capable of relocating virtual servers or virtual server instances at runtime.



The hypervisor cluster uses a shared cloud storage device to live-migrate virtual servers, as illustrated in Figures 14.3 to 14.6.

In addition to the hypervisor and resource cluster mechanisms that form the core of this architectural model and the virtual servers that are protected by the clustered environment, the following mechanisms can be incorporated:

- *Logical Network Perimeter* – The logical boundaries created by this mechanism ensure that none of the hypervisors of other cloud consumers are accidentally included in a given cluster.
- *Resource Replication* – Hypervisors in the same cluster inform one another about their status and availability. Updates on any changes that occur in the cluster, such as the creation or deletion of a virtual switch, need to be replicated to all the hypervisors via the VIM.

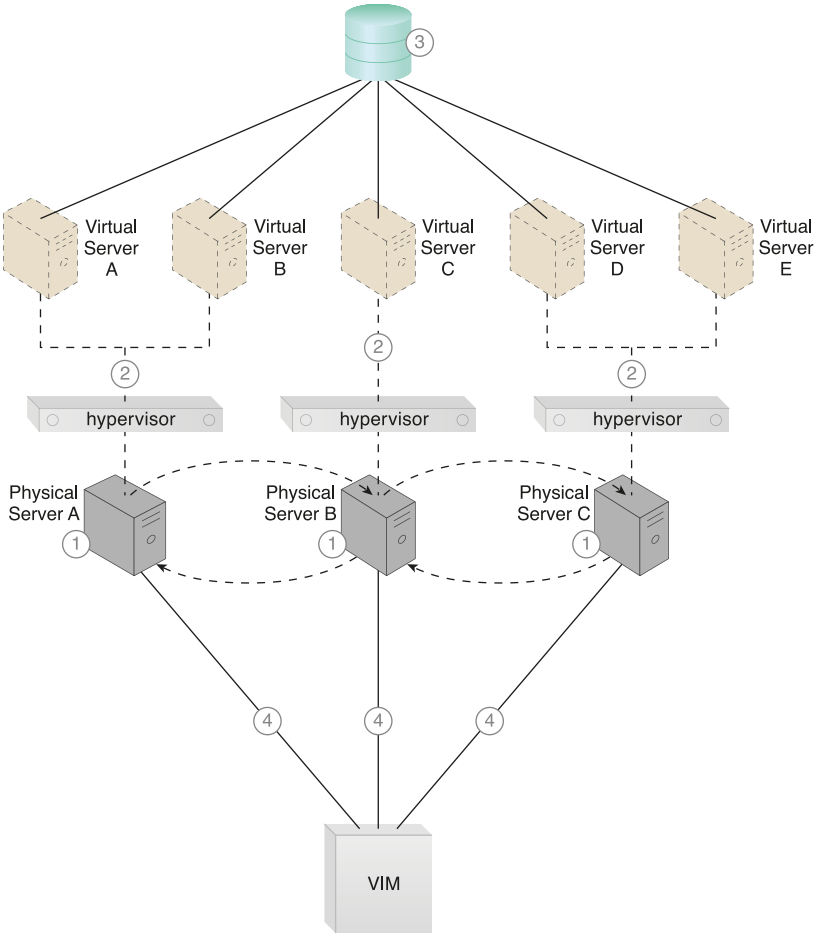


Figure 14.3

Hypervisors are installed on Physical Servers A, B, and C (1). Virtual servers are created by the hypervisors (2). A shared cloud storage device containing virtual server configuration files is positioned in a shared cloud storage device for access by all hypervisors (3). The hypervisor cluster is enabled on the three physical server hosts via a central VIM (4).

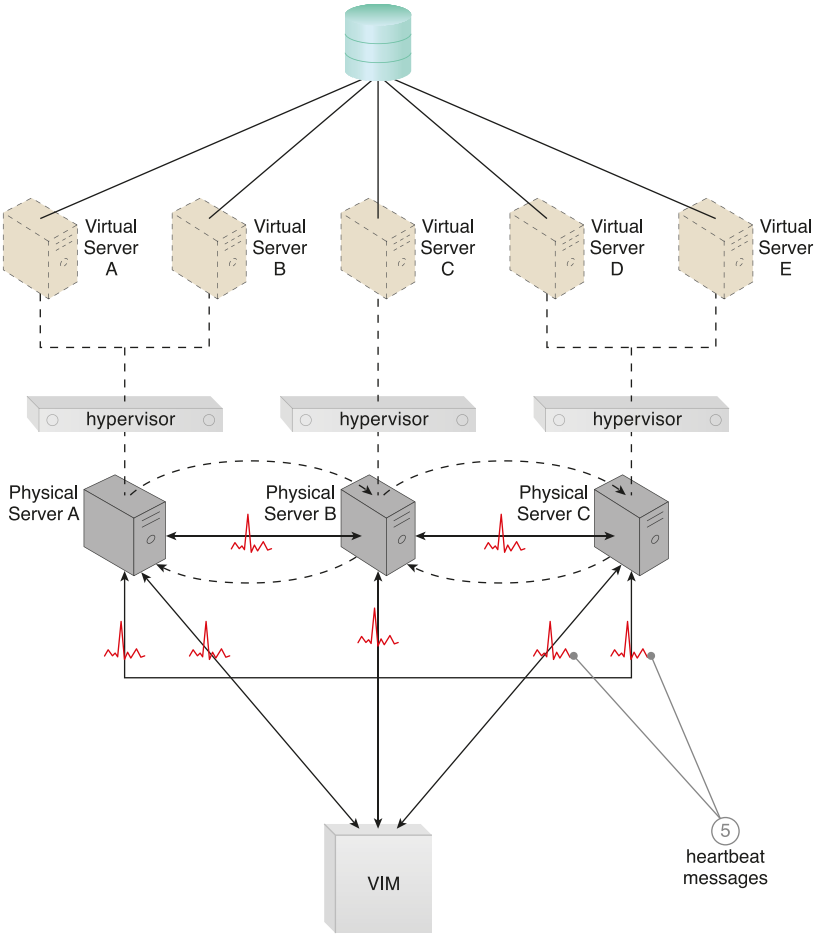


Figure 14.4

The physical servers exchange heartbeat messages with one another and the VIM according to a predefined schedule (5).

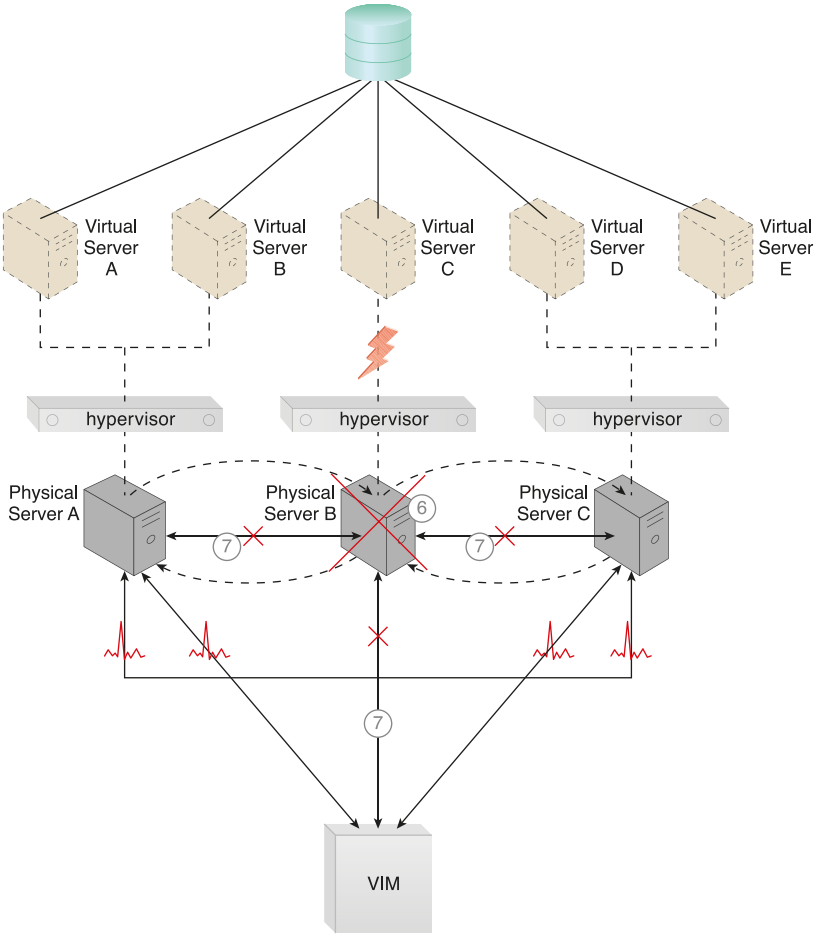


Figure 14.5

Physical Server B fails and becomes unavailable, jeopardizing Virtual Server C (6). The other physical servers and the VIM stop receiving heartbeat messages from Physical Server B (7).

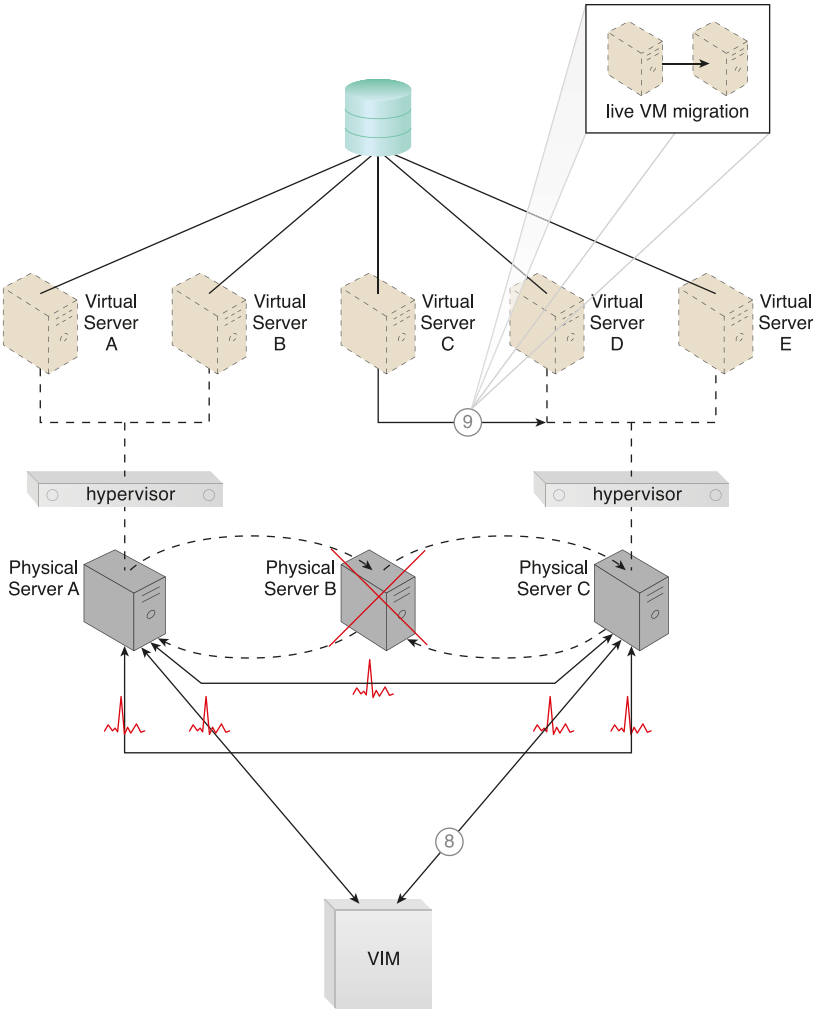


Figure 14.6

The VIM chooses Physical Server C as the new host to take ownership of Virtual Server C after assessing the available capacity of other hypervisors in the cluster (8). Virtual Server C is live-migrated to the hypervisor running on Physical Server C, where restarting may be necessary before normal operations can be resumed (9).

14.2 Virtual Server Clustering Architecture

A *virtual server clustering architecture* represents the deployment of one or more clusters of virtual servers on physical hosts running hypervisors. This architecture is focused on leveraging the efficiency, resiliency, and scalability that a cloud can provide for clusters of servers through the use of virtualization.

Individual virtual servers are instantiated on top of separate physical hosts running hypervisors (Figure 14.7). This provides the virtual infrastructure on which virtual server clusters can be configured for different purposes, such as big data analytics, service-oriented architectures, distributed NoSQL databases, and advanced container management platforms.

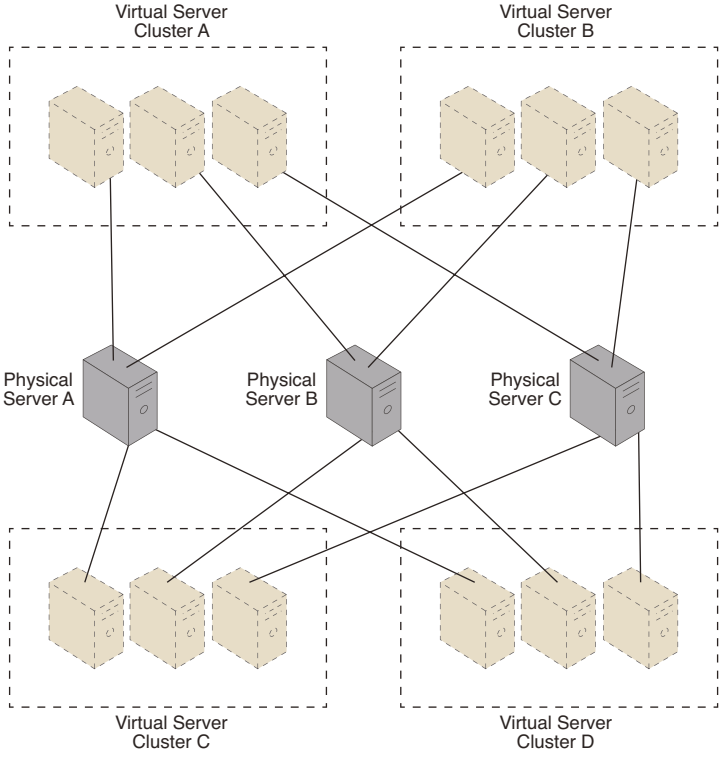


Figure 14.7

Physical Servers A, B and C are running hypervisors that allow multiple virtual servers to be hosted on each. (These virtual servers are then configured by a resource cluster mechanism into clusters of virtual servers.)

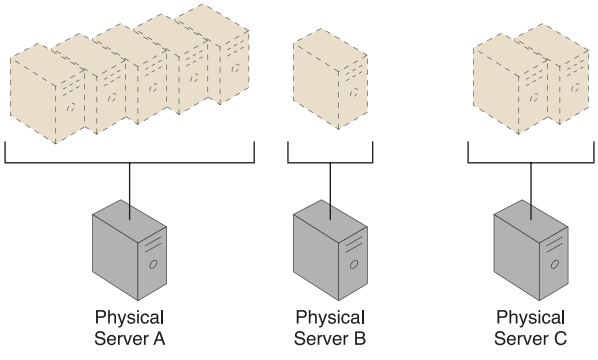
The following mechanisms can be included in this architecture, in addition to the hypervisor, resource cluster, and virtual server:

- *Logical Network Perimeter* – A logical network perimeter ensures that the virtual server cluster is enclosed in a connected environment that allows all of its nodes to communicate securely with each other.
- *Resource Replication* – Virtual servers in the same cluster inform one another about their status and availability. Updates on any changes that occur in the cluster, such as the creation or deletion of a virtual switch, need to be replicated to all virtual servers.

14.3 Load-Balanced Virtual Server Instances Architecture

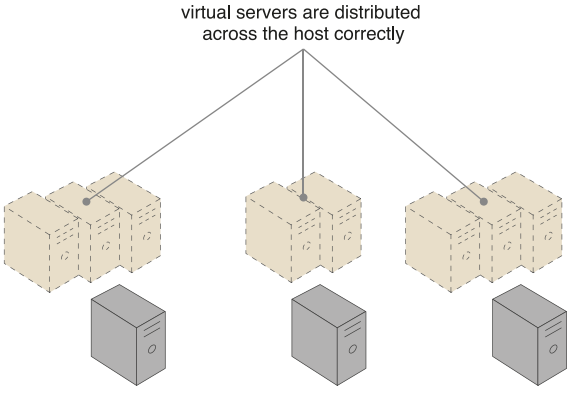
Keeping cross-server workloads evenly balanced between physical servers whose operation and management are isolated can be challenging. A physical server can easily end up hosting more virtual servers or receive larger workloads than its neighboring physical servers (Figure 14.8). Both physical server over- and underutilization can increase dramatically over time, leading to ongoing performance challenges (for overutilized servers) and constant waste (for the lost processing potential of underutilized servers).

Figure 14.8
Three physical servers have to host different quantities of virtual server instances, leading to both overutilized and underutilized servers.



The *load-balanced virtual server instances architecture* establishes a capacity watchdog system that dynamically calculates virtual server instances and associated workloads, before distributing the processing across available physical server hosts (Figure 14.9).

Figure 14.9
The virtual server instances are more evenly distributed across the physical server hosts.



The capacity watchdog system is comprised of a capacity watchdog cloud usage monitor, the live VM migration program, and a capacity planner. The capacity watchdog monitor tracks physical and virtual server usage and reports any significant fluctuations to the capacity planner, which is responsible for dynamically calculating physical server computing capacities against virtual server capacity requirements. If the capacity planner decides to move a virtual server to another host to distribute the workload, the live VM migration program is signaled to move the virtual server (Figures 14.10 to 14.12).

Figure 14.10
The hypervisor cluster architecture provides the foundation upon which the load-balanced virtual server instances architecture is built (1). Policies and thresholds are defined for the capacity watchdog monitor (2), which compares physical server capacities with virtual server processing (3). The capacity watchdog monitor reports an over-utilization to the VIM (4).

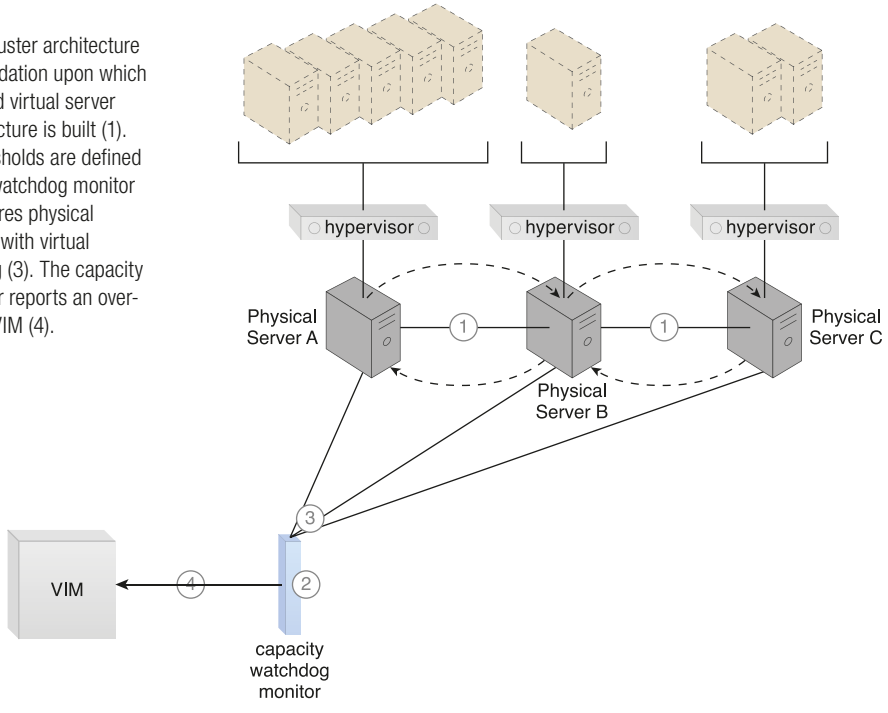


Figure 14.11

The VIM signals the load balancer to redistribute the workload based on predefined thresholds (5). The load balancer initiates the live VM migration program to move the virtual servers (6). Live VM migration moves the selected virtual servers from one physical host to another (7).

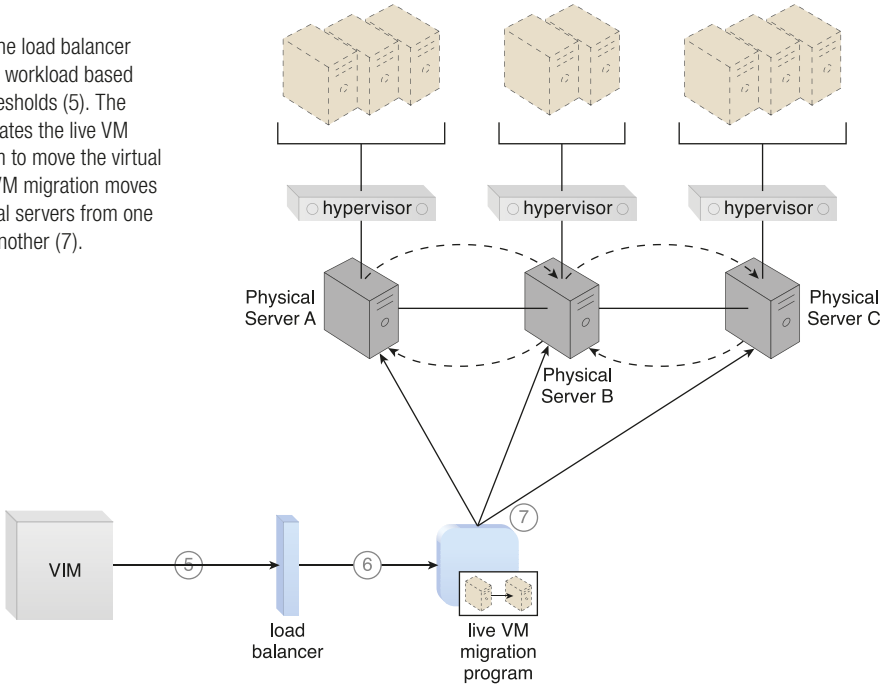
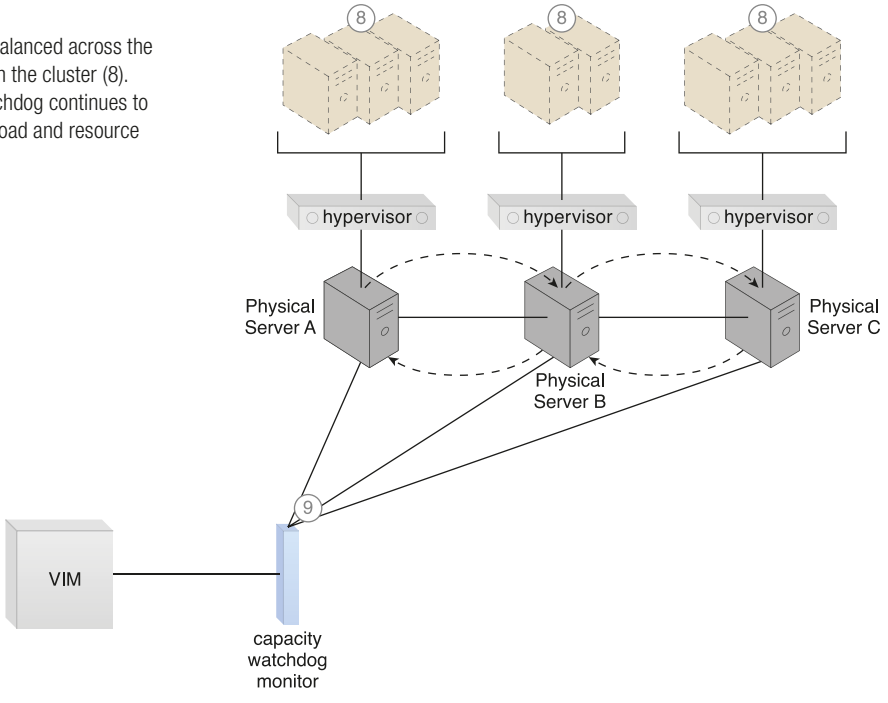


Figure 14.12

The workload is balanced across the physical servers in the cluster (8). The capacity watchdog continues to monitor the workload and resource consumption (9).



The following mechanisms can be included in this architecture, in addition to the hypervisor, resource clustering, virtual server, and (capacity watchdog) cloud usage monitor:

- *Automated Scaling Listener* – The automated scaling listener may be used to initiate the process of load balancing and to dynamically monitor workload coming to the virtual servers via the hypervisors.
- *Load Balancer* – The load balancer mechanism is responsible for distributing the workload of the virtual servers between the hypervisors.
- *Logical Network Perimeter* – A logical network perimeter ensures that the destination of a given relocated virtual server is in compliance with SLA and privacy regulations.
- *Resource Replication* – The replication of virtual server instances may be required as part of the load balancing functionality.

14.4 Nondisruptive Service Relocation Architecture

A cloud service can become unavailable for a number of reasons, such as:

- runtime usage demands that exceed its processing capacity
- a maintenance update that mandates a temporary outage
- permanent migration to a new physical server host

Cloud service consumer requests are usually rejected if a cloud service becomes unavailable, which can potentially result in exception conditions. Rendering the cloud service temporarily unavailable to cloud consumers is not preferred, even if the outage is planned.

The *nondisruptive service relocation architecture* establishes a system by which a predefined event triggers the duplication or migration of a cloud service implementation at runtime, thereby avoiding any disruption. Instead of scaling cloud services in or out with redundant implementations, cloud service activity can be temporarily diverted to another hosting environment at runtime by adding a duplicate implementation onto a new host. Similarly, cloud service consumer requests can be temporarily redirected to a duplicate implementation when the original implementation needs to undergo a maintenance outage. The relocation of the cloud service implementation and any cloud service activity can also be permanent to accommodate cloud service migrations to new physical server hosts.

A key aspect of the underlying architecture is that the new cloud service implementation is guaranteed to be successfully receiving and responding to cloud service consumer requests *before* the original cloud service implementation is deactivated or removed. A common approach is for live VM migration to move the entire virtual server instance that is hosting the cloud service. The automated scaling listener and/or load balancer mechanisms can be used to trigger a temporary redirection of cloud service consumer requests, in response to scaling and workload distribution requirements. Either mechanism can contact the VIM to initiate the live VM migration process, as shown in Figures 14.13 to 14.15.

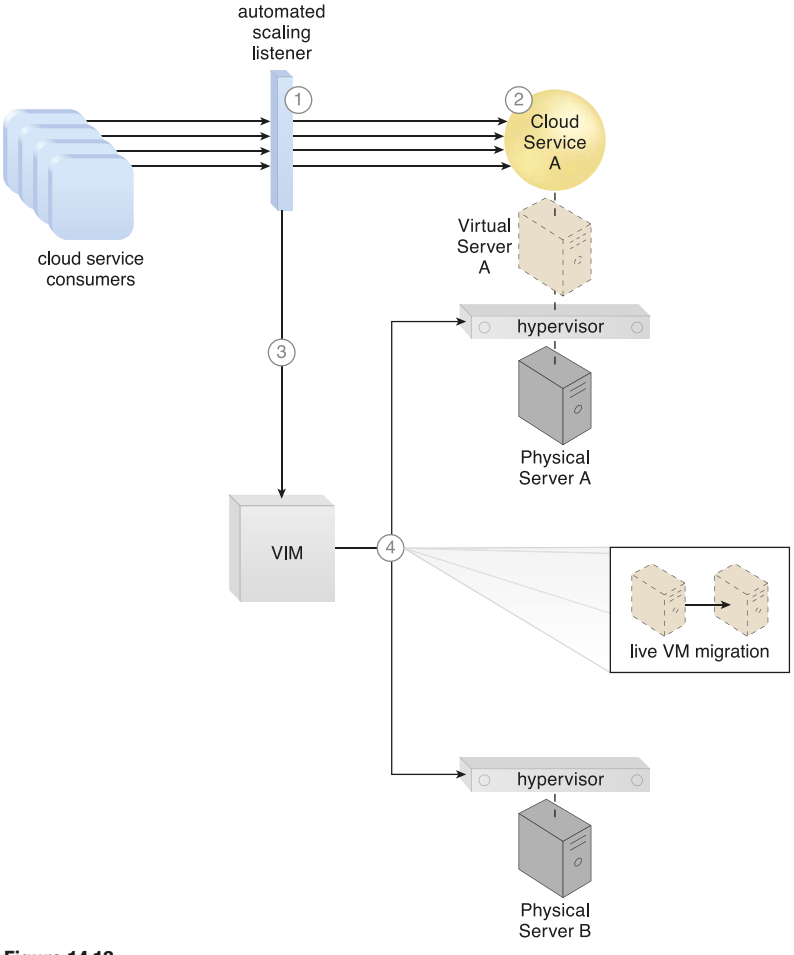


Figure 14.13

The automated scaling listener monitors the workload for a cloud service (1). The cloud service's predefined threshold is reached as the workload increases (2), causing the automated scaling listener to signal the VIM to initiate relocation (3). The VIM uses the live VM migration program to instruct both the origin and destination hypervisors to carry out runtime relocation (4).

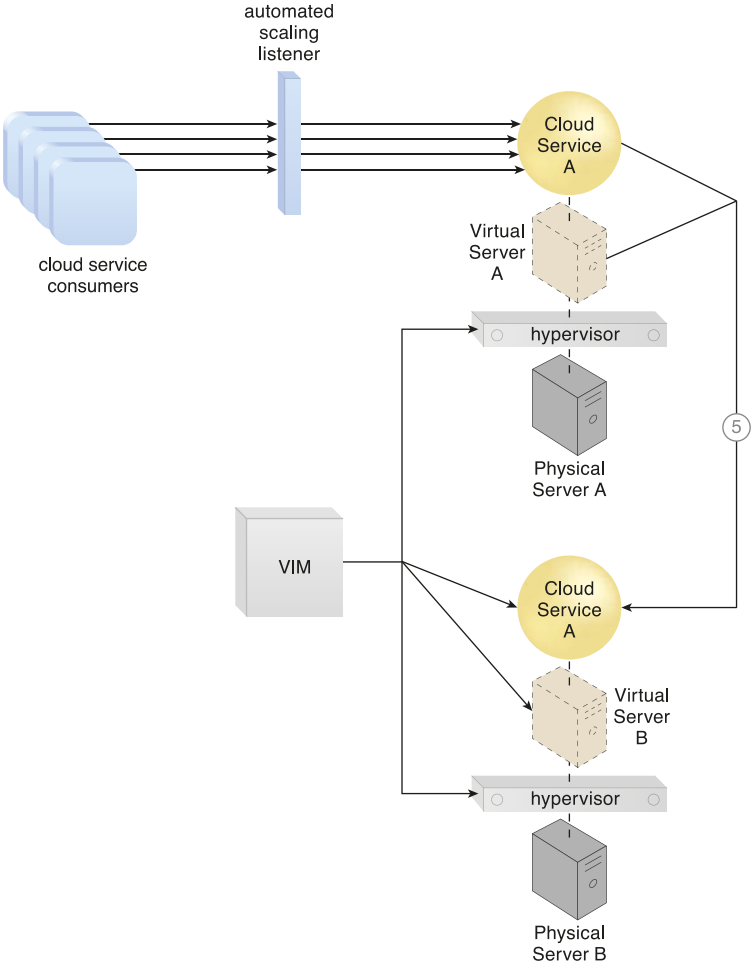


Figure 14.14

A second copy of the virtual server and its hosted cloud service are created via the destination hypervisor on Physical Server B (5).

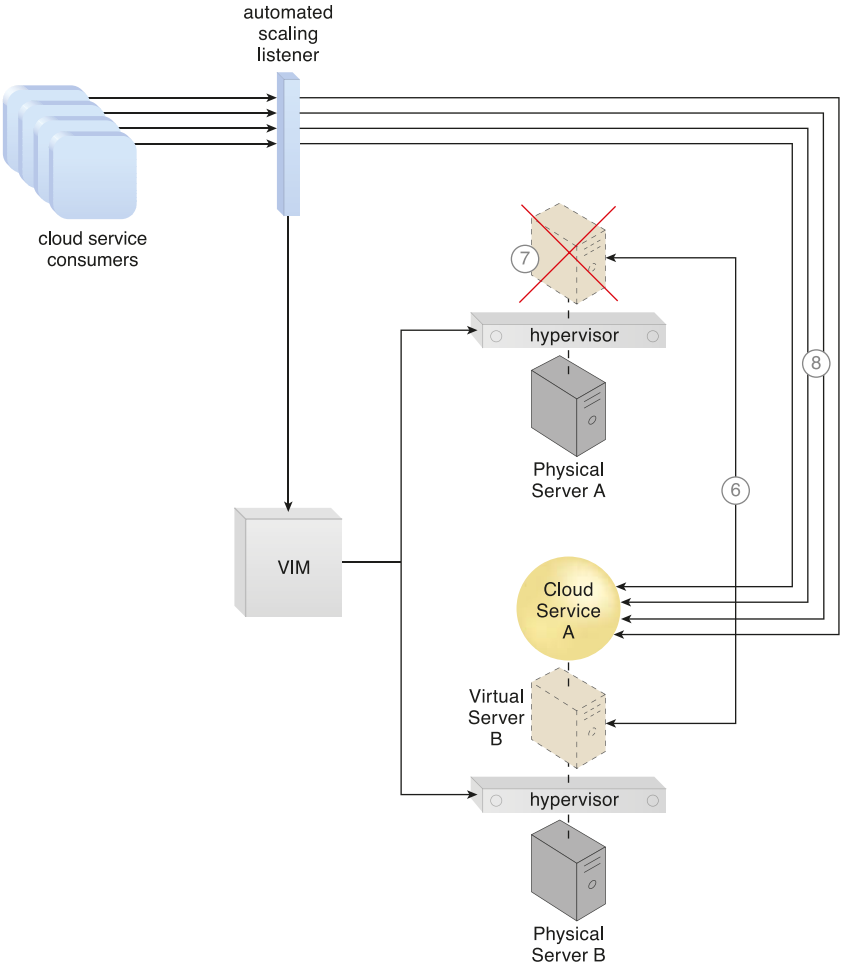


Figure 14.15

The state of both virtual server instances is synchronized (6). The first virtual server instance is removed from Physical Server A after cloud service consumer requests are confirmed to be successfully exchanged with the cloud service on Physical Server B (7). Cloud service consumer requests are now only sent to the cloud service on Physical Server B (8).

Virtual server migration can occur in one of the following two ways, depending on the location of the virtual server's disks and configuration:

- A copy of the virtual server disks is created on the destination host, if the virtual server disks are stored on a local storage device or non-shared remote storage devices attached to the source host. After the copy has been created, both virtual

server instances are synchronized and the virtual server files are removed from the origin host.

- Copying the virtual server disks is unnecessary if the virtual server's files are stored on a remote storage device that is shared between the origin and destination hosts. Ownership of the virtual server is simply transferred from the origin to the destination physical server host, and the virtual server's state is automatically synchronized.

This architecture can be supported by the persistent virtual network configurations architecture, so that the defined network configurations of migrated virtual servers are preserved to retain connection with the cloud service consumers.

Besides the automated scaling listener, load balancer, cloud storage device, hypervisor, and virtual server, other mechanisms that can be part of this architecture include the following:

- *Cloud Usage Monitor* – Different types of cloud usage monitors can be used to continuously track IT resource usage and system activity.
- *Pay-Per-Use Monitor* – The pay-per-use monitor is used to collect data for service usage cost calculations for IT resources at both the source and destination locations.
- *Resource Replication* – The resource replication mechanism is used to instantiate the shadow copy of the cloud service at its destination.
- *SLA Management System* – This management system is responsible for processing SLA data provided by the SLA monitor to obtain cloud service availability assurances, both during and after cloud service duplication or relocation.
- *SLA Monitor* – This monitoring mechanism collects the SLA information required by the SLA management system, which may be relevant if availability guarantees rely on this architecture.

NOTE

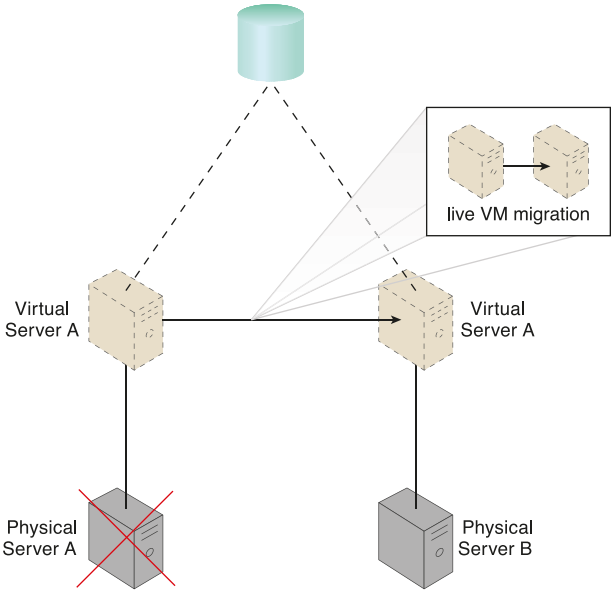
The nondisruptive service relocation technology architecture conflicts and cannot be applied together with the direct I/O access architecture covered in Chapter 15. A virtual server with direct I/O access is locked into its physical server host and cannot be moved to other hosts in this fashion.

14.5 Zero Downtime Architecture

A physical server naturally acts as a single point of failure for the virtual servers it hosts. As a result, when the physical server fails or is compromised, the availability of any (or all) hosted virtual servers can be affected. This makes the issuance of zero downtime guarantees by a cloud provider to cloud consumers challenging.

The *zero downtime architecture* establishes a sophisticated failover system that allows virtual servers to be dynamically moved to different physical server hosts, in the event that their original physical server host fails (Figure 14.16).

Figure 14.16
Physical Server A fails triggering the live VM migration program to dynamically move Virtual Server A to Physical Server B.



Multiple physical servers are assembled into a group that is controlled by a fault tolerance system capable of switching activity from one physical server to another, without interruption. The live VM migration component is typically a core part of this form of high-availability cloud architecture.

The resulting fault tolerance assures that, in case of physical server failure, hosted virtual servers will be migrated to a secondary physical server. All virtual servers are stored on a shared volume (as per the persistent virtual network configuration architecture) so that other physical server hosts in the same group can access their files.

Besides the failover system, cloud storage device, and virtual server mechanisms, the following mechanisms can be part of this architecture:

- *Audit Monitor* – This mechanism may be required to check whether the relocation of virtual servers also relocates hosted data to prohibited locations.
- *Cloud Usage Monitor* – Incarnations of this mechanism are used to monitor the actual IT resource usage of cloud consumers to help ensure that virtual server capacities are not exceeded.
- *Hypervisor* – The hypervisor of each affected physical server hosts the affected virtual servers.
- *Logical Network Perimeter* – Logical network perimeters provide and maintain the isolation that is required to ensure that each cloud consumer remains within its own logical boundary subsequent to virtual server relocation.
- *Resource Cluster* – The resource cluster mechanism is applied to create different types of active–active cluster groups that collaboratively improve the availability of virtual server-hosted IT resources.
- *Resource Replication* – This mechanism can create the new virtual server and cloud service instances upon primary virtual server failure.

14.6 Cloud Balancing Architecture

The *cloud balancing architecture* establishes a specialized architectural model in which IT resources can be load balanced across multiple clouds.

The cross-cloud balancing of cloud service consumer requests can help:

- improve the performance and scalability of IT resources
- increase the availability and reliability of IT resources
- improve load balancing and IT resource optimization

Cloud balancing functionality is primarily based on the combination of the automated scaling listener and failover system mechanisms (Figure 14.17). Many more components (and possibly other mechanisms) can be part of a complete cloud balancing architecture.

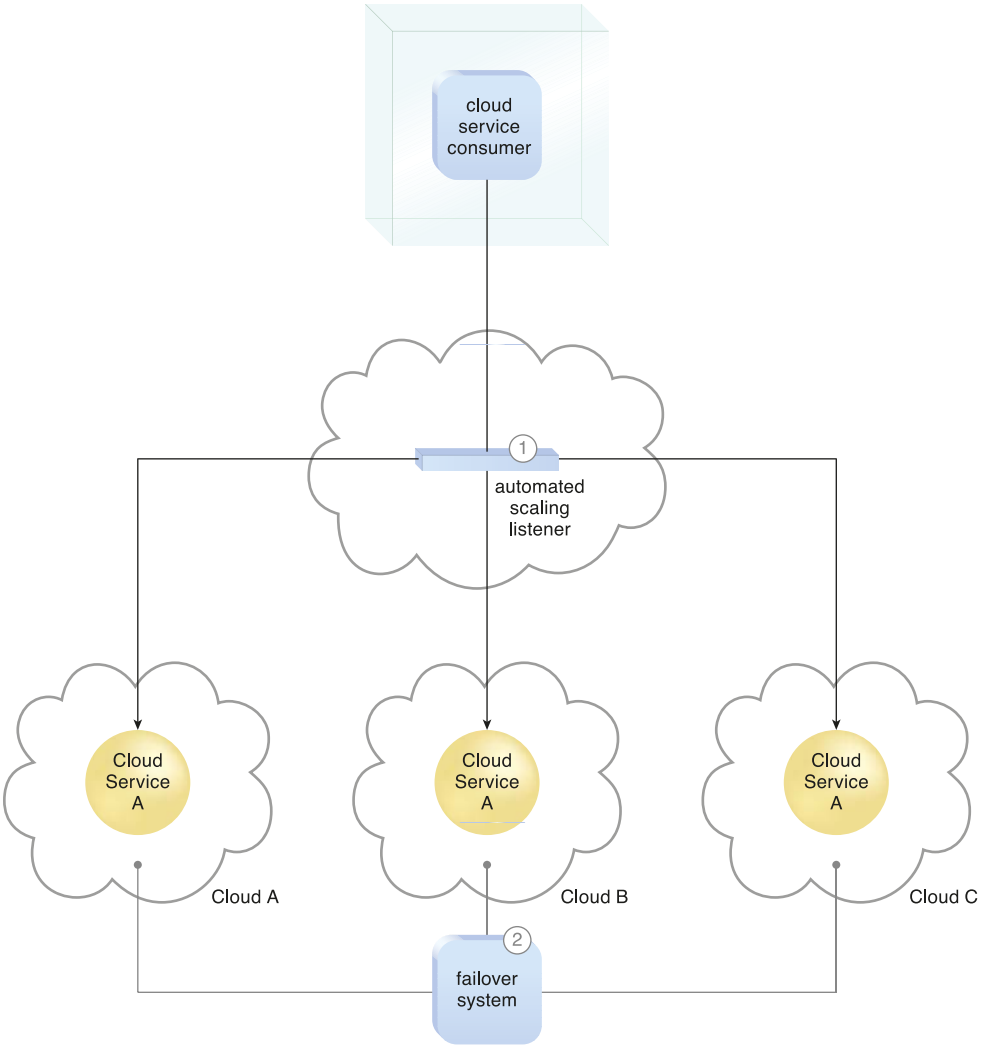


Figure 14.17

An automated scaling listener controls the cloud balancing process by routing cloud service consumer requests to redundant implementations of Cloud Service A distributed across multiple clouds (1). The failover system instills resiliency within this architecture by providing cross-cloud failover (2).

As a starting point, the two mechanisms are utilized as follows:

- The automated scaling listener redirects cloud service consumer requests to one of several redundant IT resource implementations, based on current scaling and performance requirements.
- The failover system ensures that redundant IT resources are capable of cross-cloud failover in the event of a failure within an IT resource or its underlying hosting environment. IT resource failures are announced so that the automated scaling listener can avoid inadvertently routing cloud service consumer requests to unavailable or unstable IT resources.

For a cloud balancing architecture to function effectively, the automated scaling listener needs to be aware of all redundant IT resource implementations within the scope of the cloud balanced architecture.

Note that if the manual synchronization of cross-cloud IT resource implementations is not possible, the resource replication mechanism may need to be incorporated to automate the synchronization.

14.7 Resilient Disaster Recovery Architecture

Natural or human-made disasters can occur at any time and without warning. IT enterprises can establish disaster recovery strategies to ensure that, in case an event destroys or limits the functionality of important IT systems, a secondary remote location is available with redundant implementations of those systems that can then take over. This is the purpose of the *resilient disaster recovery architecture*.

Cloud providers offer cloud-based IT resources with high levels of availability, which makes cloud environments ideal secondary sites for the protection of on-premises IT resources from disasters. The ubiquitous access and resiliency cloud characteristics support this architecture when deployed in a public cloud, since resources located there are available anytime, anywhere, and accessible by many means.

A resilient disaster recovery architecture uses resource replication mechanisms to create redundant copies of all the critical resources in an enterprise technology architecture. These copies are then placed in a remote location, where they are expected to stay synchronized with their original copies, ready to replace the originals in case a major catastrophe happens in the original location (Figure 14.18).

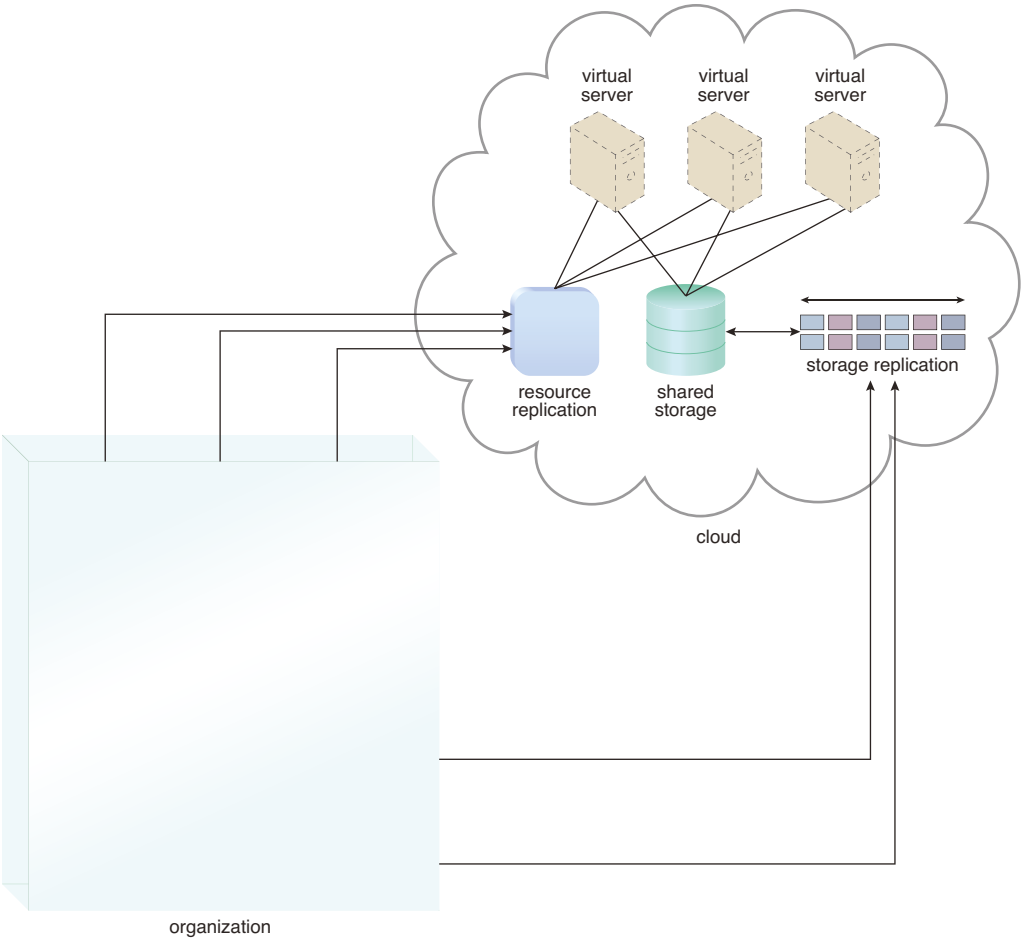


Figure 14.18

An organization uses a resource replication mechanism to create duplicate virtual instances of its physical infrastructure in a public cloud. The storage replication mechanism synchronizes on-premises data sources with their duplicates in the cloud.

The resource replication mechanism keeps the replicated IT resources in the cloud-based section of the architecture in constant synchronization with its original copies. Other mechanisms that can be part of this architecture include the following:

- *Hypervisor* – The hypervisor mechanism allows physical hosts in the cloud environment selected for redundancy to host virtual servers that are replicas of the on-premises physical or virtual servers.

- *Virtual Server* – The virtual server mechanism is used to keep synchronized replicas of original on-premises physical or virtual servers in the redundant cloud architecture.
- *Cloud Storage Device* – The cloud storage device mechanism stores redundant copies of data from the original on-premises site in the replicated cloud-based site.

14.8 Distributed Data Sovereignty Architecture

Regulations regarding the appropriate governance of data, particularly personal data, can vary across different countries and regions. Typically, such regulations require data holders to ensure that the data protected by the regulation is physically located within a particular geographical boundary. Usually cloud consumers are considered the official data holders of cloud-based data, whereas cloud providers are not commonly required to comply with these types of regulations.

Cloud providers commonly use sophisticated data replication systems to achieve redundancy levels that allow them to provide high availability for the cloud storage services they offer. The replicas are often geographically distributed to guarantee the highest level of availability possible because this distribution provides a higher degree of isolation from potential failures.

However, geographical distribution might result in a cloud provider keeping copies of protected data in locations that may be in violation of data protection regulations with which its cloud consumers are required to comply. The *distributed data sovereignty architecture* is a model that can be used to avoid this situation by ensuring that distributed data is stored in compliance with regulations. This architecture is designed to guarantee that protected data is stored in one or more specific physical locations.

An important design consideration for the distributed data sovereignty architecture is making sure that the data replication mechanisms used by the cloud provider can be configured for regulatory compliance. This architecture further relies on a data governance management mechanism to coordinate the appropriate storage of protected data within the region necessary to comply with different local or regional regulations (Figure 14.19).

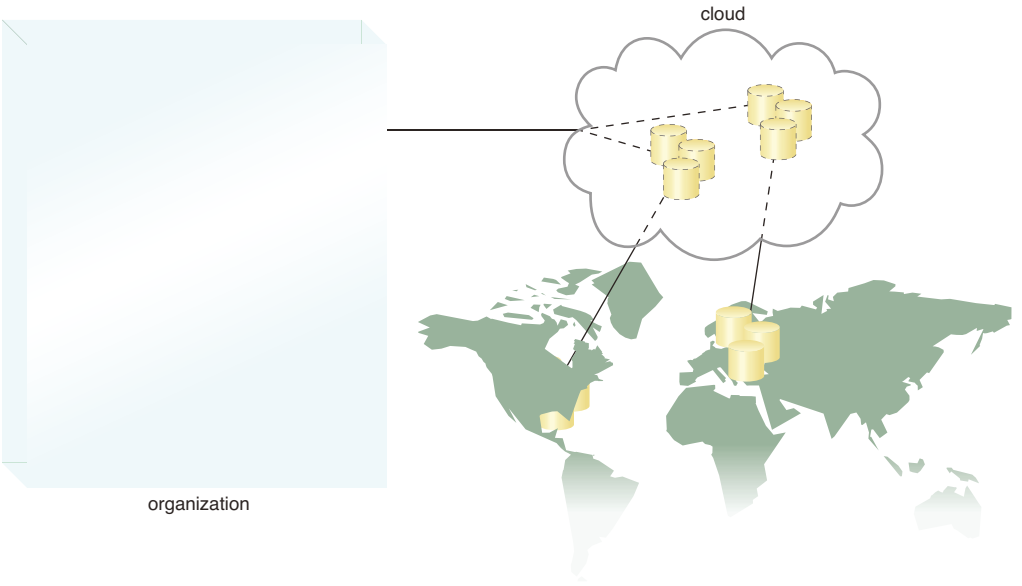


Figure 14.19
An organization uses a data governance manager mechanism to ensure that its cloud-based data is located in the region in which it must reside according to regional data protection regulations.

Additionally, the following mechanisms are part of this architecture:

- *Cloud Storage Device* – The cloud storage device mechanism stores the protected data in the location that allows the organization to comply with regional regulations.
- *Audit Monitor* – This mechanism may be required to check whether the local data has been replicated to prohibited locations.
- *Storage Replication* – The storage replication mechanism keeps copies of data made for resiliency purposes in storage devices that are geographically located in accordance with data protection regulations.

NOTE

An alternative approach is for cloud consumers to identify local cloud providers in every different region for which regulatory compliance is necessary, thereby establishing a multicloud architecture (as described in Chapter 13) in which each cloud belongs to a different cloud provider.

14.9 Resource Reservation Architecture

Depending on how IT resources are designed for shared usage and depending on their available levels of capacity, concurrent access can lead to a runtime exception condition called *resource constraint*. A resource constraint is a condition that occurs when two or more cloud consumers have been allocated to share an IT resource that does not have the capacity to accommodate the total processing requirements of those cloud consumers. As a result, one or more of the cloud consumers encounter degraded performance or may be rejected altogether. The cloud service itself may go down, resulting in all cloud consumers being rejected.

Other types of runtime conflicts can occur when an IT resource (especially one not specifically designed to accommodate sharing) is concurrently accessed by different cloud service consumers. For example, nested and sibling resource pools introduce the notion of *resource borrowing*, whereby one pool can temporarily borrow IT resources from other pools. A runtime conflict can be triggered when the borrowed IT resource is not returned due to prolonged usage by the cloud service consumer that is borrowing it. This can inevitably lead back to the occurrence of resource constraints.

The *resource reservation architecture* establishes a system whereby one of the following is set aside exclusively for a given cloud consumer (Figures 14.20 to 14.22):

- single IT resource
- portion of an IT resource
- multiple IT resources

This protects cloud consumers from each other by avoiding the aforementioned resource constraint and resource borrowing conditions.

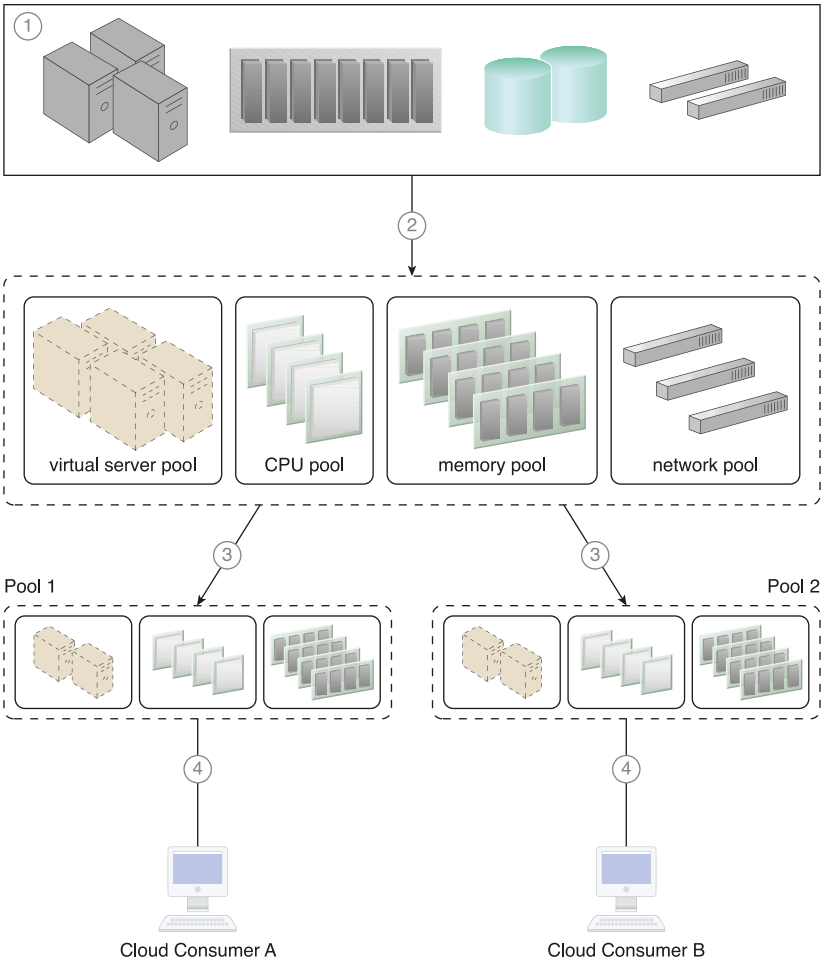


Figure 14.20

A physical resource group is created (1), from which a parent resource pool is created as per the resource pooling architecture (2). Two smaller child pools are created from the parent resource pool, and resource limits are defined using the resource management system (3). Cloud consumers are provided with access to their own exclusive resource pools (4).

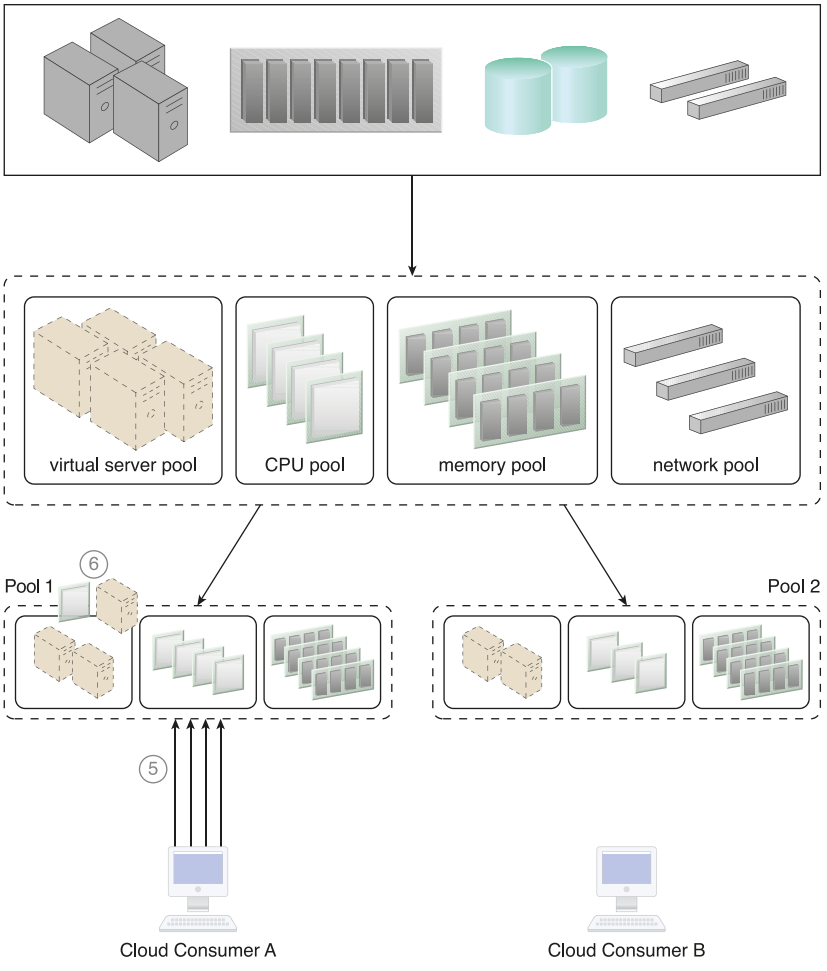


Figure 14.21

An increase in requests from Cloud Consumer A results in more IT resources being allocated to that cloud consumer (5), meaning some IT resources need to be borrowed from Pool 2. The amount of borrowed IT resources is confined by the resource limit that was defined in Step 3 to ensure that Cloud Consumer B will not face any resource constraints (6).

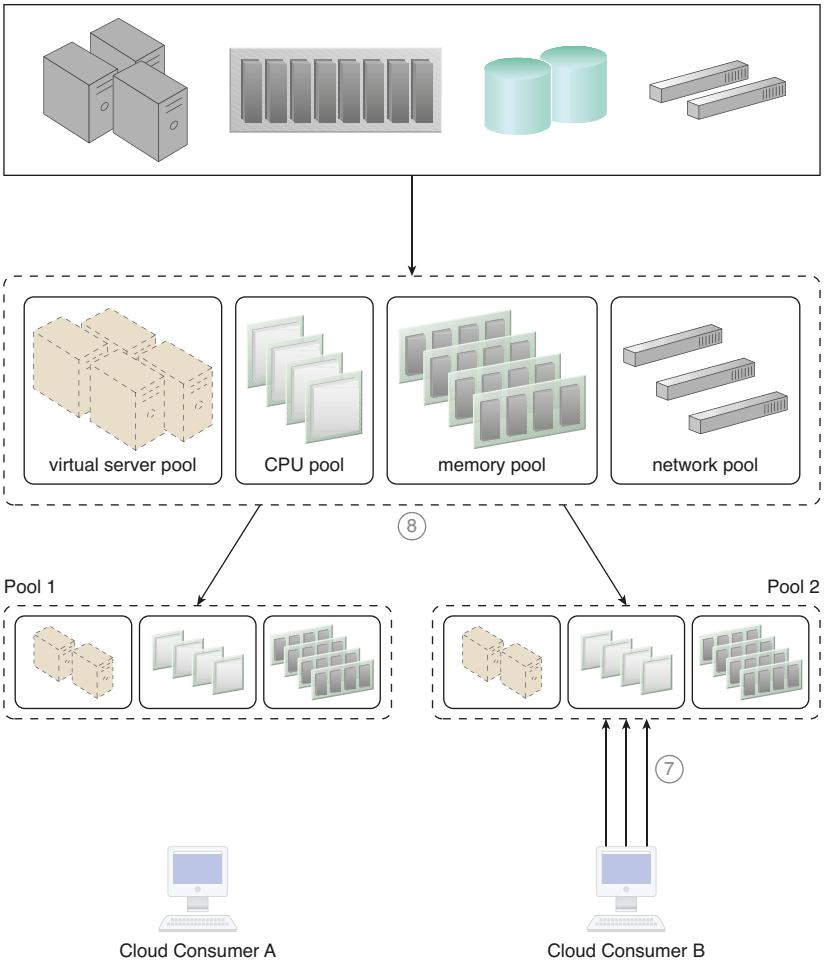


Figure 14.22

Cloud Consumer B now imposes more requests and usage demands and may soon need to utilize all available IT resources in the pool (7). The resource management system forces Pool 1 to release the IT resources and move them back to Pool 2 to become available for Cloud Consumer B (8).

The creation of an IT resource reservation system can require the involvement of the resource management system mechanism, which is used to define the usage thresholds for individual IT resources and resource pools. Reservations lock the amount of IT resources that each pool needs to keep, with the balance of the pool's IT resources still available for sharing and borrowing. The remote administration system mechanism is

also used to enable front-end customization, so that cloud consumers have administration controls for the management of their reserved IT resource allocations.

The types of mechanisms that are commonly reserved within this architecture are cloud storage devices and virtual servers. Other mechanisms that may be part of the architecture can include:

- *Audit Monitor* – The audit monitor is used to check whether the resource reservation system is complying with cloud consumer auditing, privacy, and other regulatory requirements. For example, it may track the geographical locations of reserved IT resources.
- *Cloud Usage Monitor* – A cloud usage monitor may oversee the thresholds that trigger the allocation of reserved IT resources.
- *Hypervisor* – The hypervisor mechanism may apply reservations for different cloud consumers to ensure that they are correctly allocated to their guaranteed IT resources.
- *Logical Network Perimeter* – This mechanism establishes the boundaries necessary to ensure that reserved IT resources are made exclusively available to cloud consumers.
- *Resource Replication* – This component needs to stay informed about each cloud consumer's limits for IT resource consumption to replicate and provision new IT resource instances expediently.

14.10 Dynamic Failure Detection and Recovery Architecture

Cloud-based environments can be comprised of vast quantities of IT resources that are simultaneously accessed by numerous cloud consumers. Any of those IT resources can experience failure conditions that require more than manual intervention to resolve. Manually administering and solving IT resource failures is generally inefficient and impractical.

The *dynamic failure detection and recovery architecture* establishes a resilient watchdog system to monitor and respond to a wide range of predefined failure scenarios (Figures 14.23 and 14.24). This system notifies and escalates the failure conditions that it cannot automatically resolve itself. It relies on a specialized cloud usage monitor, called the intelligent watchdog monitor, to actively track IT resources and take predefined actions in response to predefined events.

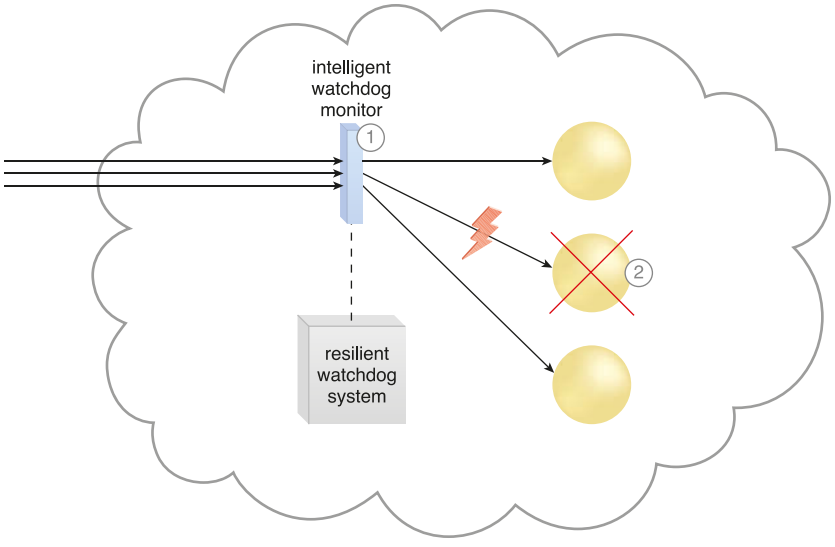


Figure 14.23

The intelligent watchdog monitor keeps track of cloud consumer requests (1) and detects that a cloud service has failed (2).

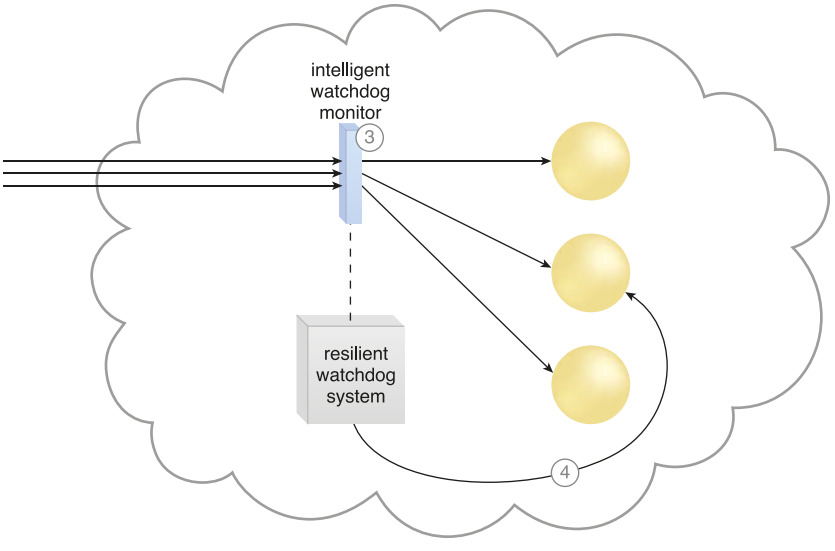


Figure 14.24

The intelligent watchdog monitor notifies the watchdog system (3), which restores the cloud service based on predefined policies. The cloud service resumes its runtime operation (4).

The resilient watchdog system performs the following five core functions:

- watching
- deciding upon an event
- acting upon an event
- reporting
- escalating

Sequential recovery policies can be defined for each IT resource to determine the steps that the intelligent watchdog monitor needs to take when a failure condition occurs. For example, a recovery policy can state that one recovery attempt needs to be automatically carried out before issuing a notification (Figure 14.25).

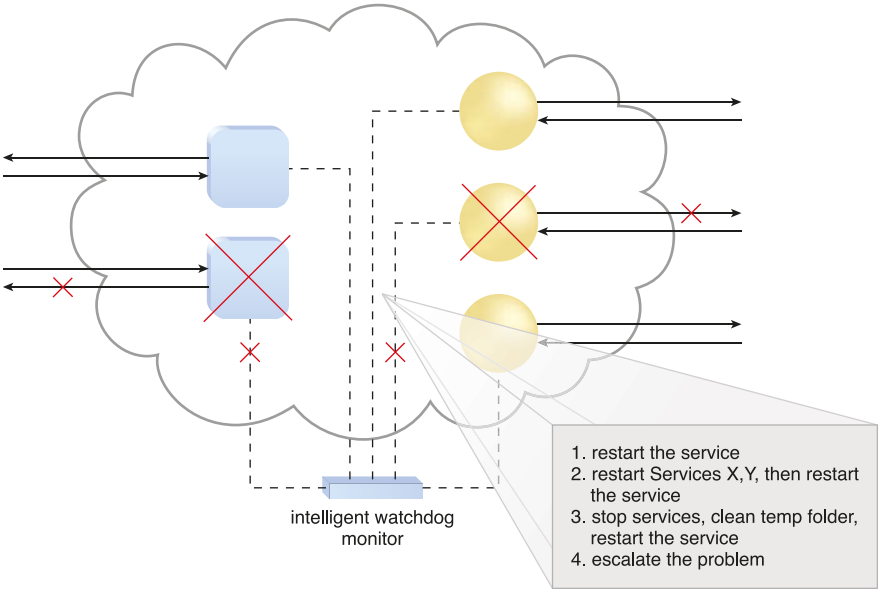


Figure 14.25

In the event of a failure, the intelligent watchdog monitor refers to its predefined policies to recover the cloud service step-by-step, escalating the process when a problem proves to be deeper than expected.

Some of the actions the intelligent watchdog monitor commonly takes to escalate an issue include:

- running a batch file
- sending a console message
- sending a text message
- sending an email message
- sending an SNMP trap
- logging a ticket

Many types of programs and products can act as intelligent watchdog monitors. Most can be integrated with standard ticketing and event management systems.

This architectural model can further incorporate the following mechanisms:

- *Audit Monitor* – This mechanism is used to track whether data recovery is carried out in compliance with legal or policy requirements.
- *Failover System* – The failover system mechanism is typically used during the initial attempts to recover failed IT resources.
- *SLA Management System* and *SLA Monitor* – Since the functionality achieved by applying this architecture is closely associated with SLA guarantees, the system commonly relies on the information that is managed and processed by these mechanisms.

14.11 Rapid Provisioning Architecture

A conventional provisioning process can involve a number of tasks that are traditionally completed manually by administrators and technology experts who prepare the requested IT resources as per prepackaged specifications or custom client requests. In cloud environments, where higher volumes of customers are serviced and where the average customer requests higher volumes of IT resources, manual provisioning processes are inadequate and can even lead to unreasonable risk due to human error and inefficient response times.

For example, a cloud consumer that requests the installation, configuration, and updating of 25 Windows servers with several applications requires that half of the applications be identical installations, while the other half be customized. Each operating

system deployment can take up to 30 minutes, followed by additional time for security patches and operating system updates that require server rebooting. Finally, the applications need to be deployed and configured. Using a manual or semi-automated approach requires excessive amounts of time and introduces a probability of human error that increases with each installation.

The *rapid provisioning architecture* establishes a system that automates the provisioning of a wide range of IT resources, either individually or as a collective. The underlying technology architecture for rapid IT resource provisioning can be sophisticated and complex, and relies on a system comprised of an automated provisioning program, rapid provisioning engine, and scripts and templates for on-demand provisioning.

Beyond the components displayed in Figure 14.26, many additional architectural artifacts are available to coordinate and automate the different aspects of IT resource provisioning, such as:

- *Server Templates* – Templates of virtual image files that are used to automate the instantiation of new virtual servers.
- *Server Images* – These images are similar to virtual server templates but are used to provision physical servers.
- *Application Packages* – Collections of applications and other software that are packaged for automated deployment.
- *Application Packager* – The software used to create application packages.
- *Custom Scripts* – Scripts that automate administrative tasks, as part of an intelligent automation engine.
- *Sequence Manager* – A program that organizes sequences of automated provisioning tasks.
- *Sequence Logger* – A component that logs the execution of automated provisioning task sequences.
- *Operating System Baseline* – A configuration template that is applied after the operating system is installed, to quickly prepare it for usage.
- *Application Configuration Baseline* – A configuration template with the settings and environmental parameters that are needed to prepare new applications for use.
- *Deployment Data Store* – The repository that stores virtual images, templates, scripts, baseline configurations, and other related data.

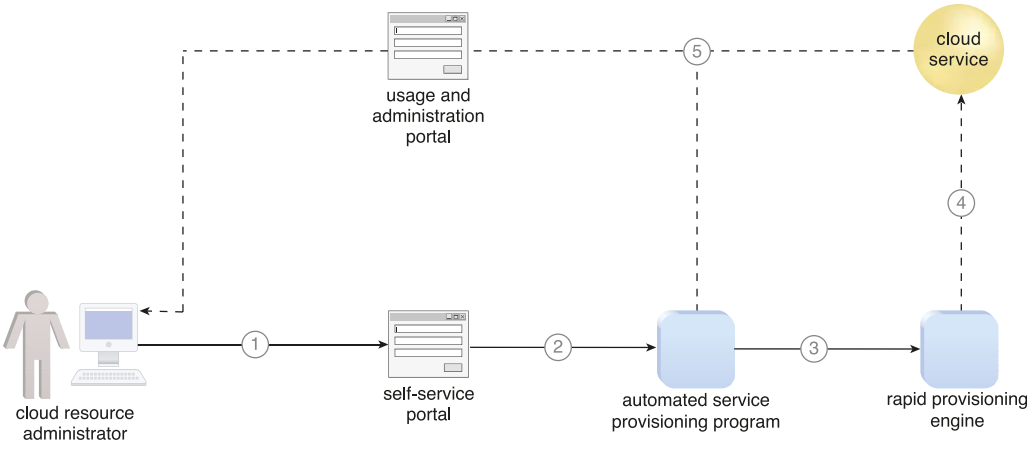


Figure 14.26

A cloud resource administrator requests a new cloud service through the self-service portal (1). The self-service portal passes the request to the automated service provisioning program installed on the virtual server (2), which passes the necessary tasks to be performed to the rapid provisioning engine (3). The rapid provisioning engine announces when the new cloud service is ready (4). The automated service provisioning program finalizes and publishes the cloud service on the usage and administration portal for cloud consumer access (5).

The following step-by-step description helps provide some insight into the inner workings of a rapid provisioning engine, involving a number of the previously listed system components:

1. A cloud consumer requests a new server through the self-service portal.
2. The sequence manager forwards the request to the deployment engine for the preparation of an operating system.
3. The deployment engine uses the virtual server templates for provisioning if the request is for a virtual server. Otherwise, the deployment engine sends the request to provision a physical server.
4. The predefined image for the requested type of operating system is used for the provisioning of the operating system, if available. Otherwise, the regular deployment process is executed to install the operating system.
5. The deployment engine informs the sequence manager when the operating system is ready.

6. The sequence manager updates and sends the logs to the sequence logger for storage.
7. The sequence manager requests that the deployment engine apply the operating system baseline to the provisioned operating system.
8. The deployment engine applies the requested operating system baseline.
9. The deployment engine informs the sequence manager that the operating system baseline has been applied.
10. The sequence manager updates and sends the logs of completed steps to the sequence logger for storage.
11. The sequence manager requests that the deployment engine install the applications.
12. The deployment engine deploys the applications on the provisioned server.
13. The deployment engine informs the sequence manager that the applications have been installed.
14. The sequence manager updates and sends the logs of completed steps to the sequence logger for storage.
15. The sequence manager requests that the deployment engine apply the application's configuration baseline.
16. The deployment engine applies the configuration baseline.
17. The deployment engine informs the sequence manager that the configuration baseline has been applied.
18. The sequence manager updates and sends the logs of completed steps to the sequence logger for storage.

The cloud storage device mechanism is used to provide storage for application baseline information, templates, and scripts, while the hypervisor rapidly creates, deploys, and hosts the virtual servers that either are provisioned themselves or host other provisioned IT resources. The resource replication mechanism is typically used to generate replicated instances of IT resources in response to rapid provisioning requirements.

14.12 Storage Workload Management Architecture

Overutilized cloud storage devices increase the workload on the storage controller and can cause a range of performance challenges. Conversely, cloud storage devices that are underutilized are wasteful due to lost processing and storage capacity potential (Figure 14.27).

LUN MIGRATION

LUN migration is a specialized storage program that is used to move LUNs from one storage device to another without interruption, while remaining transparent to cloud consumers.

LUN migration

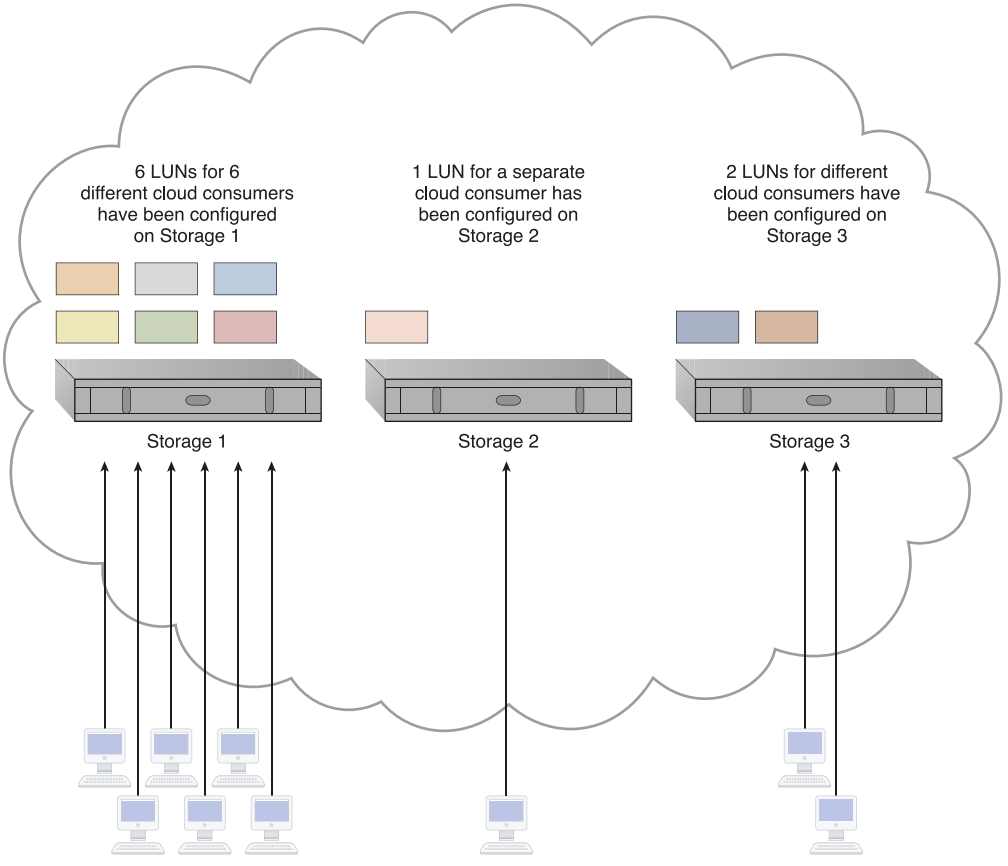


Figure 14.27

An unbalanced cloud storage architecture has six storage LUNs in Storage 1 for cloud consumers to use, while Storage 2 is hosting one LUN, and Storage 3 is hosting two. The majority of the workload ends up with Storage 1, since it is hosting the most LUNs.

The *storage workload management architecture* enables LUNs to be evenly distributed across available cloud storage devices, while a storage capacity system is established to ensure that runtime workloads are evenly distributed across the LUNs (Figure 14.28).

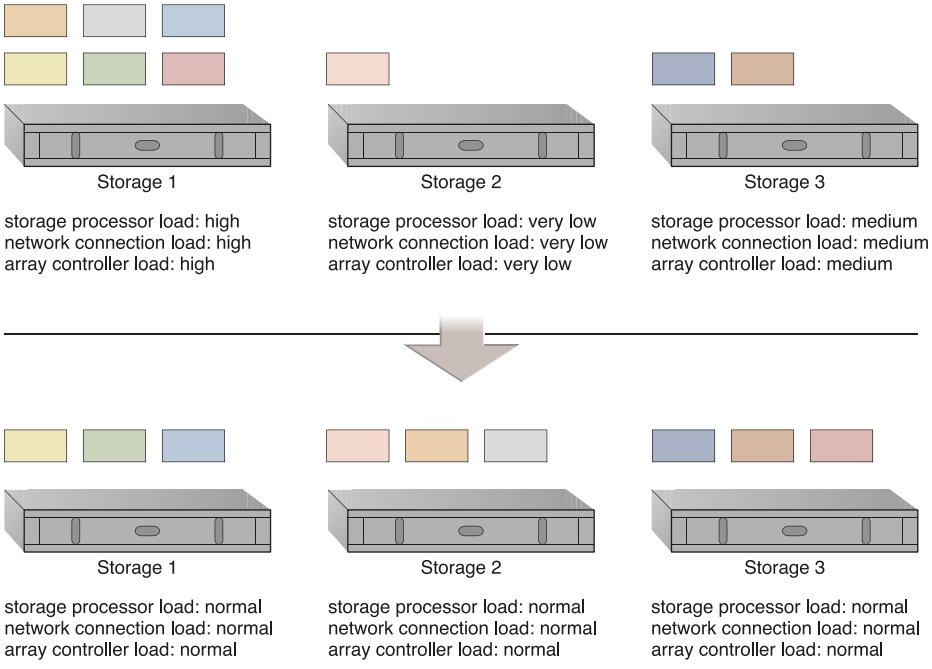


Figure 14.28

LUNs are dynamically distributed across cloud storage devices, resulting in more even distribution of associated types of workloads.

Combining cloud storage devices into a group allows LUN data to be distributed between available storage hosts equally. A storage management system is configured, and an automated scaling listener is positioned to monitor and equalize runtime workloads among the grouped cloud storage devices, as illustrated in Figures 14.29 to 14.31.

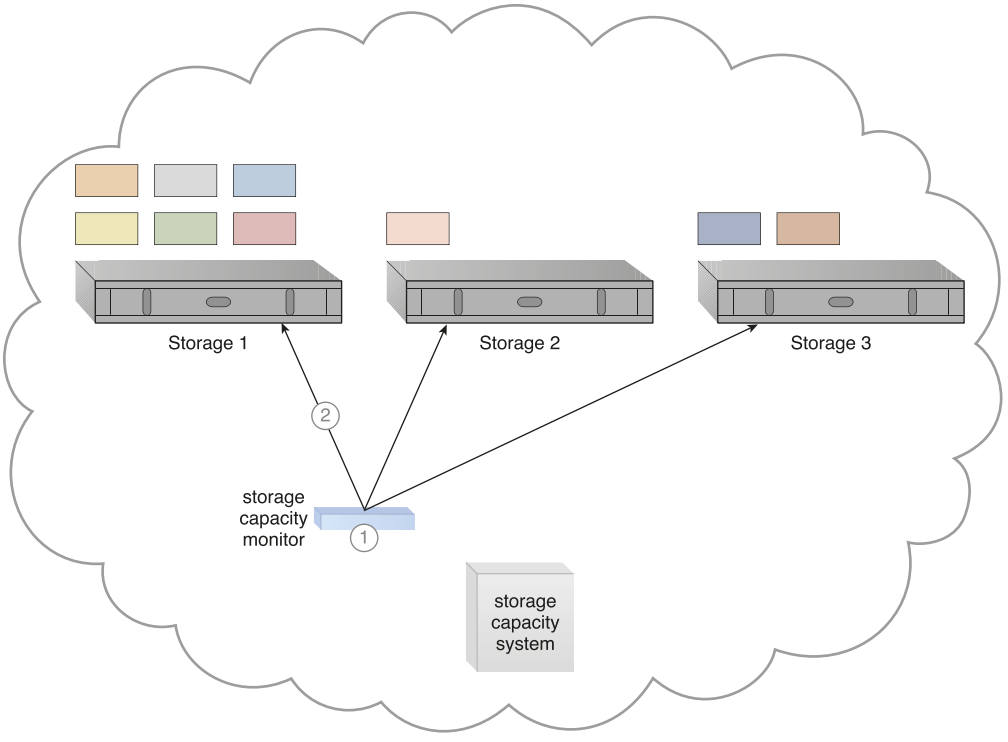


Figure 14.29

The storage capacity system and storage capacity monitor are configured to survey three storage devices in realtime, whose workload and capacity thresholds are predefined (1). The storage capacity monitor determines that the workload on Storage 1 is reaching its threshold (2).

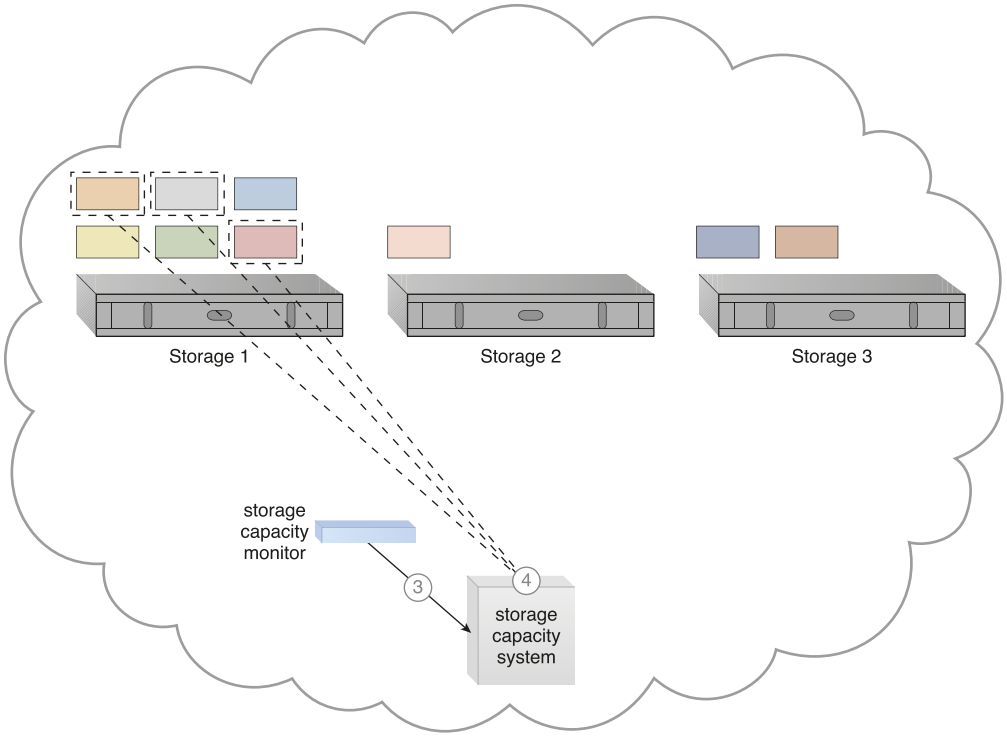


Figure 14.30

The storage capacity monitor informs the storage capacity system that Storage 1 is overutilized (3). The storage capacity system identifies the LUNs to be moved from Storage 1 (4).

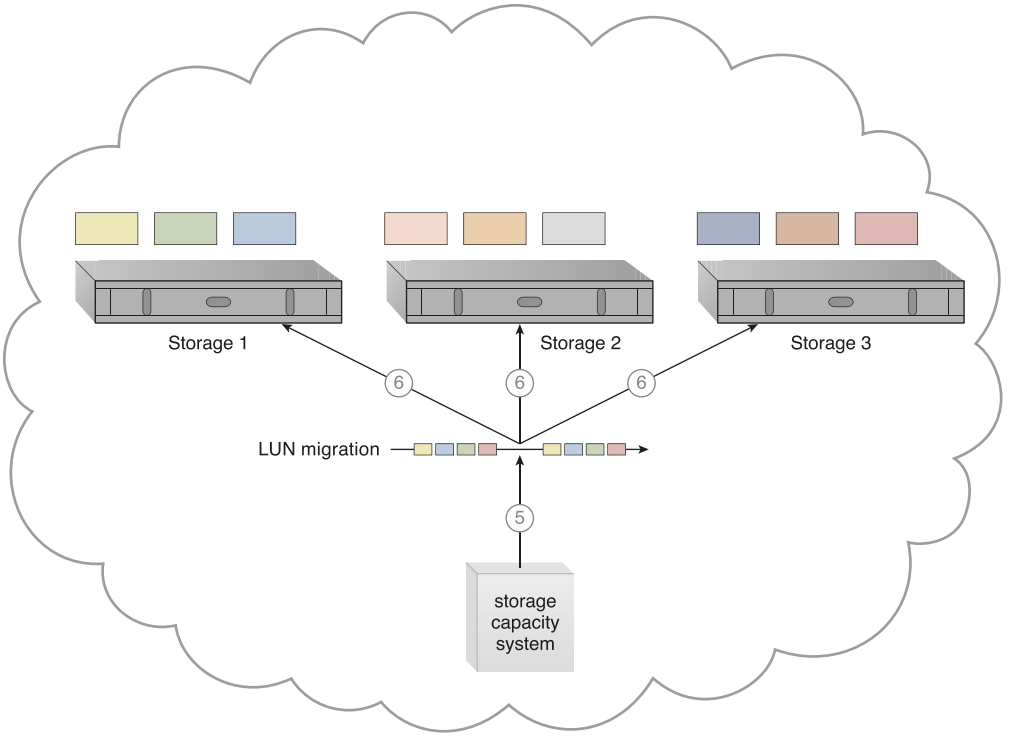


Figure 14.31

The storage capacity system calls for LUN migration to move some of the LUNs from Storage 1 to the other two storage devices (5). LUN migration transitions LUNs to Storage 2 and 3 to balance the workload (6).

The storage capacity system can keep the hosting storage device in power-saving mode for the periods when the LUNs are being accessed less frequently or only at specific times.

Some other mechanisms that can be included in the storage workload management architecture to accompany the cloud storage device are as follows:

- *Audit Monitor* – This monitoring mechanism is used to check for compliance with regulatory, privacy, and security requirements, since the system established by this architecture can physically relocate data.
- *Automated Scaling Listener* – The automated scaling listener is used to watch and respond to workload fluctuations.

- *Cloud Usage Monitor* – In addition to the capacity workload monitor, specialized cloud usage monitors are used to track LUN movements and collect workload distribution statistics.
- *Load Balancer* – This mechanism can be added to horizontally balance workloads across available cloud storage devices.
- *Logical Network Perimeter* – Logical network perimeters provide levels of isolation so that cloud consumer data that undergoes relocation remains inaccessible to unauthorized parties.

14.13 Virtual Private Cloud Architecture

The *virtual private cloud architecture* establishes a private cloud with underlying infrastructure that belongs to a public cloud provider, but that is exclusively dedicated to one specific cloud consumer for whom the private cloud is delivered. This can be useful for an organization that wants to have a private cloud but does not have the necessary infrastructure to support it on premises.

To the cloud consumer with exclusive access, this is a private cloud. However, from the cloud provider's point of view, it is part of its infrastructure, which is why it is referred to as a "virtual" private cloud. The underlying physical resources, which are typically virtualized for more efficient utilization, are not shared with other cloud consumers. Instead, they are solely dedicated to the "owner" (the cloud consumer) of the virtual private cloud.

The physical resources used to build this architecture need special isolation from the rest of the cloud provider's infrastructure, including a separate physical network to which the cloud consumer connects via a secure virtual private network (VPN), as shown in Figure 14.32. Sometimes this VPN can be replaced by a dedicated physical link from the cloud provider to the cloud consumer (although that could result in a much more expensive architecture).

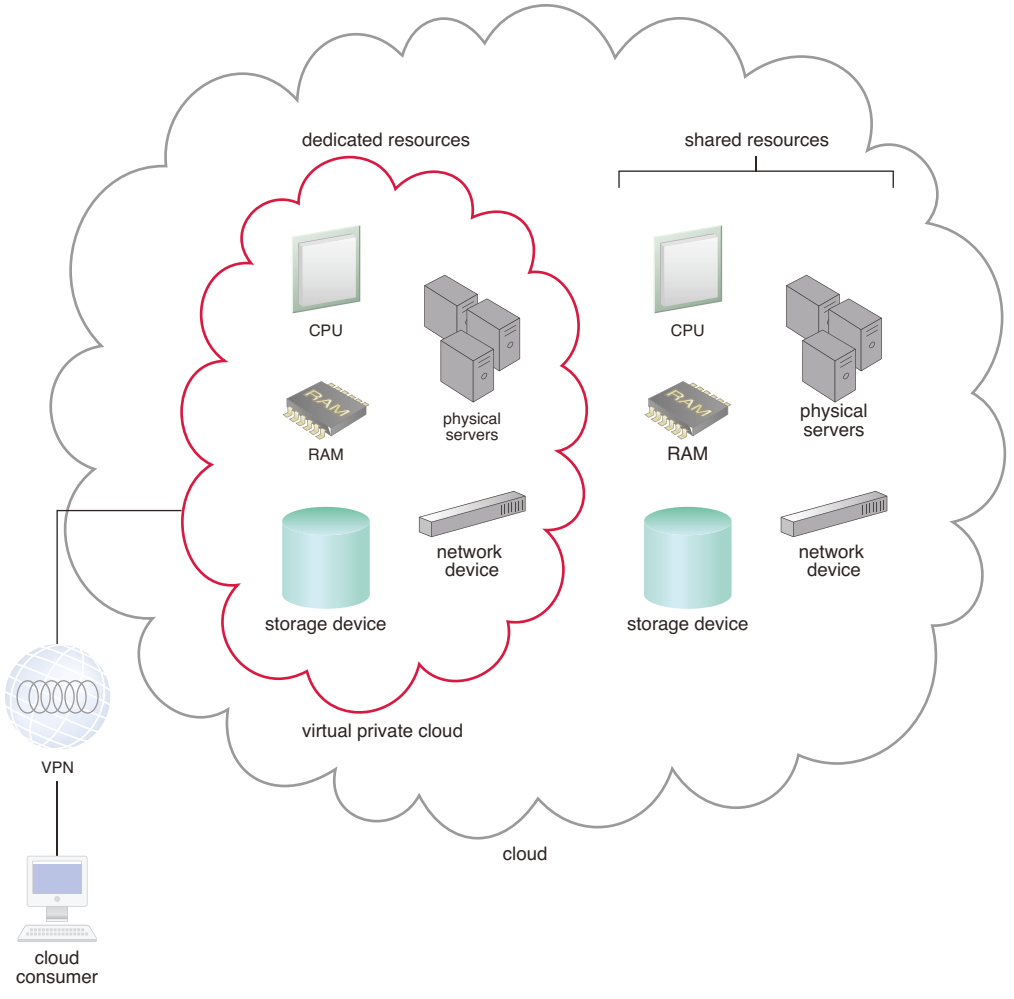


Figure 14.32

A virtual private cloud architecture utilizes physical resources from a public cloud provider dedicated for exclusive use by a specific cloud consumer, accessible via a secure connection, such as can be provided by a VPN.

The mechanisms involved in this architecture are the same mechanisms required to build any other private cloud, with the exception of the VPN, which is normally not required when a private cloud is deployed on infrastructure that resides within the physical boundary of an organization. These mechanisms include:

- *Hypervisor* – The hypervisor mechanism provides an efficient way for utilizing physical servers by allowing for the deployment of virtual servers on physical servers.
- *Virtual Server* – The virtual server mechanism provides the most common type of resource used in cloud environments to host workloads of all types.
- *Cloud Storage Device* – The cloud storage device mechanism provides storage capabilities within the virtual private cloud.
- *Virtual Switch* – The virtual switch mechanism provides connectivity between virtual servers and the rest of the resources in the virtual private cloud.

14.14 CASE STUDY EXAMPLE

Innovartus is leasing two cloud-based environments from two different cloud providers, and intends to take advantage of this opportunity to establish a pilot cloud balancing architecture for its Role Player cloud service.

After assessing its requirements against the respective clouds, Innovartus's cloud architects produce a design specification that is based on each cloud having multiple implementations of the cloud service. This architecture incorporates separate automated scaling listener and failover system implementations, together with a central load balancer mechanism (Figure 14.33).

The load balancer distributes cloud service consumer requests across clouds using a workload distribution algorithm, while each cloud's automated scaling listener routes requests to local cloud service implementations. The failover systems can failover to the redundant cloud service implementations that are both within and across clouds. Intercloud failover is carried out primarily when local cloud service implementations are nearing their processing thresholds, or if a cloud is encountering a severe platform failure.

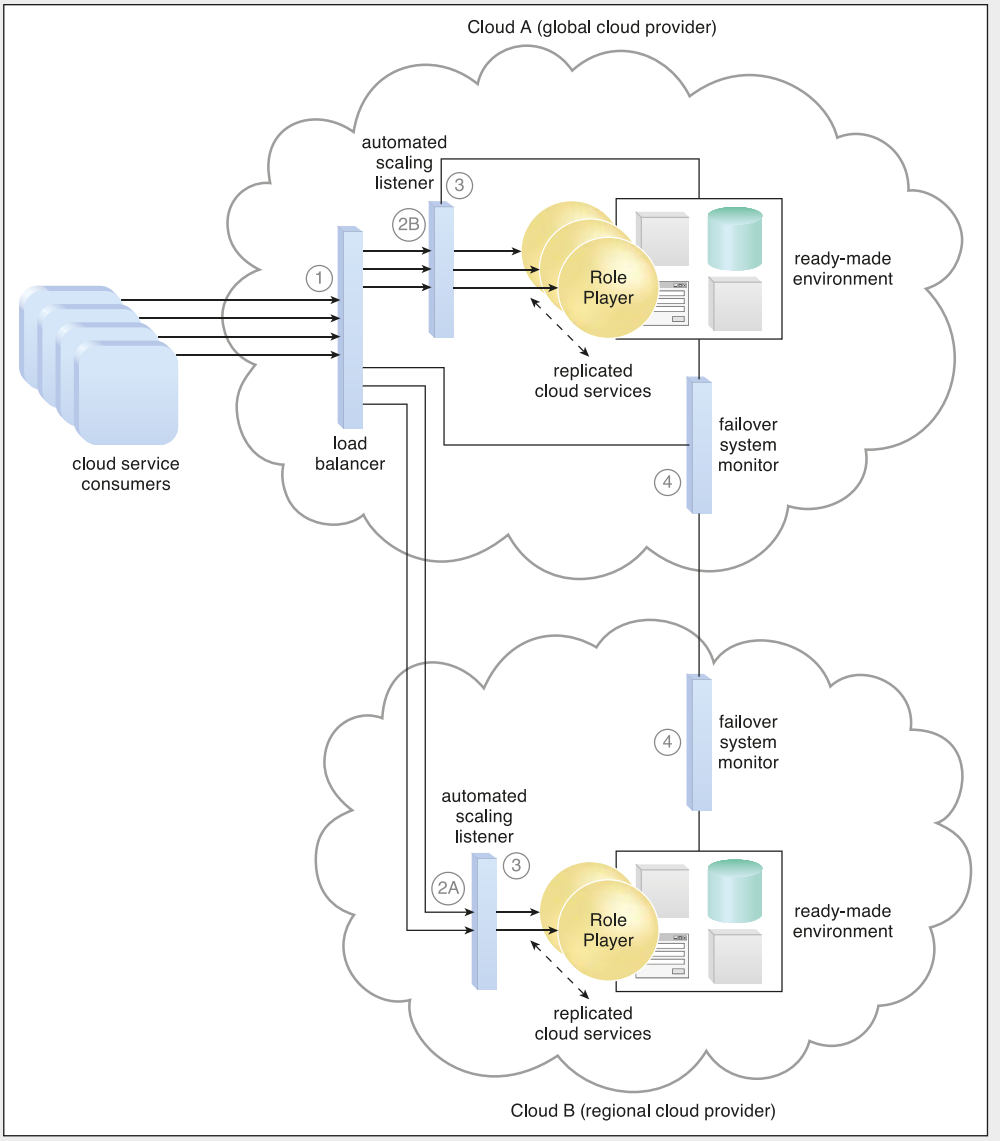


Figure 14.33

A load balancing service agent routes cloud service consumer requests according to a predefined algorithm (1). Requests are received by the local or external automated scaling listener (2A, 2B), which forwards each request to a cloud service implementation (3). Failover system monitors are used to detect and respond to cloud service failure (4).

Chapter 15

Specialized Cloud Architectures

The architectural models that are covered in this chapter span a broad range of functional areas and topics to offer creative combinations of mechanisms and specialized components.

The following architectures are covered:

- Direct I/O Access
- Direct LUN Access
- Dynamic Data Normalization
- Elastic Network Capacity
- Cross-Storage Device Vertical Tiering
- Intra-Storage Device Vertical Data Tiering
- Load-Balanced Virtual Switches
- Multipath Resource Access
- Persistent Virtual Network Configuration
- Redundant Physical Connection for Virtual Servers
- Storage Maintenance Window
- Edge Computing
- Fog Computing
- Virtual Data Abstraction
- Metacloud
- Federated Cloud Application

Where applicable, the involvement of related cloud mechanisms is described.

15.1 Direct I/O Access Architecture

Access to the physical I/O cards that are installed on a physical server is usually provided to hosted virtual servers via a hypervisor-based layer of processing called I/O virtualization. However, virtual servers sometimes need to connect to and use I/O cards without any hypervisor interaction or emulation.

With the *direct I/O access architecture*, virtual servers are allowed to circumvent the hypervisor and directly access the physical server's I/O card as an alternative to emulating a connection via the hypervisor (Figures 15.1 to 15.3).

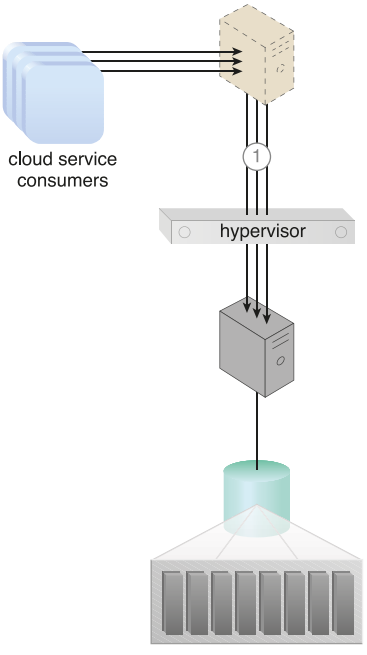


Figure 15.1
Cloud service consumers access a virtual server, which accesses a database on a SAN storage LUN (1). Connectivity from the virtual server to the database occurs via a virtual switch.

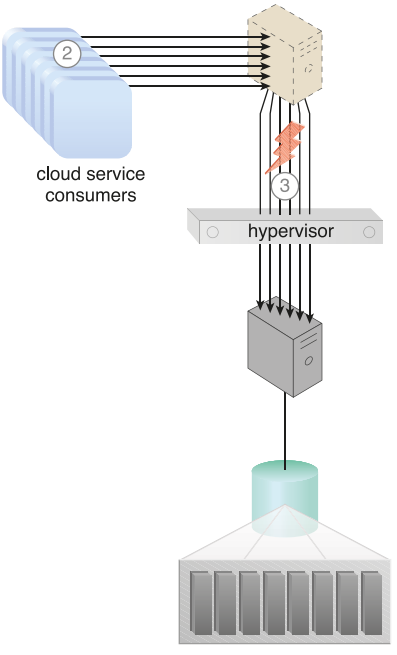


Figure 15.2
There is an increase in the amount of cloud service consumer requests (2), causing the bandwidth and performance of the virtual switch to become inadequate (3).

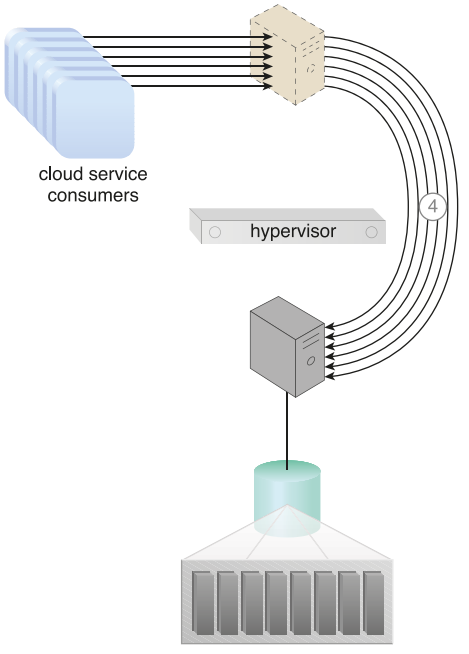


Figure 15.3
The virtual server bypasses the hypervisor to connect to the database server via a direct physical link to the physical server (4). The increased workload can now be properly handled.

To achieve this solution and access the physical I/O card without hypervisor interaction, the host CPU needs to support this type of access with the appropriate drivers installed on the virtual server. The virtual server can then recognize the I/O card as a hardware device after the drivers are installed.

Other mechanisms that can be involved in this architecture in addition to the virtual server and hypervisor include:

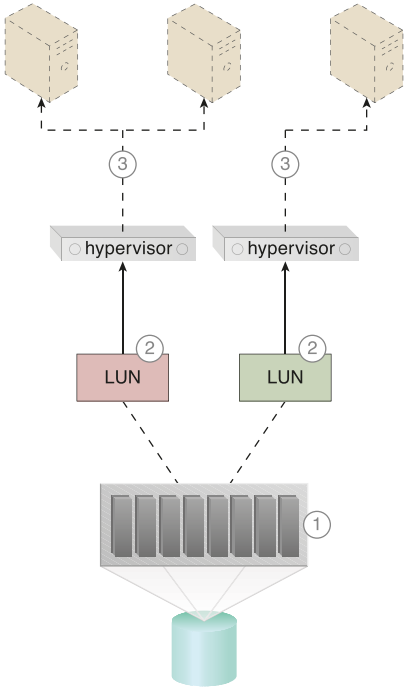
- *Cloud Usage Monitor* – The cloud service usage data that is collected by runtime monitors can include and separately classify direct I/O access.
- *Logical Network Perimeter* – The logical network perimeter ensures that the allocated physical I/O card does not allow cloud consumers to access other cloud consumers' IT resources.
- *Pay-Per-Use Monitor* – This monitor collects usage cost information for the allocated physical I/O card.
- *Resource Replication* – Replication technology is used to replace virtual I/O cards with physical I/O cards.

15.2 Direct LUN Access Architecture

Storage LUNs are typically mapped via a host bus adapter (HBA) on the hypervisor, with the storage space emulated as file-based storage to virtual servers (Figure 15.4). However, virtual servers sometimes need direct access to RAW block-based storage. For example, access via an emulated adapter is insufficient when a cluster is implemented and a LUN is used as the shared cluster storage device between two virtual servers.

Figure 15.4

The cloud storage device is installed and configured (1). The LUN mapping is defined so that each hypervisor has access to its own LUN and can also see all the mapped LUNs (2). The hypervisor shows the mapped LUNs to the virtual servers as normal file-based storage to be used as such (3).



The *direct LUN access architecture* provides virtual servers with LUN access via a physical HBA card, which is effective because virtual servers in the same cluster can use the LUN as a shared volume for clustered databases. After implementing this solution, the virtual servers' physical connectivity to the LUN and cloud storage device is enabled by the physical hosts.

The LUNs are created and configured on the cloud storage device for LUN presentation to the hypervisors. The cloud storage device needs to be configured using raw device mapping to make the LUNs visible to the virtual servers as a block-based RAW SAN LUN, which is unformatted, unpartitioned storage. The LUN needs to be represented with a unique LUN ID to be used by all the virtual servers as shared storage. Figures 15.5 and 15.6 illustrate how virtual servers are given direct access to block-based storage LUNs.

Figure 15.5

The cloud storage device is installed and configured (1). The required LUNs are created and presented to the hypervisors (2), which map the presented LUNs directly to the virtual servers (3). The virtual servers can see the LUNs as RAW block-based storage and can access them directly (4).

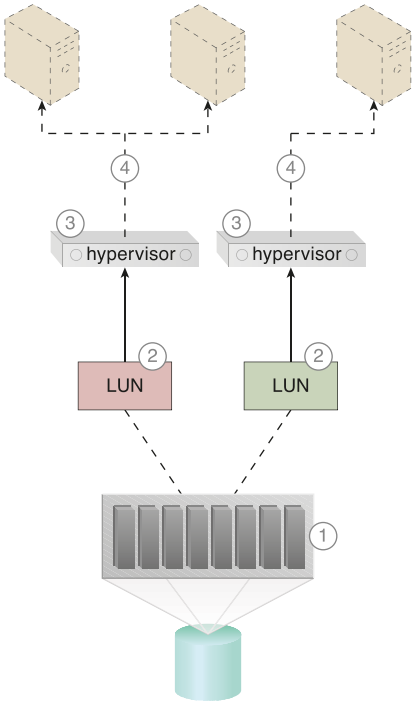
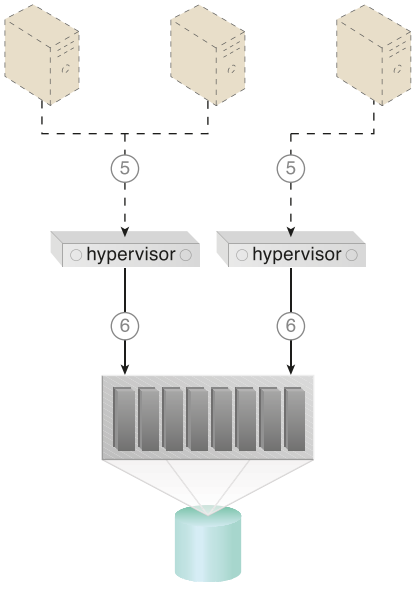


Figure 15.6

The virtual servers' storage commands are received by the hypervisors (5), which process and forward the requests to the storage processor (6).



Besides the virtual server, hypervisor, and cloud storage device, the following mechanisms can be incorporated into this architecture:

- *Cloud Usage Monitor* – This monitor tracks and collects storage usage information that pertains to the direct usage of LUNs.
- *Pay-Per-Use Monitor* – The pay-per-use monitor collects and separately classifies usage cost information for direct LUN access.
- *Resource Replication* – This mechanism relates to how virtual servers directly access block-based storage in replacement of file-based storage.

15.3 Dynamic Data Normalization Architecture

Redundant data can cause a range of issues in cloud-based environments, such as:

- increased time required to store and catalog files
- increased required storage and backup space
- increased costs due to increased data volume
- increased time required for replication to secondary storage
- increased time required to back up data

For example, if a cloud consumer copies 100 MB of files onto a cloud storage device and the data is redundantly copied ten times, the consequences can be considerable:

- The cloud consumer will be charged for using 10 × 100 MB of storage space, even though only 100 MB of unique data was actually stored.
- The cloud provider needs to provide an unnecessary 900 MB of space in the online cloud storage device and any backup storage systems.
- Significantly more time is required to store and catalog data.
- Data replication duration and performance are unnecessarily taxed whenever the cloud provider performs a site recovery, since 1,000 MB need to be replicated instead of 100 MB.

These impacts can be significantly amplified in multitenant public clouds.

The *dynamic data normalization architecture* establishes a de-duplication system, which prevents cloud consumers from inadvertently saving redundant copies of data by detecting and eliminating redundant data on cloud storage devices. This system can be applied to both block- and file-based storage devices, although it is most effective on the former. This de-duplication system checks each received block to determine whether it is redundant with a block that has already been received. Redundant blocks are replaced with pointers to the equivalent blocks that are already in storage (Figure 15.7).

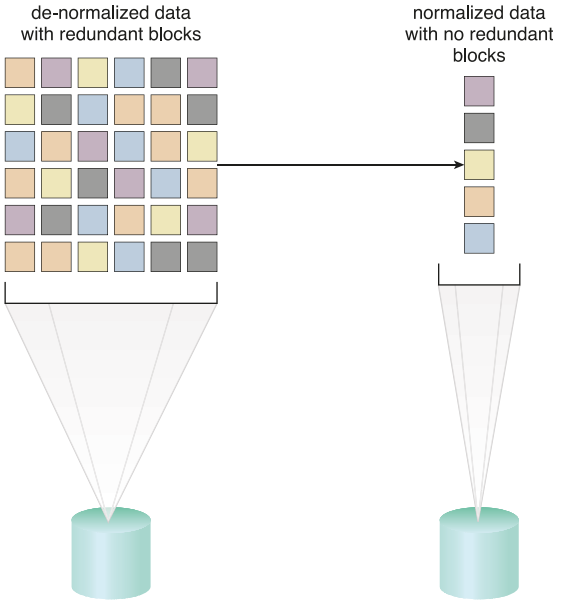


Figure 15.7
Datasets containing redundant data are unnecessarily bloating storage (left). The data de-duplication system normalizes the data so that only unique data is stored (right).

The de-duplication system examines received data prior to passing it to storage controllers. As part of the examination process, a hash code is assigned to every piece of data that has been processed and stored. An index of hashes and pieces is also maintained. As a result, the generated hash of a newly received block of data is compared with the hashes in storage to determine whether it is a new or duplicate data block. New blocks are saved, while duplicate data is eliminated, and a pointer to the original data block is created and saved instead.

This architectural model can be used for both disk storage and backup tape drives. One cloud provider can decide to prevent redundant data only on backup cloud storage devices, while another can more aggressively implement the data de-duplication system on all of its cloud storage devices. There are different methods and algorithms for comparing blocks of data to confirm their duplicity with other blocks.

15.4 Elastic Network Capacity Architecture

Even if IT resources are scaled on-demand by a cloud platform, performance and scalability can still be inhibited when remote access to the IT resources is impacted by network bandwidth limitations (Figure 15.8).

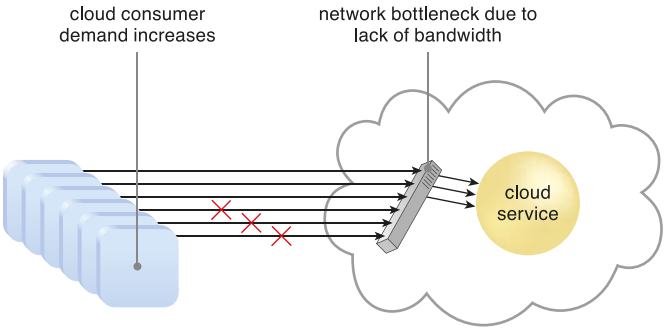


Figure 15.8
A lack of available bandwidth causes performance issues for cloud consumer requests.

The *elastic network capacity architecture* establishes a system in which additional bandwidth is allocated dynamically to the network to avoid runtime bottlenecks. This system ensures that each cloud consumer is using a different set of network ports to isolate individual cloud consumer traffic flows.

The automated scaling listener and intelligent automation engine scripts are used to detect when traffic reaches the bandwidth threshold and to dynamically allocate additional bandwidth and/or network ports when required.

The cloud architecture can be equipped with a network resource pool containing network ports that are made available for shared usage. The automated scaling listener monitors workload and network traffic and signals the intelligent automation engine to modify the number of allocated network ports and/or bandwidth in response to usage fluctuations.

Note that when this architectural model is implemented at the virtual switch level, the intelligent automation engine may need to run a separate script that adds physical uplinks to the virtual switch specifically. Alternatively, the direct I/O access architecture can also be incorporated to increase network bandwidth that is allocated to the virtual server.

In addition to the automated scaling listener, the following mechanisms can be part of this architecture:

- *Cloud Usage Monitor* – This monitor is responsible for tracking elastic network capacity before, during, and after scaling.

- *Hypervisor* – The hypervisor provides virtual servers with access to the physical network, via virtual switches and physical uplinks.
- *Logical Network Perimeter* – This mechanism establishes the boundaries that are needed to provide individual cloud consumers with their allocated network capacity.
- *Pay-Per-Use Monitor* – This monitor keeps track of any billing-related data that pertains to dynamic network bandwidth consumption.
- *Resource Replication* – Resource replication is used to add network ports to physical and virtual servers in response to workload demands.
- *Virtual Server* – Virtual servers host the IT resources and cloud services to which network resources are allocated and are themselves affected by the scaling of network capacity.

15.5 Cross-Storage Device Vertical Tiering Architecture

Cloud storage devices are sometimes unable to accommodate the performance requirements of cloud consumers and have more data processing power or bandwidth added to increase input/output operations per second (IOPS). These conventional methods of vertical scaling are usually inefficient and time-consuming to implement and can become wasteful when the increased capacity is no longer required.

The scenario in Figures 15.9 and 15.10 depicts an approach in which the number of requests for access to a LUN has increased, requiring its manual transfer to a high-performance cloud storage device.

The *cross-storage device vertical tiering architecture* establishes a system that survives bandwidth and data processing power constraints by vertically scaling between storage devices that have different capacities. LUNs can automatically scale up and down across multiple devices in this system so that requests can use the appropriate storage device level to perform cloud consumer tasks.

New cloud storage devices with increased capacity can also be made available, even if the automated tiering technology can move data to cloud storage devices with the same storage processing capacity. For example, solid-state drives (SSDs) can be suitable devices for data processing power upgrades.

Figure 15.9

A cloud provider installs and configures a cloud storage device (1) and creates LUNs that are made available to the cloud service consumers for usage (2). The cloud service consumers initiate data access requests to the cloud storage device (3), which forwards the requests to one of the LUNs (4).

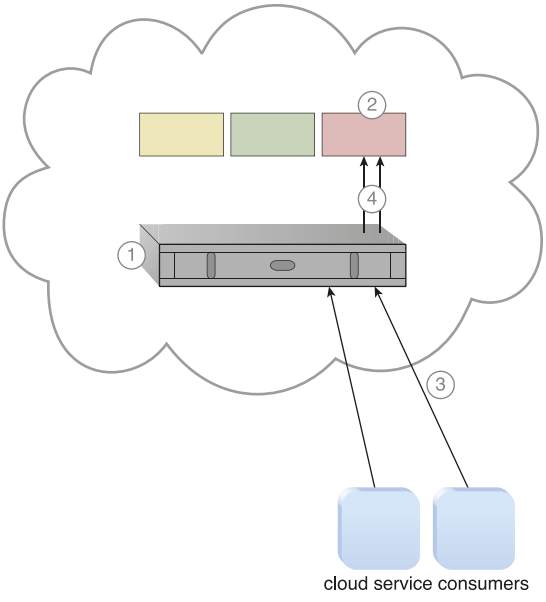
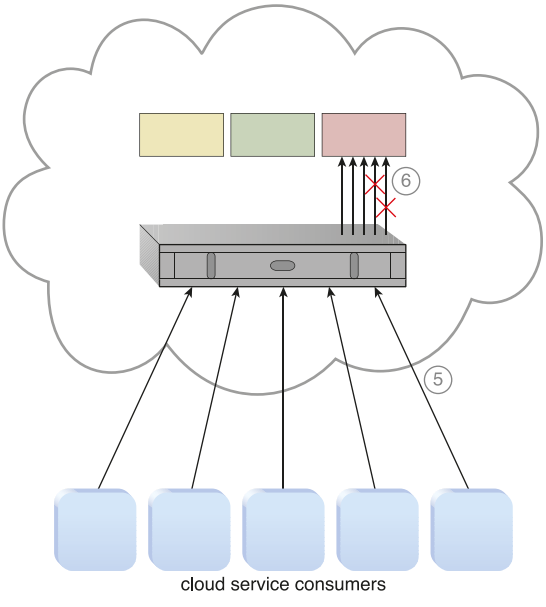


Figure 15.10

The number of requests increases, resulting in high storage bandwidth and performance demands (5). Some of the requests are rejected or time out due to performance capacity limitations within the cloud storage device (6).



The automated scaling listener monitors the requests that are sent to specific LUNs and signals the storage management program to move the LUN to a higher-capacity device once it identifies a predefined threshold has been reached. Service interruption is prevented because there is never a disconnection during the transfer. The original device remains up and running, while the LUN data scales up to another device. Cloud consumer requests are automatically redirected to the new cloud storage device as soon as the scaling is completed (Figures 15.11 to 15.13).

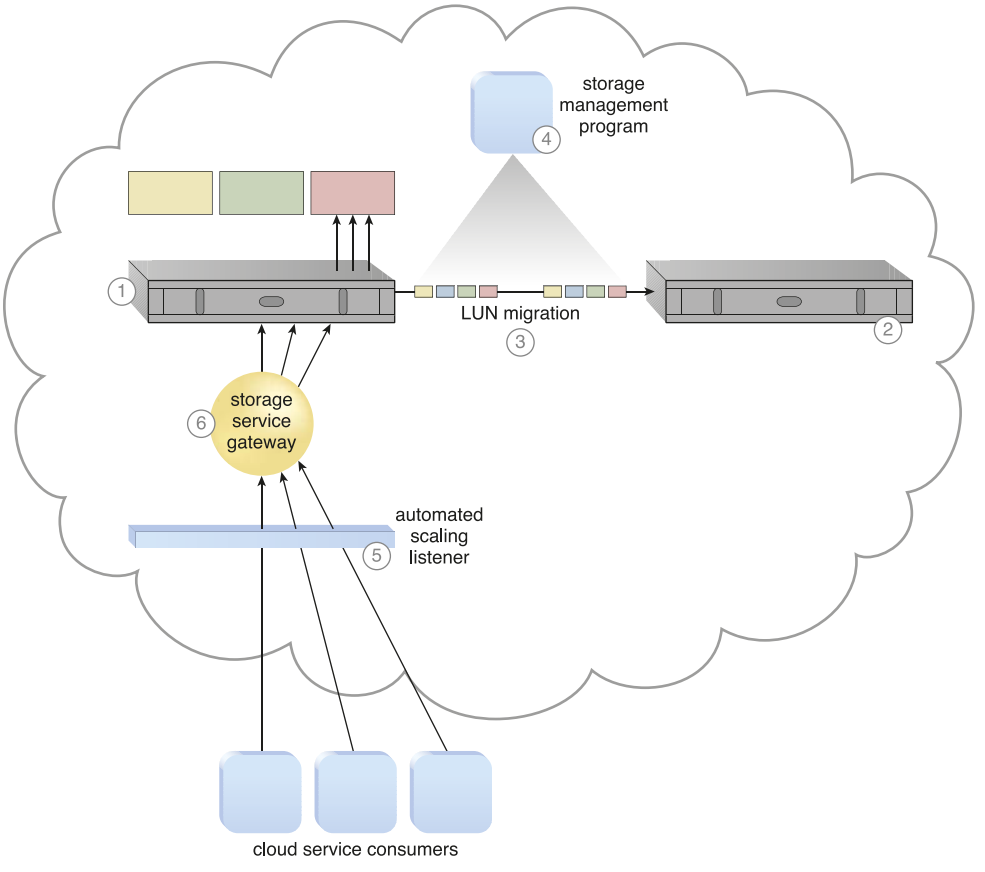


Figure 15.11

The lower-capacity primary cloud storage device is responding to cloud service consumer storage requests (1). A secondary cloud storage device with higher capacity and performance is installed (2). The LUN migration (3) is configured via the storage management program that is configured to categorize the storage based on device performance (4). Thresholds are defined in the automated scaling listener that is monitoring the requests (5). Cloud service consumer requests are received by the storage service gateway and sent to the primary cloud storage device (6).

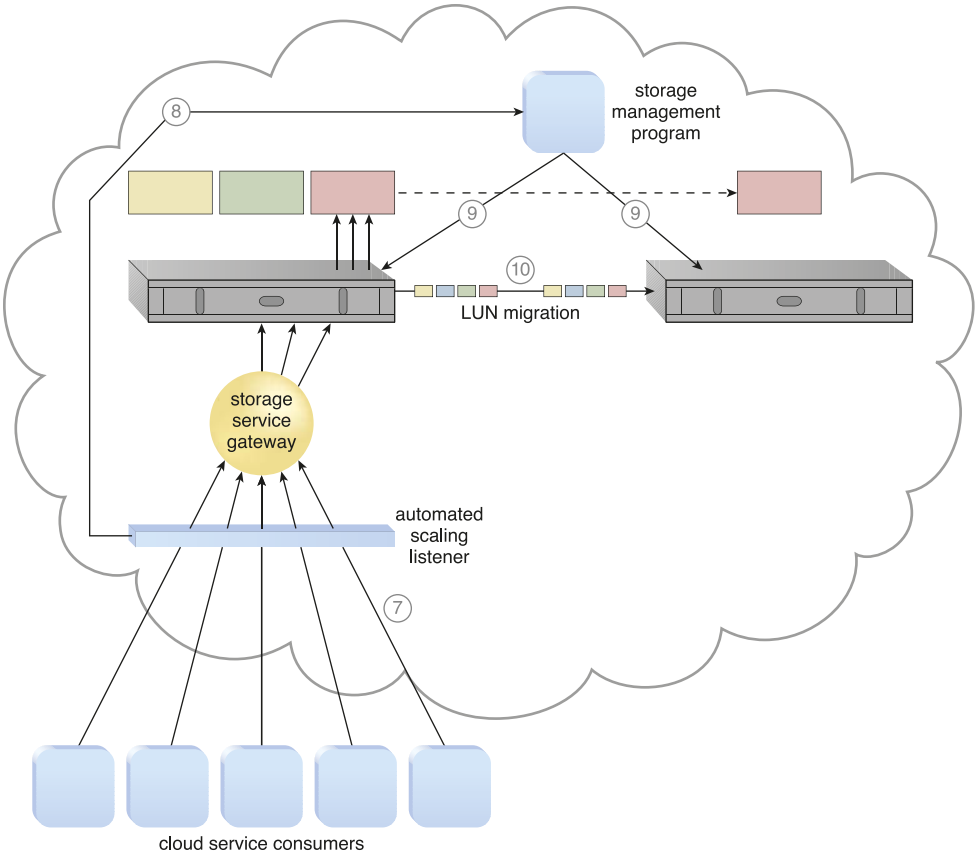


Figure 15.12

The number of cloud service consumer requests reaches the predefined threshold (7), and the automated scaling listener notifies the storage management program that scaling is required (8). The storage management program calls LUN migration to move the cloud consumer's LUN to the secondary, higher-capacity storage device (9), and the LUN migration performs this move (10).

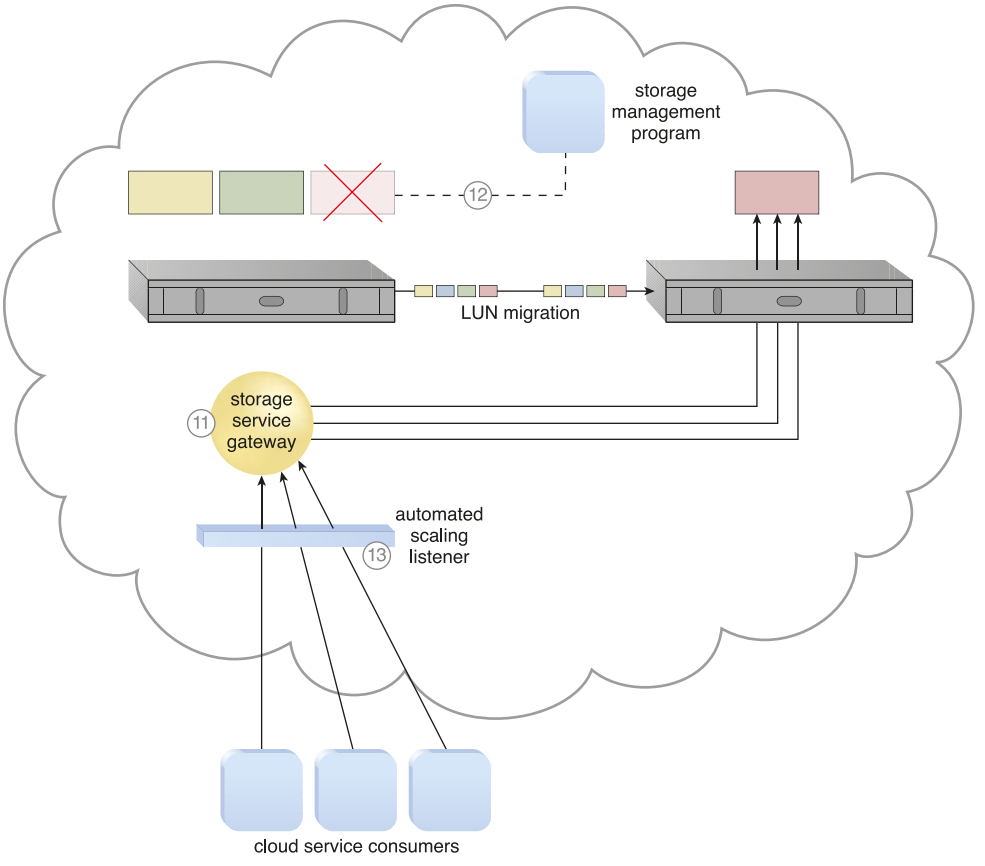


Figure 15.13

The storage service gateway forwards the cloud service consumer requests from the LUN to the new cloud storage device (11). The original LUN is deleted from the lower-capacity device via the storage management program and LUN migration (12). The automated scaling listener monitors cloud service consumer requests to ensure that the request volume continues to require access to the higher-capacity secondary storage for the migrated LUN (13).

In addition to the automated scaling listener and cloud storage device, the mechanisms that can be incorporated in this technology architecture include:

- *Audit Monitor* – The auditing performed by this monitor checks whether the relocation of cloud consumer data does not conflict with any legal or data privacy regulations or policies.

- *Cloud Usage Monitor* – This infrastructure mechanism represents various runtime monitoring requirements for tracking and recording data transfer and usage at both source and destination storage locations.
- *Pay-Per-Use Monitor* – Within the context of this architecture, the pay-per-use monitor collects storage usage information on both source and destination locations, as well as IT resource usage information for carrying out cross-storage tiering functionality.

15.6 Intra-Storage Device Vertical Data Tiering Architecture

Some cloud consumers may have distinct data storage requirements that restrict the data's physical location to a single cloud storage device. Distribution across other cloud storage devices may be disallowed due to security, privacy, or various legal reasons. This type of limitation can impose severe scalability limitations upon the device's storage and performance capacity. These limitations can further cascade to any cloud services or applications that are dependent upon the use of the cloud storage device.

The *intra-storage device vertical data tiering architecture* establishes a system to support vertical scaling within a single cloud storage device. This intra-device scaling system optimizes the availability of different disk types with different capacities (Figure 15.14).

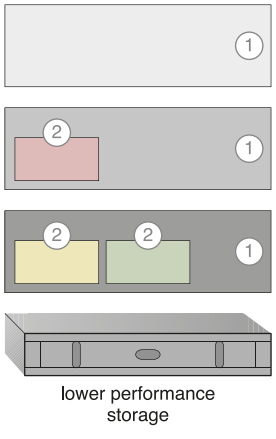


Figure 15.14
The cloud intra-storage device system vertically scales through disk types graded into different tiers (1). Each LUN is moved to a tier that corresponds to its processing and storage requirements (2).

This cloud storage architecture requires the use of a complex storage device that supports different types of hard disks, especially high-performance disks like SATAs, SASs, and SSDs. The disk types are organized into graded tiers so that LUN migration can vertically scale the device based on the allocation of disk types, which align with the processing and capacity requirements.

Data load conditions and definitions are set after disk categorization so that the LUNs can move to either a higher or lower grade, depending on which predefined conditions are met. These thresholds and conditions are used by the automated scaling listener when monitoring runtime data processing traffic (Figures 15.15 to 15.17).

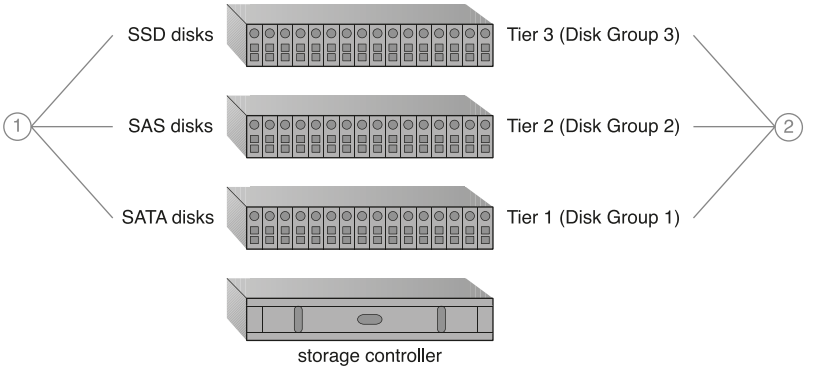


Figure 15.15

Different types of hard disks are installed in the enclosures of a cloud storage device (1). Similar disk types are grouped into tiers to create different grades of disk groups based on I/O performance (2).

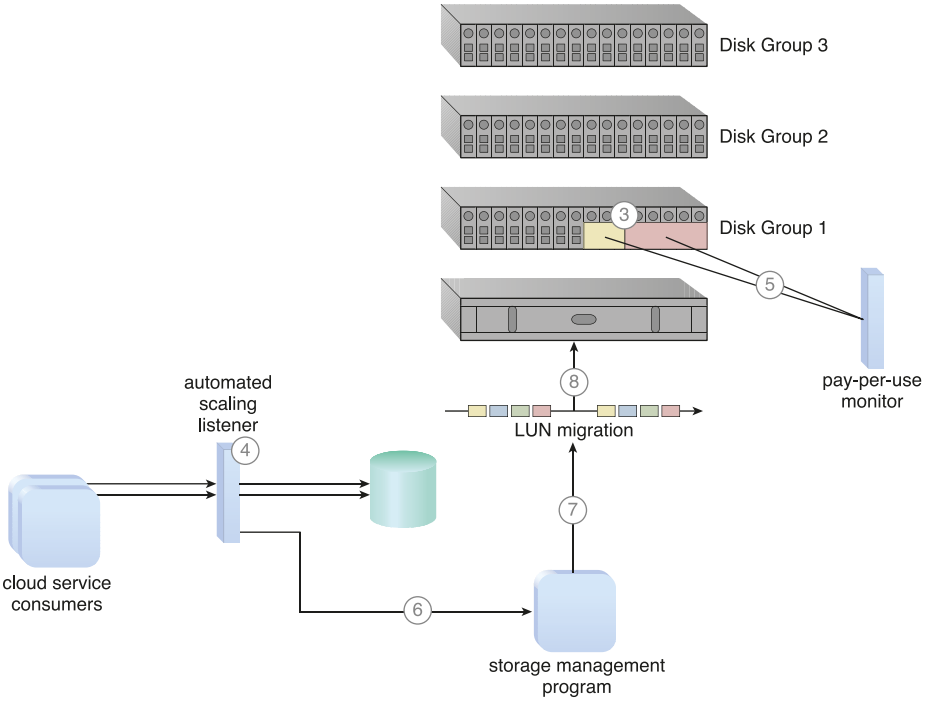
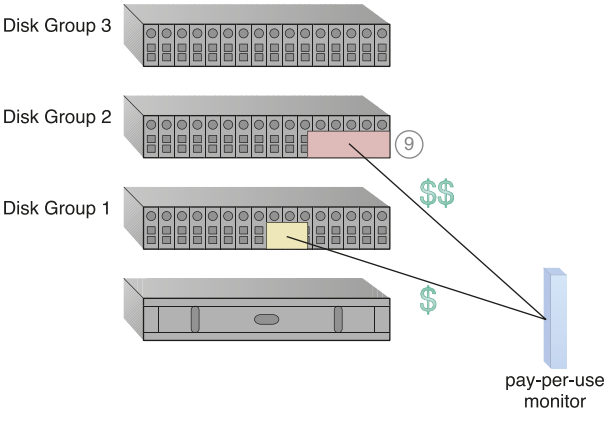


Figure 15.16

Two LUNs have been created on Disk Group 1 (3). The automated scaling listener monitors the requests in relation to predefined thresholds (4). The pay-per-use monitor tracks the actual amount of disk usage, based on free space and disk group performance (5). The automated scaling listener determines that the number of requests is reaching a threshold and informs the storage management program that the LUN needs to be moved to a higher-performance disk group (6). The storage management program signals the LUN migration program to perform the required move (7). The LUN migration program works with the storage controller to move the LUN to the higher-capacity Disk Group 2 (8).

Figure 15.17

The usage price of the migrated LUN in Disk Group 2 is now higher than before, because a higher-performance disk group is being used (9).



15.7 Load-Balanced Virtual Switches Architecture

Virtual servers are connected to the outside world via virtual switches, which send and receive traffic with the same uplink. Bandwidth bottlenecks form when the network traffic on the uplink's port increases to a point that it causes transmission delays, performance issues, packet loss, and lag time (Figures 15.18 and 15.19).

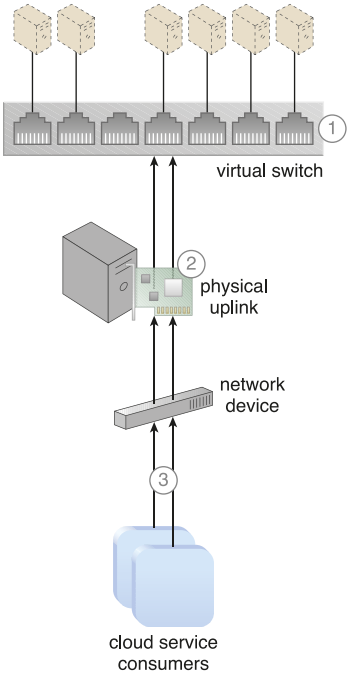


Figure 15.18

A virtual switch is interconnecting virtual servers (1). A physical network adapter has been attached to the virtual switch to be used as an uplink to the physical (external) network, connecting the virtual servers to cloud consumers (2). Cloud service consumers send requests via the physical uplink (3).

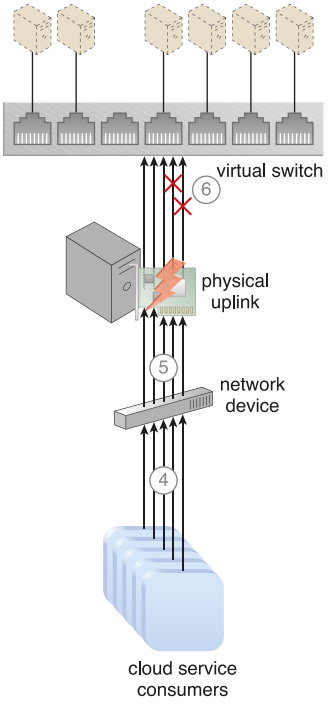


Figure 15.19

The amount of traffic passing through the physical uplink grows in parallel with the increasing number of requests. The number of packets that need to be processed and forwarded by the physical network adapter also increases (4). The physical adapter cannot handle the workload, now that the network traffic has exceeded its capacity (5). The network forms a bottleneck that results in performance degradation and the loss of delay-sensitive data packets (6).

The *load-balanced virtual switches architecture* establishes a load balancing system wherein multiple uplinks are provided to balance network traffic workloads across multiple uplinks or redundant paths, which can help avoid slow transfers and data loss (Figure 15.20). Link aggregation can be executed to balance the traffic, which allows the workload to be distributed across multiple uplinks at the same time so that none of the network cards is overloaded.

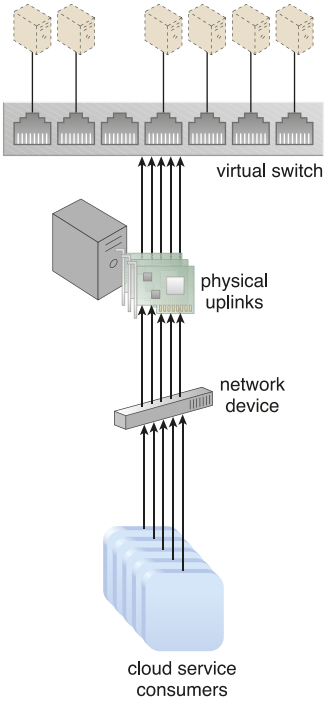


Figure 15.20
Additional physical uplinks are added to distribute and balance network traffic.

The virtual switch needs to be configured to support multiple physical uplinks, which are usually configured as an NIC team that has defined traffic-shaping policies.

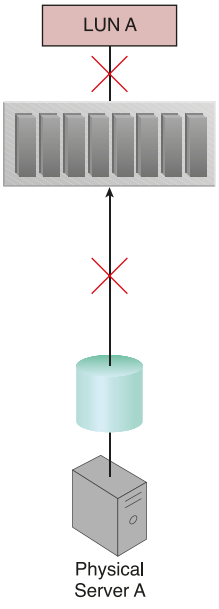
The following mechanisms can be incorporated into this architecture:

- *Cloud Usage Monitor* – These monitors are used to monitor network traffic and bandwidth usage.
- *Hypervisor* – This mechanism hosts and provides the virtual servers with access to both the virtual switches and external network.
- *Load Balancer* – The load balancer distributes the network workload across the different uplinks.
- *Logical Network Perimeter* – The logical network perimeter creates boundaries that protect and limit the bandwidth usage for each cloud consumer.
- *Resource Replication* – This mechanism is used to generate additional uplinks to the virtual switch.
- *Virtual Server* – Virtual servers host the IT resources that benefit from the additional uplinks and bandwidth via virtual switches.

15.8 Multipath Resource Access Architecture

Certain IT resources can only be accessed using an assigned path (or hyperlink) that leads to their exact location. This path can be lost or incorrectly defined by the cloud consumer or changed by the cloud provider. An IT resource whose hyperlink is no longer in the possession of the cloud consumer becomes inaccessible and unavailable (Figure 15.21). Exception conditions that result from IT resource unavailability can compromise the stability of larger cloud solutions that depend on the IT resource.

Figure 15.21
Physical Server A is connected to LUN A via a single fiber channel, and uses the LUN to store different types of data. The fiber channel connection becomes unavailable due to a HBA card failure and invalidates the path used by Physical Server A, which has now lost access to LUN A and all of its stored data.



The *multipath resource access architecture* establishes a multipathing system with alternative paths to IT resources, so that cloud consumers have the means to programmatically or manually overcome path failures (Figure 15.22).

This technology architecture requires the use of a multipathing system and the creation of alternative physical or virtual hyperlinks that are assigned to specific IT resources. The multipathing system resides on the server or hypervisor and ensures that each IT resource can be seen via each alternative path identically (Figure 15.23).

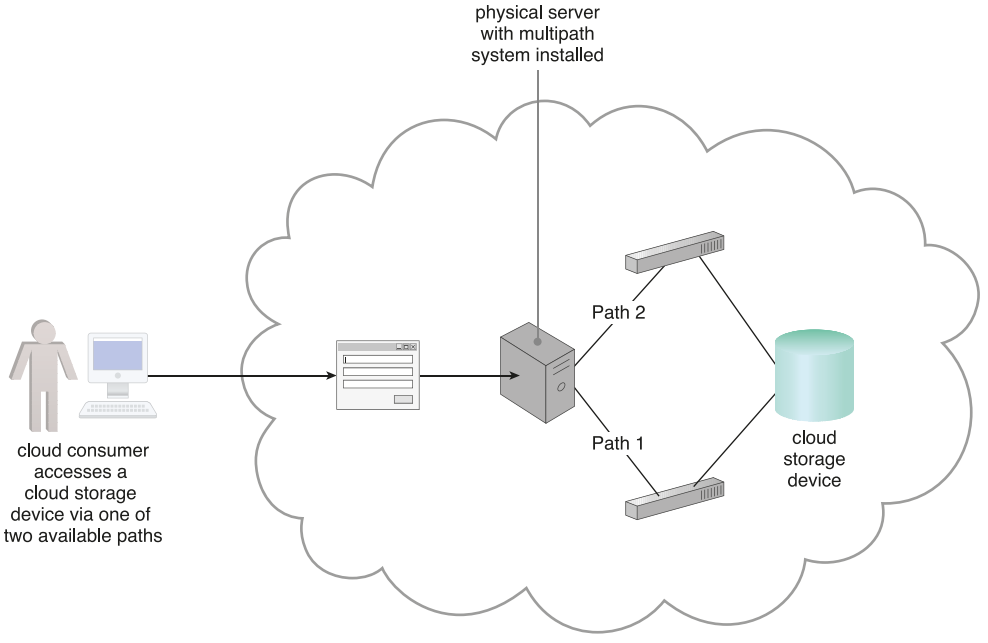


Figure 15.22
A multipathing system is providing alternative paths to a cloud storage device.

This architecture can involve the following mechanisms:

- *Cloud Storage Device* – The cloud storage device is a common IT resource that requires the creation of alternative paths in order to remain accessible to solutions that rely on data access.
- *Hypervisor* – Alternative paths to a hypervisor are required in order to have redundant links to the hosted virtual servers.
- *Logical Network Perimeter* – This mechanism guarantees the maintenance of cloud consumer privacy, even when multiple paths to the same IT resource are created.
- *Resource Replication* – The resource replication mechanism is required when a new instance of an IT resource needs to be created to generate the alternative path.
- *Virtual Server* – These servers host the IT resources that have multipath access via different links or virtual switches. Hypervisors can provide multipath access to the virtual servers.

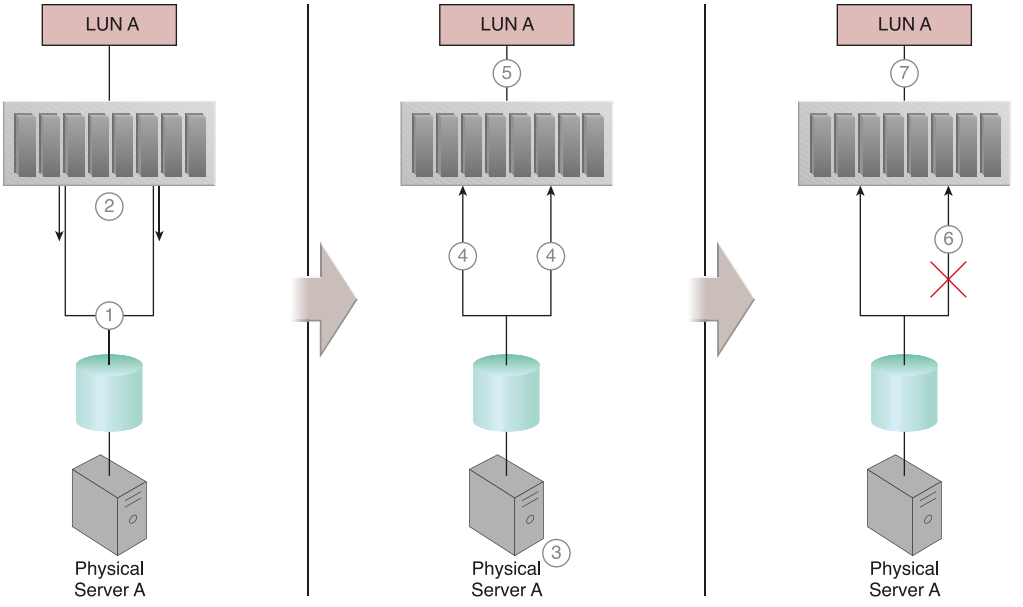


Figure 15.23

Physical Server A is connected to the LUN A cloud storage device via two different paths (1). LUN A is seen as different LUNs from each of the two paths (2). The multipathing system is configured (3). LUN A is seen as one identical LUN from both paths (4), and Physical Server A has access to LUN A from two different paths (5). A link failure occurs and one of the paths becomes unavailable (6). Physical Server A can still use LUN A because the other link remains active (7).

15.9 Persistent Virtual Network Configuration Architecture

Network configurations and port assignments for virtual servers are generated during the creation of the virtual switch on the host physical server and the hypervisor hosting the virtual server. These configurations and assignments reside in the virtual server's immediate hosting environment, meaning a virtual server that is moved or migrated to another host will lose network connectivity because destination hosting environments do not have the required port assignments and network configuration information (Figure 15.24).

In the *persistent virtual network configuration architecture*, network configuration information is stored in a centralized location and replicated to physical server hosts. This allows the destination host to access the configuration information when a virtual server is moved from one host to another.

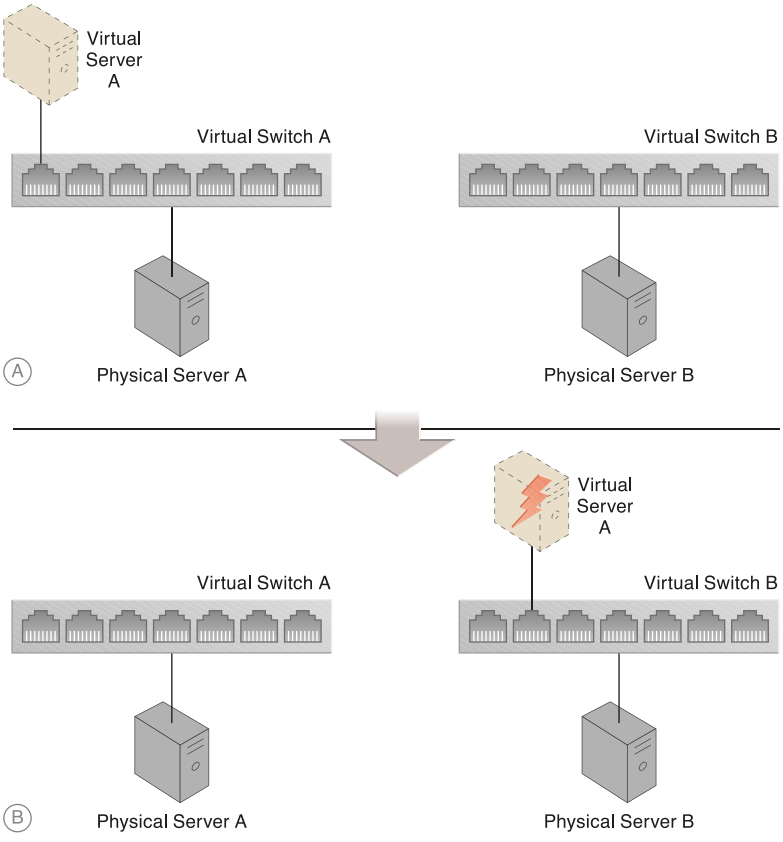


Figure 15.24

Part A shows Virtual Server A connected to the network through Virtual Switch A, which was created on Physical Server A. In Part B, Virtual Server A is connected to Virtual Switch B after being moved to Physical Server B. The virtual server cannot connect to the network because its configuration settings are missing.

The system established with this architecture includes a centralized virtual switch, VIM, and configuration replication technology. The centralized virtual switch is shared by physical servers and configured via the VIM, which initiates replication of the configuration settings to the physical servers (Figure 15.25).

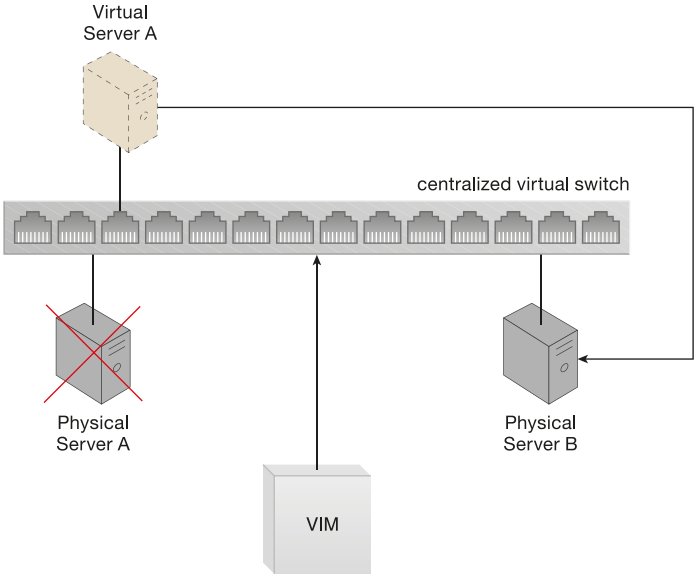


Figure 15.25

A virtual switch's configuration settings are maintained by the VIM, which ensures that these settings are replicated to other physical servers. The centralized virtual switch is published, and each host physical server is assigned some of its ports. Virtual Server A is moved to Physical Server B when Physical Server A fails. The virtual server's network settings are retrievable, since they are stored on a centralized virtual switch that is shared by both physical servers. Virtual Server A maintains network connectivity on its new host, Physical Server B.

In addition to the virtual server mechanism for which this architecture provides a migration system, the following mechanisms can be included:

- *Hypervisor* – The hypervisor hosts the virtual servers that require the configuration settings to be replicated across the physical hosts.
- *Logical Network Perimeter* – The logical network perimeter helps ensure that access to the virtual server and its IT resources is isolated to the rightful cloud consumer, before and after a virtual server is migrated.
- *Resource Replication* – The resource replication mechanism is used to replicate the virtual switch configurations and network capacity allocations across the hypervisors, via the centralized virtual switch.

15.10 Redundant Physical Connection for Virtual Servers Architecture

A virtual server is connected to an external network via a virtual switch uplink port, meaning the virtual server will become isolated and disconnected from the external network if the uplink fails (Figure 15.26).

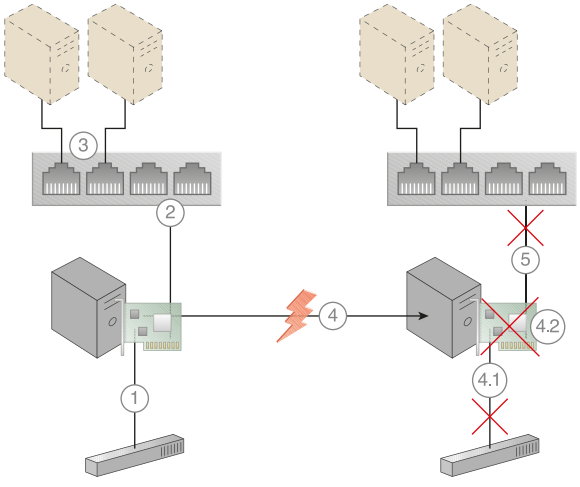


Figure 15.26
A physical network adapter installed on the host physical server is connected to the physical switch on the network (1). A virtual switch is created for use by two virtual servers. The physical network adapter is attached to the virtual switch to act as an uplink, since it requires access to the physical (external) network (2). The virtual servers communicate with the external network via the attached physical uplink network card (3). A connection failure occurs, either because of a physical link connectivity issue between the physical adapter and the physical switch (4.1), or because of a physical network card failure (4.2). The virtual servers lose access to the physical external network and are no longer accessible to their cloud consumers (5).

The *redundant physical connection for virtual servers architecture* establishes one or more redundant uplink connections and positions them in standby mode. This architecture ensures that a redundant uplink connection is available to connect the active uplink, whenever the primary uplink connection becomes unavailable (Figure 15.27).

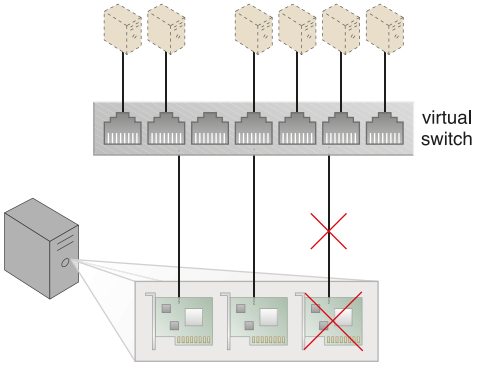


Figure 15.27
Redundant uplinks are installed on a physical server that is hosting several virtual servers. When an uplink fails, another uplink takes over to maintain the virtual servers' active network connections.

In a process that is transparent to both virtual servers and their users, a standby uplink automatically becomes the active uplink as soon as the main uplink fails, and the virtual servers use the newly active uplink to sends packets externally.

The second NIC does not forward any traffic while the primary uplink is alive, even though it receives the virtual server's packets. However, the secondary uplink will start forwarding packets immediately if the primary uplink were to fail (Figures 15.28 to 15.30). The failed uplink becomes the primary uplink again after it returns to operation, while the second NIC returns to standby mode.

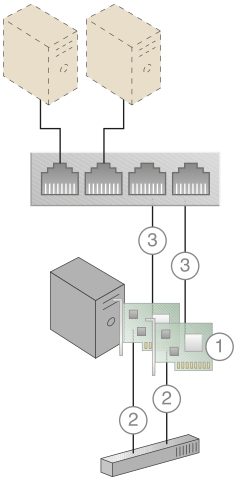


Figure 15.28
A new network adapter is added to support a redundant uplink (1). Both network cards are connected to the physical external switch (2), and both physical network adapters are configured to be used as uplink adapters for the virtual switch (3).

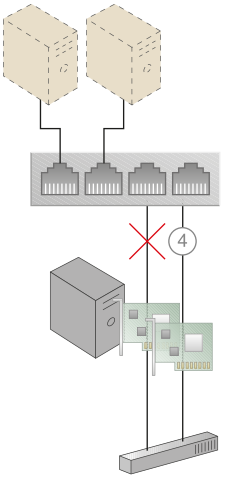


Figure 15.29
One physical network adapter is designated as the primary adapter (4), while the other is designated as the secondary adapter providing the standby uplink. The secondary adapter does not forward any packets.

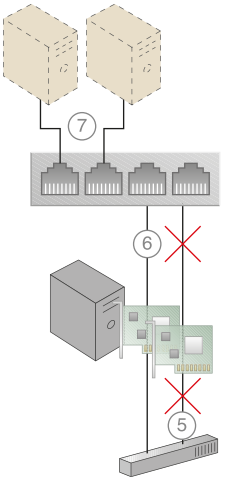


Figure 15.30
The primary uplink becomes unavailable (5). The secondary standby uplink automatically takes over and uses the virtual switch to forward the virtual servers' packets to the external network (6). The virtual servers do not experience interruptions and remain connected to the external network (7).

The following mechanisms are commonly part of this architecture, in addition to the virtual server:

- *Failover System* – The failover system performs the transition of unavailable uplinks to standby uplinks.
- *Hypervisor* – This mechanism hosts virtual servers and some virtual switches, and provides virtual networks and virtual switches with access to the virtual servers.
- *Logical Network Perimeter* – Logical network perimeters ensure that the virtual switches that are allocated or defined for each cloud consumer remain isolated.
- *Resource Replication* – Resource replication is used to replicate the current status of active uplinks to standby uplinks so as to maintain the network connection.

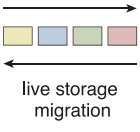
15.11 Storage Maintenance Window Architecture

Cloud storage devices that are subject to maintenance and administrative tasks sometimes need to be temporarily shut down, meaning cloud service consumers and IT resources consequently lose access to these devices and their stored data (Figure 15.31).

The data of a cloud storage device that is about to undergo a maintenance outage can be temporarily moved to a secondary duplicate cloud storage device. The *storage maintenance window architecture* enables cloud service consumers to be automatically and transparently redirected to the secondary cloud storage device, without becoming aware that their primary storage device has been taken offline.

LIVE STORAGE MIGRATION

The live storage migration program is a sophisticated system that utilizes the LUN migration component to reliably move LUNs by enabling the original copy to remain active until after the destination copy has been verified as being fully functional.



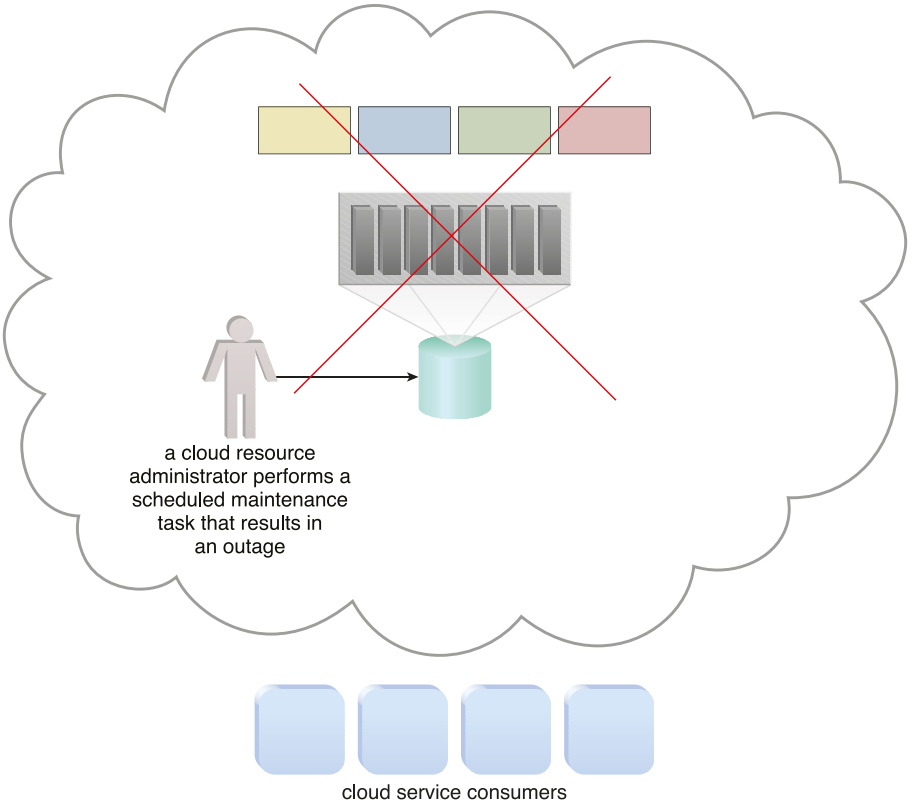


Figure 15.31

A prescheduled maintenance task carried out by a cloud resource administrator causes an outage for the cloud storage device, which becomes unavailable to cloud service consumers. Because cloud consumers were previously notified of the outage, cloud consumers do not attempt any data access.

This architecture uses a live storage migration program, as demonstrated in Figures 15.32 to 15.37.

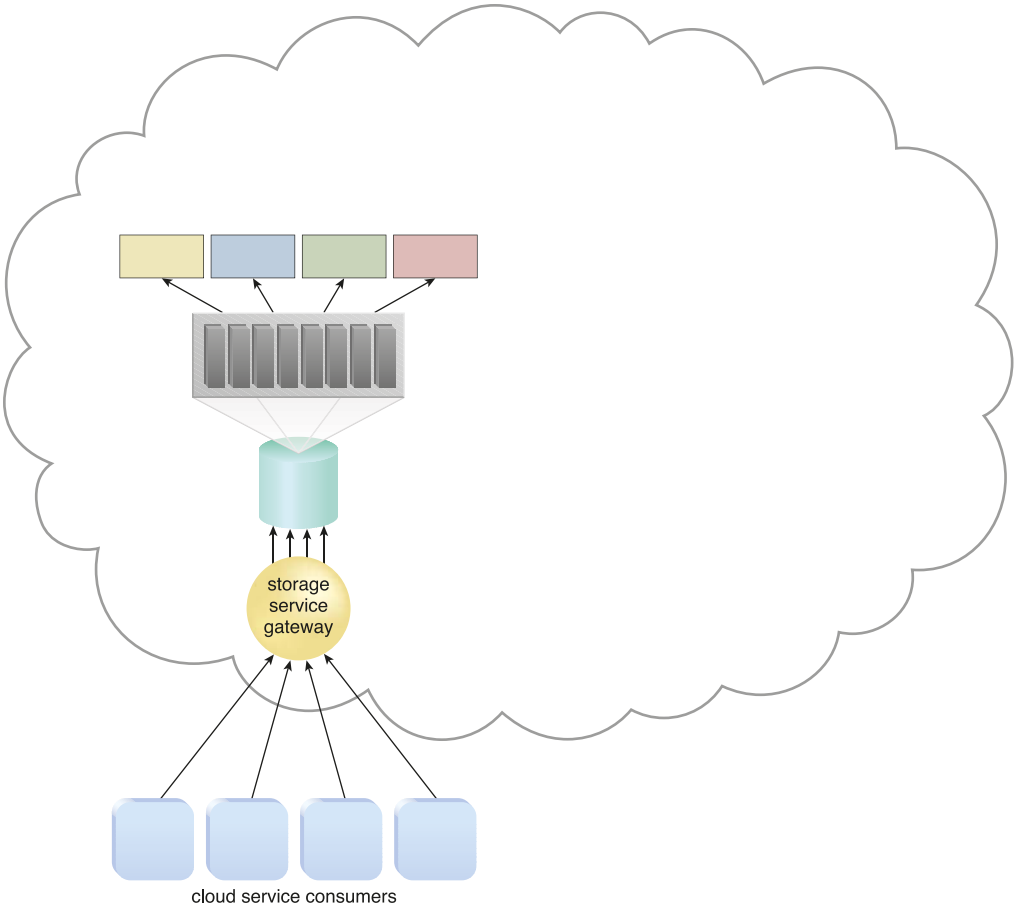


Figure 15.32

The cloud storage device is scheduled to undergo a maintenance outage, but unlike the scenario depicted in Figure 15.31, the cloud service consumers were not notified of the outage and continue to access the cloud storage device.

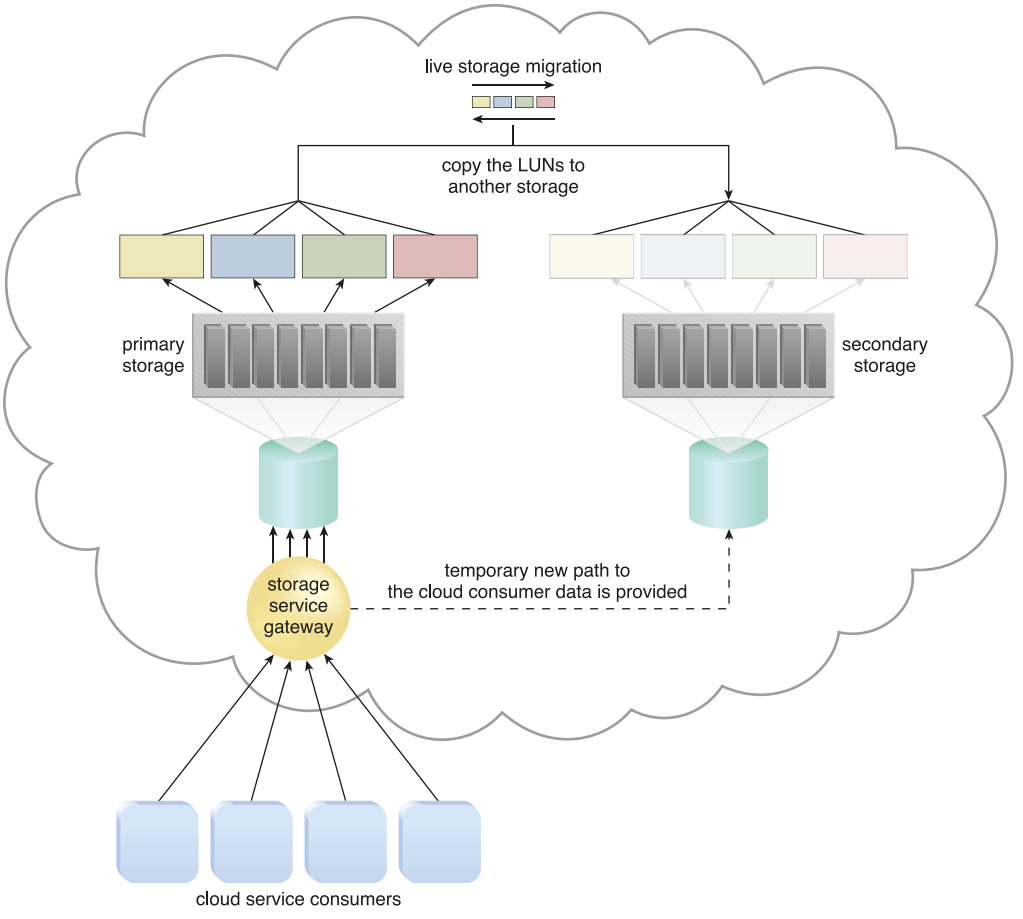


Figure 15.33

Live storage migration moves the LUNs from the primary storage device to a secondary storage device.

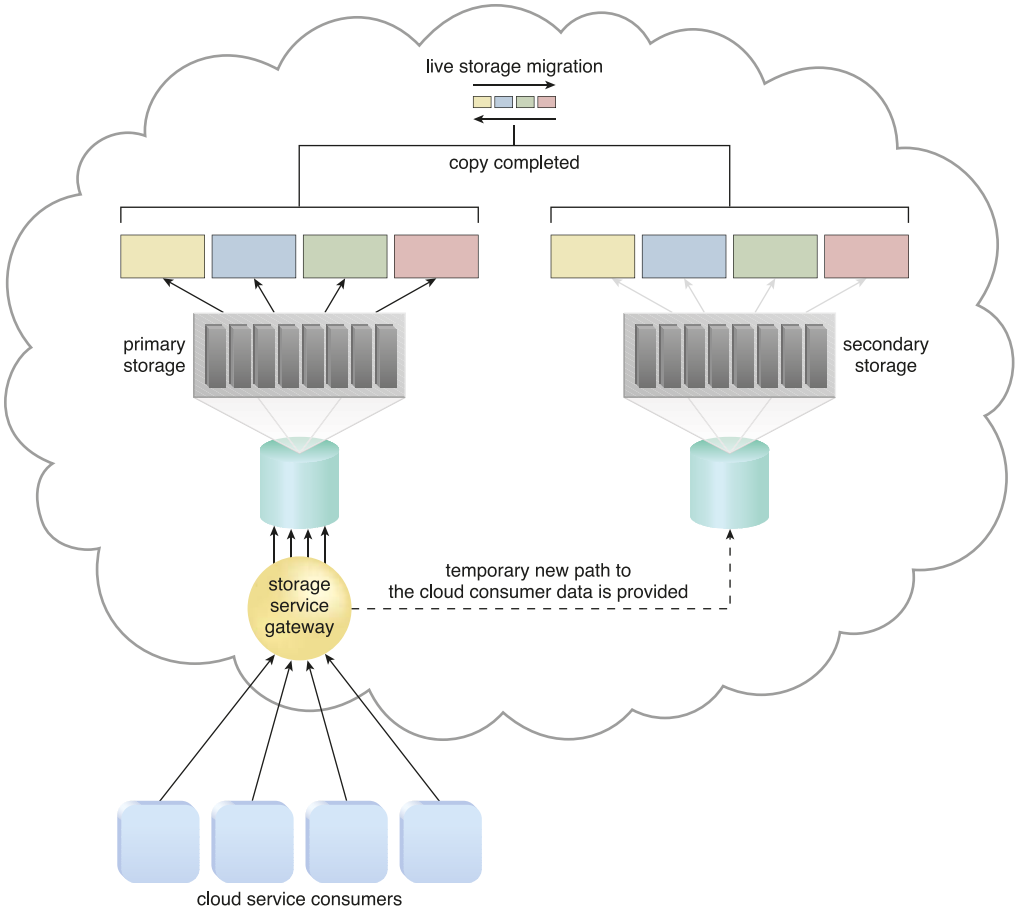


Figure 15.34

Requests for the data are forwarded to the duplicate LUNs on the secondary storage device once the LUN's data has been migrated.

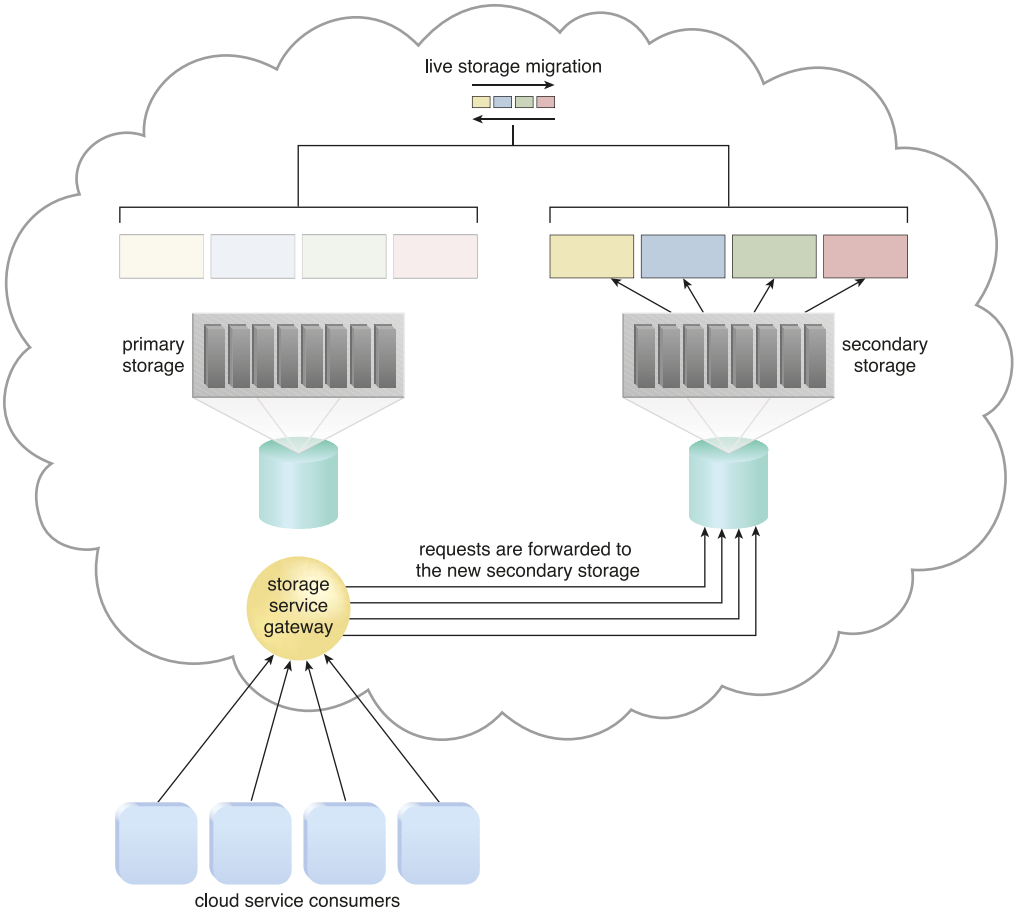


Figure 15.35
The primary storage is powered off for maintenance.

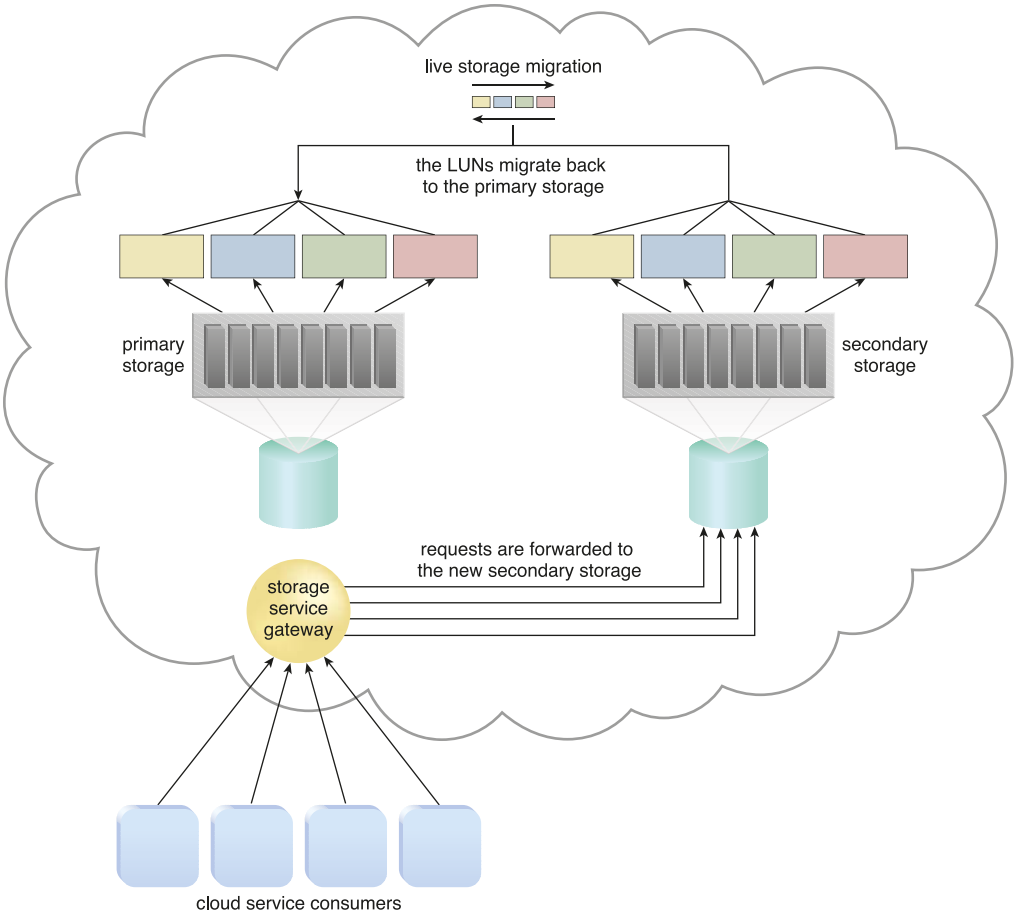


Figure 15.36

The primary storage is brought back online after the maintenance task is finished. Live storage migration restores the LUN data from the secondary storage device to the primary storage device.

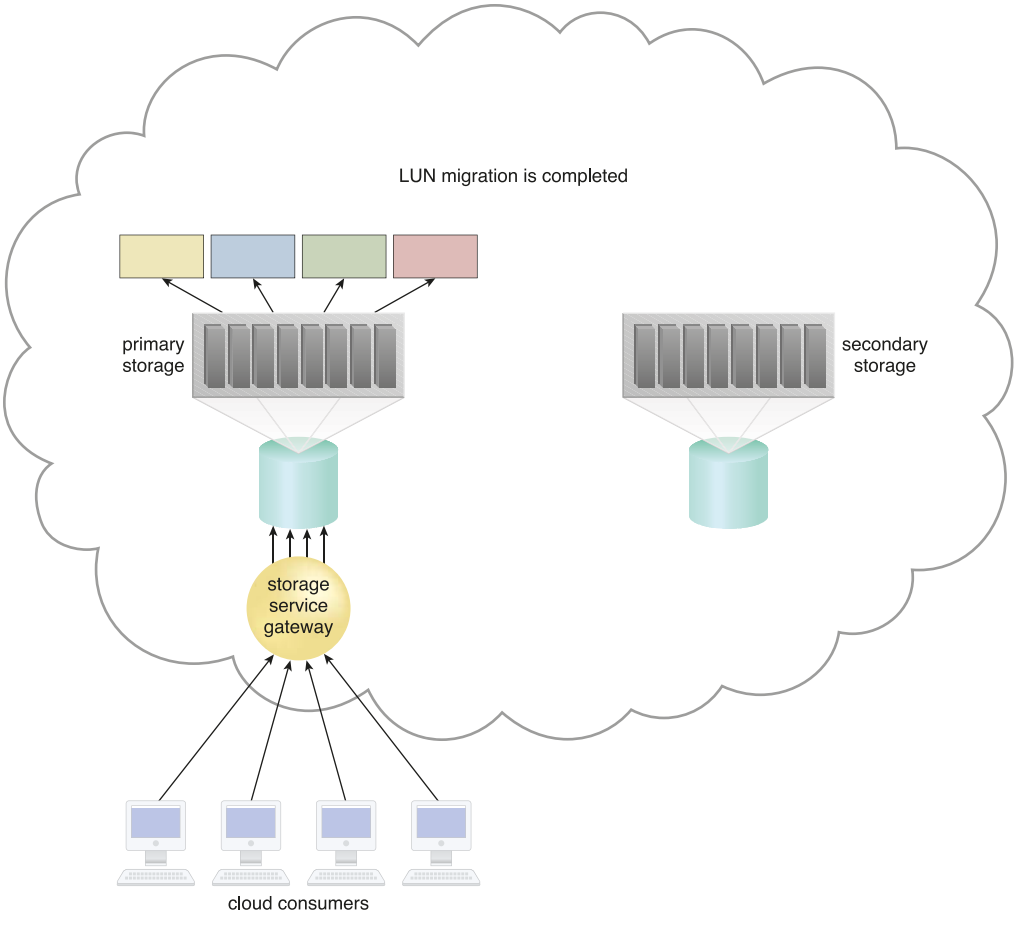


Figure 15.37

The live storage migration process is completed and all the data access requests are forwarded back to the primary cloud storage device.

In addition to the cloud storage device mechanism that is principal to this architecture, the resource replication mechanism is used to keep the primary and secondary storage devices synchronized. Both manually and automatically initiated failover can also be incorporated into this cloud architecture via the failover system mechanism even though the migration is often prescheduled.

NOTE

Edge and fog computing architectures establish environments outside of clouds but are covered here because these environments still relate to clouds and are primarily created in support of alleviating clouds from processing responsibilities so as to improve the performance, responsiveness, and scalability of consumer organization solutions.

Edge and fog computing architectures offer data processing and storage capacity closer to end user devices to streamline the processing and storage of data that will eventually be processed and stored in the cloud.

Edge and fog architectures are commonly used for IoT solutions in support of geographically distributed IoT devices. However, both architectures can be utilized to improve the effectiveness of standard business automation solutions for organizations, especially those with end users in multiple physical locations.

15.12 Edge Computing Architecture

An *edge computing architecture* introduces an intermediate processing layer that is physically positioned between the cloud and the cloud consumer. The edge environment is intentionally designed and located to be more accessible and performant for the consumer organization.

Portions of the cloud-based solution are moved to the edge environment, where they can be supported with dedicated infrastructure that enables them to perform faster, more responsively, and with greater scalability. Typically, the heavier processing responsibilities will remain with the cloud, while the parts of a solution with lower-end processing responsibilities are moved to the edge layer.

Edge architectures are typically utilized by consumer organizations with multiple, distributed physical locations. For each such location, a separate edge environment can be established (Figure 15.38). Edge computing environments can be implemented in suitable third-party locations that have the necessary resources, such as internet service providers and telecommunication providers.

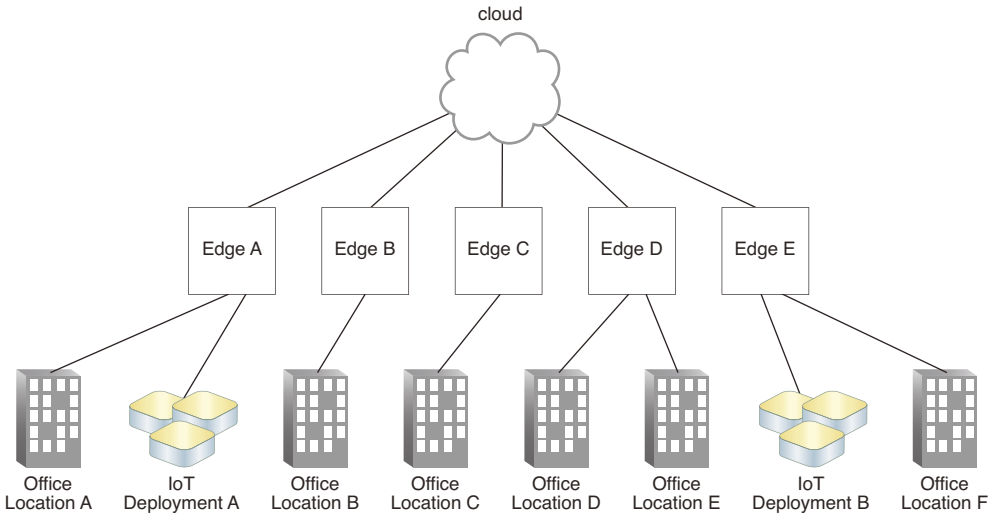


Figure 15.38

An edge computing architecture with a set of edge environments, each of which accommodates users or devices in a separate physical location.

Edge computing can benefit application architectures by reducing bandwidth requirements, optimizing resource utilization, improving security (by encrypting data closer to its origin), and even reducing power consumption.

15.13 Fog Computing Architecture

A *fog computing architecture* adds an additional processing layer in between edge environments and a cloud (Figure 15.39). This allows intermediate-level processing responsibilities to be moved from the cloud to fog environments, each of which can support and facilitate multiple edge environments.

Fog computing pushes data processing capacity from the cloud to the fog layer, where gateways may exist to effectively relay data back and forth between the edge environments and the cloud. When edge environments need to send massive volumes of data to the cloud, the fog environment can first determine which data carries more value to optimize the data transfers. The gateways in the fog then first send critical data to the cloud to be stored and processed, while the remaining data relayed by edge computers may need to then be locally processed by resources in the fog environment.

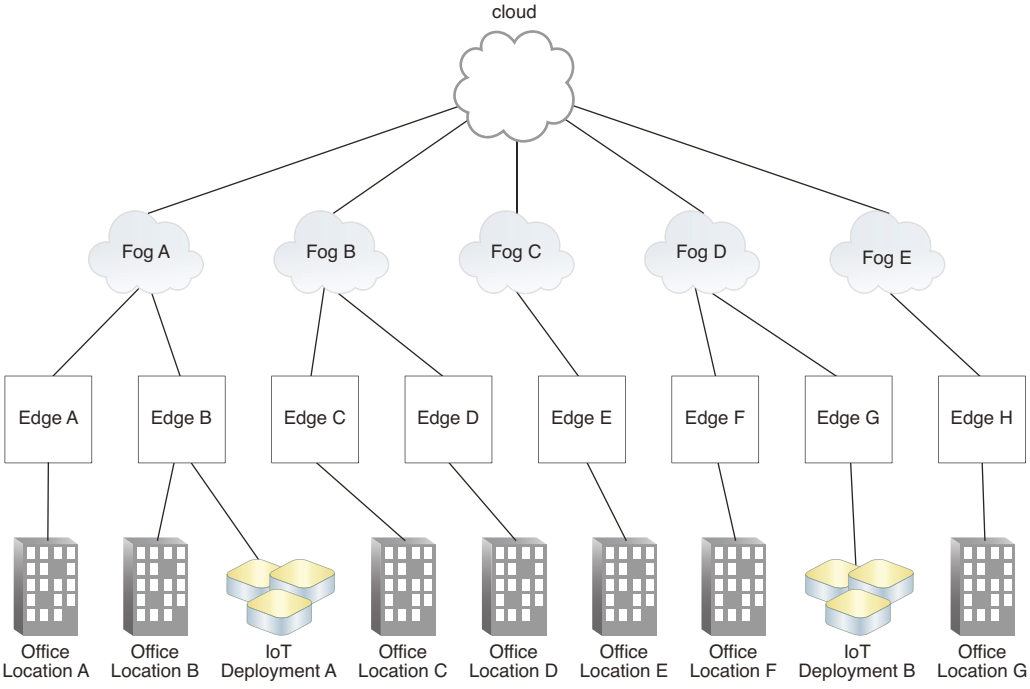


Figure 15.39

The use of the fog computing architecture inserts an intermediary processing layer between the cloud and the edge environments.

As with edge computing, fog computing is also commonly used to support IoT solutions. The use of fog computing for a business automation solution is generally warranted when the solution needs to support many users across highly distributed user bases.

NOTE

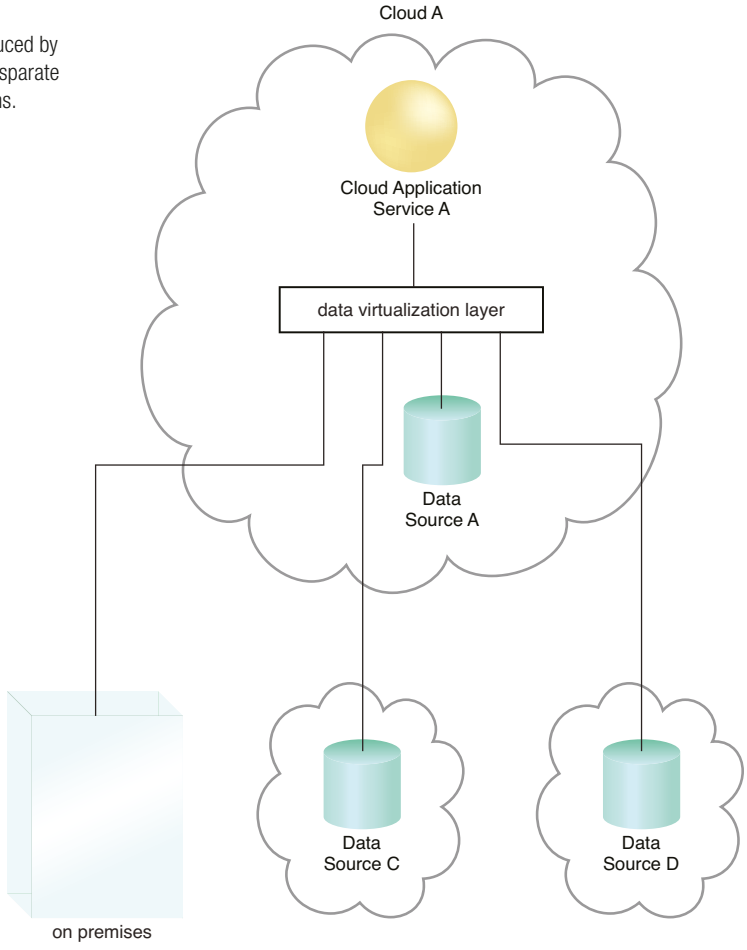
The remaining three architectures in this chapter originated from content published in *An Insider's Guide to Cloud Computing* (Pearson Education, ISBN: 9780137935697), authored by David Linthicum.

15.14 Virtual Data Abstraction Architecture

Cloud applications that require access to data sources that supply data in different formats, structures, and schemas will be burdened with the additional responsibility to transform and consolidate disparate data into relevant, uniform datasets. A further negative consequence is the tight coupling that the cloud applications need to form with data sources that may be subject to change, replacement, or retirement in the future.

The *virtual data abstraction architecture* alleviates these concerns by introducing a data virtualization layer that acts as the connection point for cloud applications that require access to disparate data sources (Figure 15.40). Within this layer, the data exists virtually in data virtualization software, which is configured to resolve the data structure differences to provide a single, uniform data API for cloud applications to access.

Figure 15.40
The data virtualization layer introduced by this architecture sits in between disparate data sources and cloud applications.



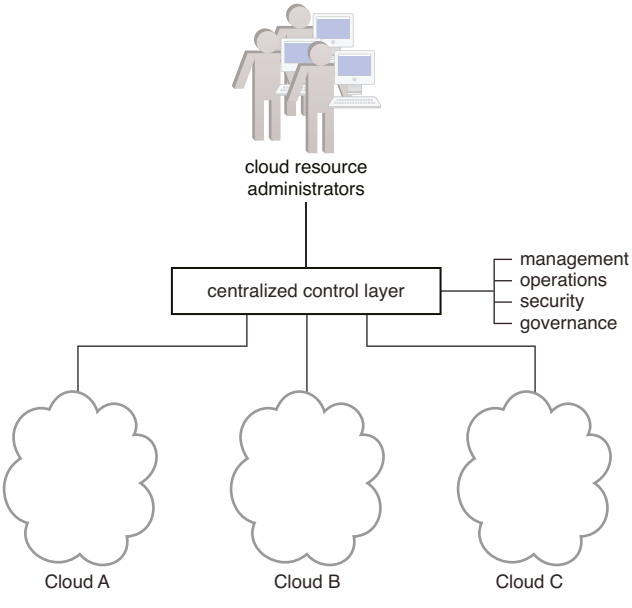
The use of the virtual data virtualization layer enables cloud applications to establish a loosely coupled relationship with disparate data sources. Should those data sources change over time, the data virtualization layer can be updated, ideally without changes to the APIs it exposes to the cloud applications.

15.15 Metacloud Architecture

While the multicloud architecture empowers cloud consumers with the flexibility to utilize diverse clouds to best fulfill business requirements, it can also introduce complexity when it comes to having to manage heterogeneity—meaning operate and govern multiple clouds, each with potentially different administration requirements, proprietary features, and security controls.

The *metacloud architecture* (Figure 15.41) abstracts these management, operational, and governance controls into a single logical domain that provides a central administration access point for the cloud consumer. This architecture is ideally established prior to proceeding with a multicloud architecture so that the centralized administration layer can be put in place from the start.

Figure 15.41
A metacloud architecture, in which a layer is introduced to abstract operational, management, security, and governance control.



The meta layer can be physically located wherever the cloud consumer chooses. It can be based in a specific cloud, distributed across multiple clouds, or even placed on premises. By abstracting management, operational, and governance controls into a central location, the cloud consumer can evolve its multicloud architecture more easily over time, which can significantly improve the organization's overall agility and responsiveness to business change.

15.16 Federated Cloud Application Architecture

A common limitation of distributed cloud applications is that their components or services are typically located within a single cloud environment. This limits the performance and functioning of those distributed application parts to the capacity and feature set of a single cloud's infrastructure.

When using a multicloud architecture, there is an opportunity to leverage the distributed nature of a cloud application by placing individual application components or services in different cloud environments to maximize the benefits each may have to offer. For example, for a given application service, one cloud may offer better high-performance compute power, another more resiliency, and another perhaps more favorable usage costs.

In a *federated cloud application architecture* (Figure 15.42), application components and services are distributed among available clouds, so that each is deployed in the most advantageous and beneficial location. This can result in a variety of improvements to the cloud application but will also introduce significant architectural complexity.

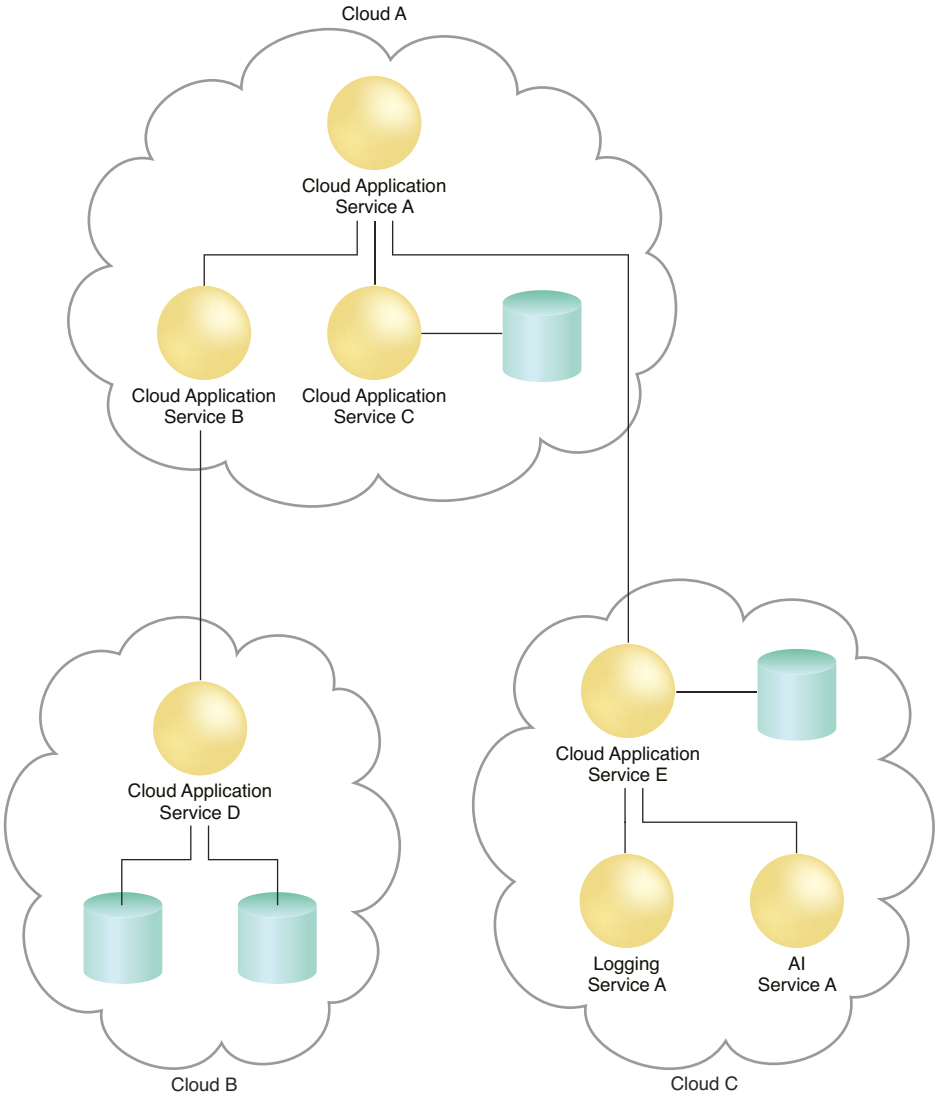
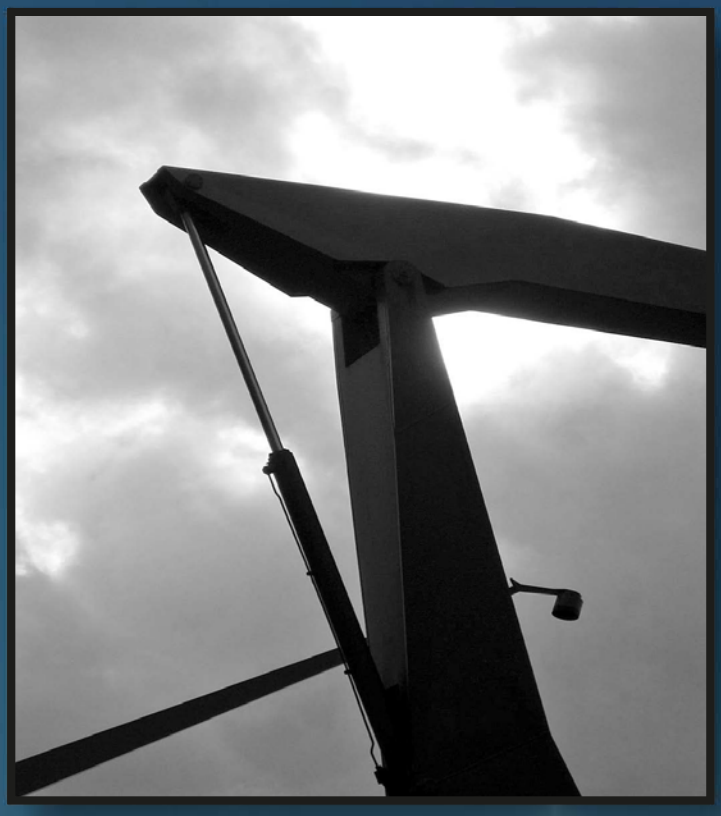


Figure 15.42

In a federated cloud application architecture, the distributed parts of the application can end up residing in different hosting environments, including different clouds and on-premises environments. Each part of the application is placed in a location that best supports its distinct requirements.

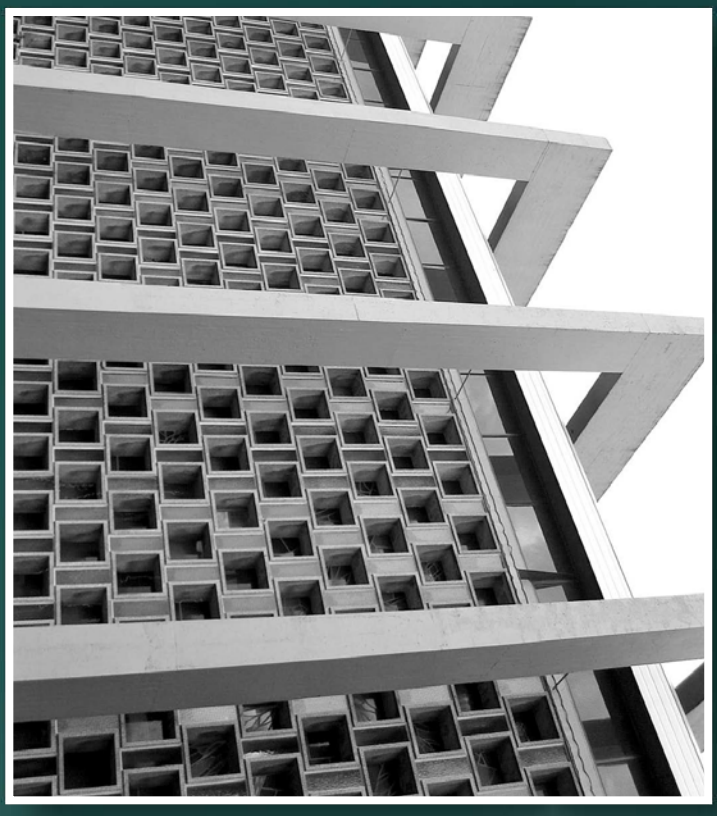
This page intentionally left blank



Part IV

Working with Clouds

Each of the chapters in this part of the book addresses a different topic area that pertains to planning or using cloud environments and cloud-based technologies. The numerous considerations, strategies, and metrics provided in these chapters help associate topics covered in preceding chapters with real-world requirements and constraints.



Chapter 16

Cloud Delivery Model Considerations

Most of the preceding chapters have been focused on technologies and models used to define and implement infrastructure and architecture layers within cloud environments. This chapter revisits the cloud delivery models that were introduced in Chapter 4 in order to address a number of real-world considerations within the context of IaaS-, PaaS-, and SaaS-based environments.

The chapter is organized into two primary sections that explore cloud delivery model issues pertaining to cloud providers and cloud consumers, respectively.

16.1 Cloud Delivery Models: The Cloud Provider Perspective

This section explores the architecture and administration of IaaS, PaaS, and SaaS cloud delivery models from the point of view of the cloud provider. The integration and management of these cloud-based environments as part of greater environments and how they can relate to different technologies and cloud mechanism combinations are examined.

Building IaaS Environments

The virtual server and cloud storage device mechanisms represent the two most fundamental IT resources that are delivered as part of a standard rapid provisioning architecture within IaaS environments. They are offered in various standardized configurations that are defined by the following properties:

- operating system
- primary memory capacity
- processing capacity
- virtualized storage capacity

Memory and virtualized storage capacity is usually allocated in increments of 1 GB to simplify the provisioning of underlying physical IT resources. When limiting cloud consumer access to virtualized environments, IaaS offerings are preemptively assembled

by cloud providers via virtual server images that capture the predefined configurations. Some cloud providers may offer cloud consumers direct administrative access to physical IT resources, in which case the bare-metal provisioning architecture may come into play.

Snapshots can be taken of a virtual server to record the current state, memory, and configuration of a virtualized IaaS environment for backup and replication purposes, in support of horizontal and vertical scaling requirements. For example, a virtual server can use its snapshot to become reinitialized in another hosting environment after its capacity has been increased to allow for vertical scaling. The snapshot can alternatively be used to duplicate a virtual server. The management of custom virtual server images is a vital feature that is provided via the remote administration system mechanism. Most cloud providers also support importing and exporting options for custom-built virtual server images in both proprietary and standard formats.

Data Centers

Cloud providers can offer IaaS-based IT resources from multiple geographically diverse data centers, which provides the following primary benefits:

- Multiple data centers can be linked together for increased resiliency. Each data center is placed in a different location to lower the chances of a single failure forcing all the data centers to go offline simultaneously.
- Connected through high-speed communications networks with low latency, data centers can perform load balancing, IT resource backup and replication, and increase storage capacity, while improving availability and reliability. Having multiple data centers spread over a greater area further reduces network latency.
- Data centers that are deployed in different countries make access to IT resources more convenient for cloud consumers that are constricted by legal and regulatory requirements.

Figure 16.1 provides an example of a cloud provider that is managing four data centers that are split between two different geographic regions.

When an IaaS environment is used to provide cloud consumers with virtualized network environments, each cloud consumer is segregated into a tenant environment that isolates IT resources from the rest of the cloud through the internet. VLANs and network access control software collaboratively realize the corresponding logical network perimeters.

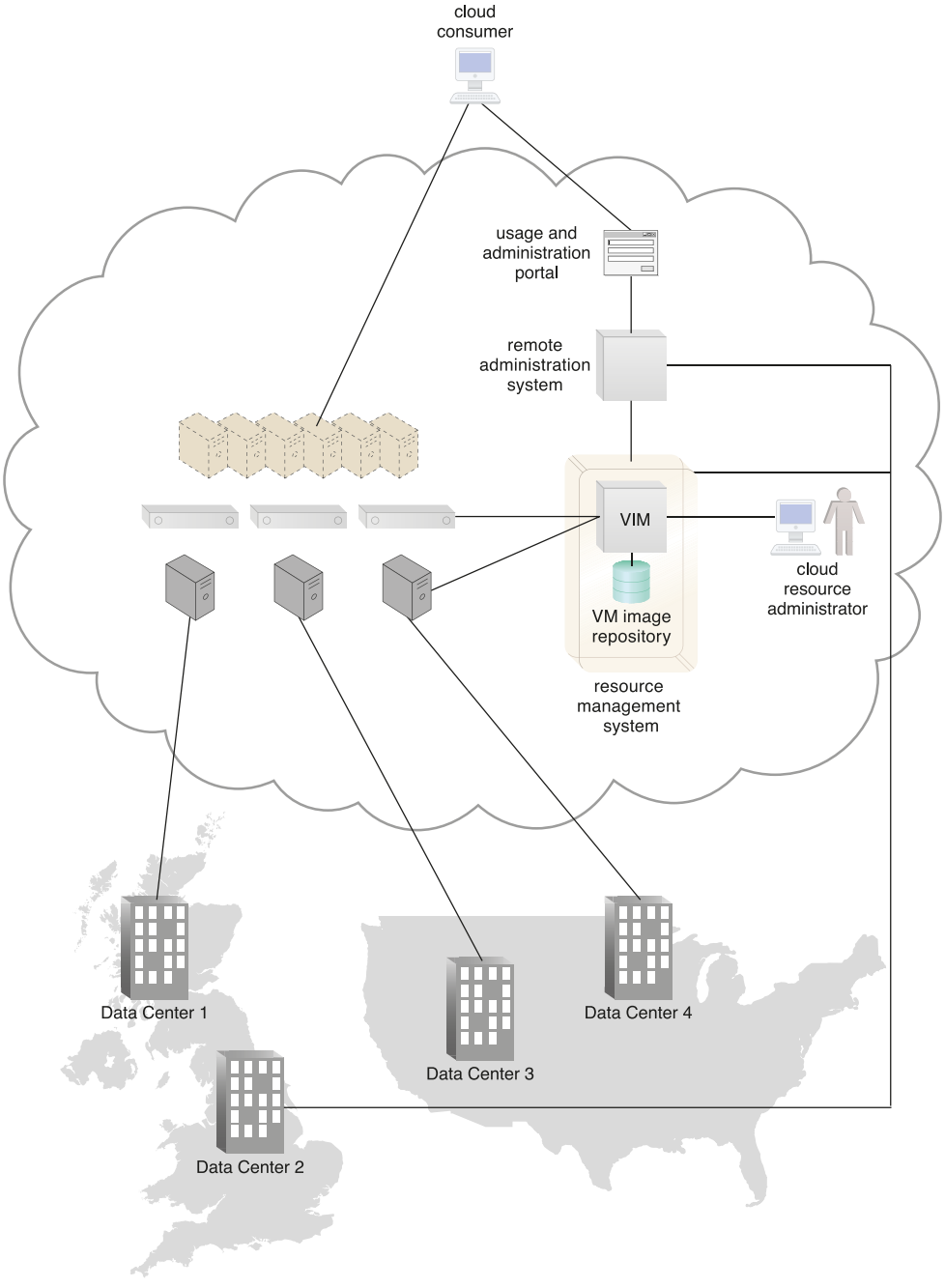


Figure 16.1

A cloud provider is provisioning and managing an IaaS environment with IT resources from different data centers in the United States and the United Kingdom.

Scalability and Reliability

Within IaaS environments, cloud providers can automatically provision virtual servers via the dynamic vertical scaling type of the dynamic scalability architecture. This can be performed through the VIM as long as the host physical servers have sufficient capacity. The VIM can scale virtual servers out using resource replication as part of a resource pool architecture if a given physical server has insufficient capacity to support vertical scaling. The load balancer mechanism, as part of a workload distribution architecture, can be used to distribute the workload among IT resources in a pool to complete the horizontal scaling process.

Manual scalability requires the cloud consumer to interact with a usage and administration program to explicitly request IT resource scaling. In contrast, automatic scalability requires the automated scaling listener to monitor the workload and reactively scale the resource capacity. This mechanism typically acts as a monitoring agent that tracks IT resource usage to notify the resource management system when capacity has been exceeded.

Replicated IT resources can be arranged in a high-availability configuration that forms a failover system for implementation via standard VIM features. Alternatively, a high-availability/high-performance resource cluster can be created at the physical or virtual server level, or both simultaneously. The multipath resource access architecture is commonly employed to enhance reliability via the use of redundant access paths, and some cloud providers further offer the provisioning of dedicated IT resources via the resource reservation architecture.

Monitoring

Cloud usage monitors in an IaaS environment can be implemented using the VIM or specialized monitoring tools that directly comprise and/or interface with the virtualization platform. Several common capabilities of the IaaS platform involve monitoring:

- *Virtual Server Lifecycles* – Recording and tracking uptime periods and the allocation of IT resources, for pay-per-use monitors and time-based billing purposes.
- *Data Storage* – Tracking and assigning the allocation of storage capacity to cloud storage devices on virtual servers, for pay-per-use monitors that record storage usage for billing purposes.
- *Network Traffic* – For pay-per-use monitors that measure inbound and outbound network usage and SLA monitors that track QoS metrics, such as response times and network losses.

- *Failure Conditions* – For SLA monitors that track IT resource and QoS metrics to provide warning in times of failure.
- *Event Triggers* – For audit monitors that appraise and evaluate the regulatory compliance of select IT resources.

Monitoring architectures within IaaS environments typically involve service agents that communicate directly with back-end management systems.

Security

Cloud security mechanisms that are relevant for securing IaaS environments include:

- encryption, hashing, digital signature, and PKI mechanisms for overall protection of data transmission
- IAM and SSO mechanisms for accessing services and interfaces in security systems that rely on user identification, authentication, and authorization capabilities
- cloud-based security groups for isolating virtual environments through hypervisors and network segments via network management software
- hardened virtual server images for internal and externally available virtual server environments
- various cloud usage monitors to track provisioned virtual IT resources to detect abnormal usage patterns

Equipping PaaS Environments

PaaS environments typically need to be outfitted with a selection of application development and deployment platforms to accommodate different programming models, languages, and frameworks. A separate ready-made environment is usually created for each programming stack that contains the necessary software to run applications specifically developed for the platform.

Each platform is accompanied by a matching SDK and IDE, which can be custom-built or enabled by IDE plugins supplied by the cloud provider. IDE toolkits can simulate the cloud runtime locally within the PaaS environment and usually include executable application servers. The security restrictions that are inherent to the runtime are also simulated in the development environment, including checks for unauthorized attempts to access system IT resources.

Cloud providers often offer a resource management system mechanism that is customized for the PaaS platform so that cloud consumers can create and control customized virtual server images with ready-made environments. This mechanism also provides features specific to the PaaS platform, such as managing deployed applications and configuring multitenancy. Cloud providers further rely on a variation of the rapid provisioning architecture known as platform provisioning, which is designed specifically to provision ready-made environments.

Scalability and Reliability

The scalability requirements of cloud services and applications that are deployed within PaaS environments are generally addressed via dynamic scalability and workload distribution architectures that rely on the use of native automated scaling listeners and load balancers. The resource pooling architecture is further utilized to provision IT resources from resource pools made available to multiple cloud consumers.

Cloud providers can evaluate network traffic and server-side connection usage against the instance's workload when determining how to scale an overloaded application as per parameters and cost limitations provided by the cloud consumer. Alternatively, cloud consumers can configure the application designs to customize the incorporation of available mechanisms themselves.

The reliability of ready-made environments and hosted cloud services and applications can be supported with standard failover system mechanisms (Figure 16.2), as well as the nondisruptive service relocation architecture so as to shield cloud consumers from failover conditions. The resource reservation architecture may also be in place to offer exclusive access to PaaS-based IT resources. As with other IT resources, ready-made environments can also span multiple data centers and geographical regions to further increase availability and resiliency.

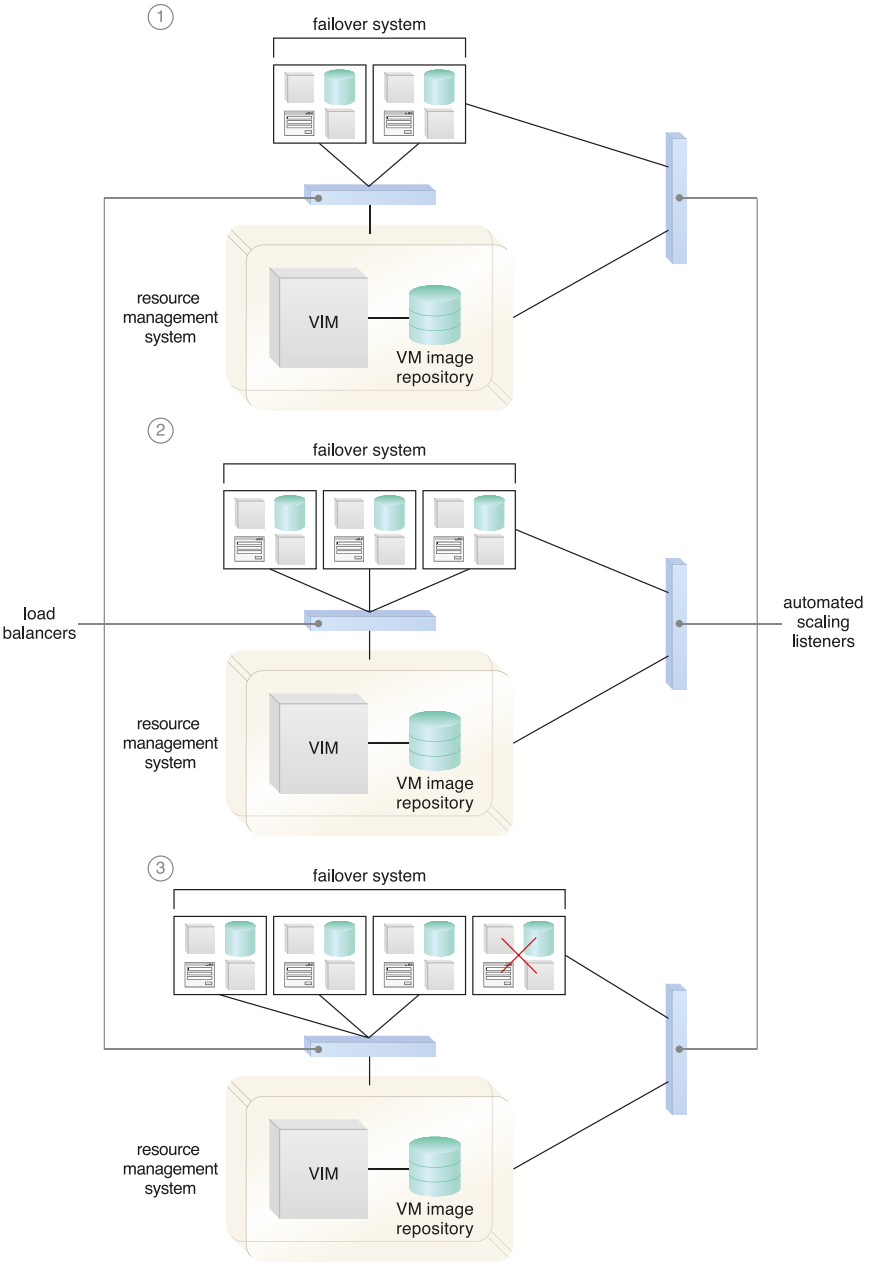


Figure 16.2

Load balancers are used to distribute ready-made environment instances that are part of a failover system, while automated scaling listeners are used to monitor the network and instance workloads (1). The ready-made environments are scaled out in response to an increase in workload (2), and the failover system detects a failure condition and stops replicating a failed ready-made environment (3).

Monitoring

Specialized cloud usage monitors in PaaS environments are used to monitor the following:

- *Ready-Made Environment Instances* – The applications of these instances are recorded by pay-per-use monitors for the calculation of time-based usage fees.
- *Data Persistence* – This statistic is provided by pay-per-use monitors that record the number of objects, individual occupied storage sizes, and database transactions per billing period.
- *Network Usage* – Inbound and outbound network usage is tracked for pay-per-use monitors and SLA monitors that track network-related QoS metrics.
- *Failure Conditions* – SLA monitors that track the QoS metrics of IT resources need to capture failure statistics.
- *Event Triggers* – This metric is primarily used by audit monitors that need to respond to certain types of events.

Security

The PaaS environment, by default, does not usually introduce the need for new cloud security mechanisms beyond those that are already provisioned for IaaS environments.

Optimizing SaaS Environments

In SaaS implementations, cloud service architectures are generally based on multitenant environments that enable and regulate concurrent cloud consumer access (Figure 16.3). SaaS IT resource segregation does not typically occur at the infrastructure level in SaaS environments as it does in IaaS and PaaS environments.

SaaS implementations rely heavily on the features provided by the native dynamic scalability and workload distribution architectures, as well as nondisruptive service relocation to ensure that failover conditions do not impact the availability of SaaS-based cloud services.

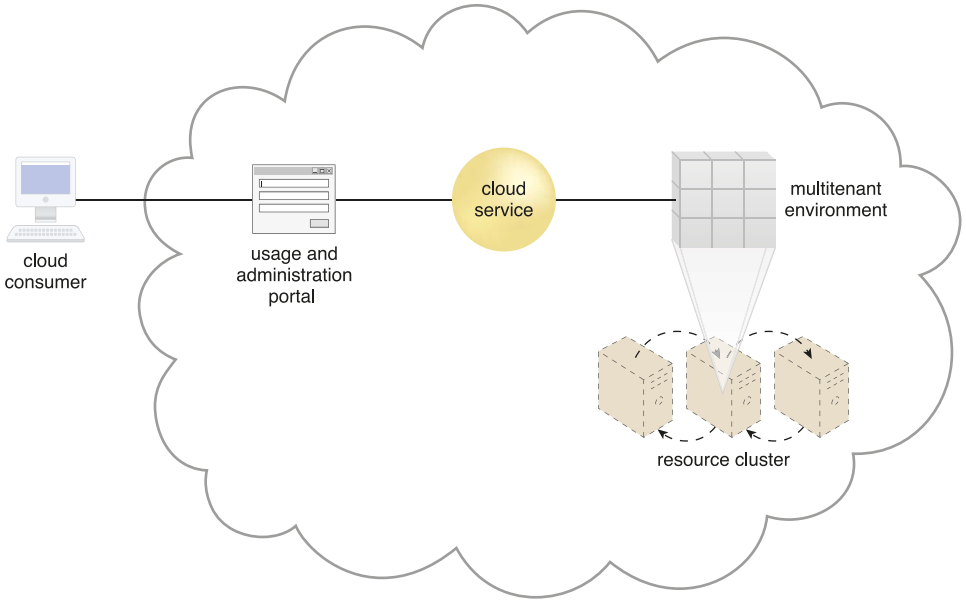


Figure 16.3

The SaaS-based cloud service is hosted by a multitenant environment deployed in a high-performance virtual server cluster. A usage and administration portal is used by the cloud consumer to access and configure the cloud service.

However, it is vital to acknowledge that, unlike the relatively vanilla designs of IaaS and PaaS products, each SaaS deployment will bring with it unique architectural, functional, and runtime requirements. These requirements are specific to the nature of the business logic the SaaS-based cloud service is programmed with, as well as the distinct usage patterns it is subjected to by its cloud service consumers.

For example, consider the diversity in functionality and usage of the following recognized online SaaS offerings:

- collaborative authoring and information-sharing (Wikipedia, Blogger)
- collaborative management (Zimbra, Google Apps)
- conferencing services for instant messaging, audio/video communications (Zoom, Skype, Google Meet)
- enterprise management systems (ERP, CRM, CM)
- file-sharing and content distribution (YouTube, Dropbox)

- industry-specific software (engineering, bioinformatics)
- messaging systems (email, voicemail)
- mobile application marketplaces (Google Play Store, Apple App Store)
- office productivity software suites (Microsoft Office, Adobe Creative Cloud)
- search engines (Google, Yahoo)
- social networking media (Twitter, LinkedIn)

Now consider that many of the previously listed cloud services are offered in one or more of the following implementation mediums:

- mobile application
- REST service
- web service

Each of these SaaS implementation mediums provides web-based APIs for interfacing by cloud consumers. Examples of online SaaS-based cloud services with web-based APIs include:

- electronic payment services (PayPal)
- mapping and routing services (Google Maps)
- publishing tools (WordPress)

Mobile-enabled SaaS implementations are commonly supported by the multi-device broker mechanism, unless the cloud service is intended exclusively for access by specific mobile devices.

The potentially diverse nature of SaaS functionality, the variation in implementation technology, and the tendency to offer a SaaS-based cloud service redundantly with multiple different implementation mediums makes the design of SaaS environments highly specialized. Though not essential to a SaaS implementation, specialized processing requirements can prompt the need to incorporate architectural models, such as:

- *Service Load Balancing* – for workload distribution across redundant SaaS-based cloud service implementations
- *Dynamic Failure Detection and Recovery* – to establish a system that can automatically resolve some failure conditions without disruption of service to the SaaS implementation

- *Storage Maintenance Window* – to allow for planned maintenance outages that do not impact SaaS implementation availability
- *Elastic Resource Capacity/Elastic Network Capacity* – to establish inherent elasticity within the SaaS-based cloud service architecture that enables it to automatically accommodate a range of runtime scalability requirements
- *Cloud Balancing* – to instill broad resiliency within the SaaS implementation, which can be especially important for cloud services subjected to extreme concurrent usage volumes

Specialized cloud usage monitors can be used in SaaS environments to track the following types of metrics:

- *Tenant Subscription Period* – This metric is used by pay-per-use monitors to record and track application usage for time-based billing. This type of monitoring usually incorporates application licensing and regular assessments of leasing periods that extend beyond the hourly periods of IaaS and PaaS environments.
- *Application Usage* – This metric, based on user or security groups, is used with pay-per-use monitors to record and track application usage for billing purposes.
- *Tenant Application Functional Module* – This metric is used by pay-per-use monitors for function-based billing. Cloud services can have different functionality tiers according to whether the cloud consumer is free-tier or a paid subscriber.

Similar to the cloud usage monitoring that is performed in IaaS and PaaS implementations, SaaS environments are also commonly monitored for data storage, network traffic, failure conditions, and event triggers.

Security

SaaS implementations generally rely on a foundation of security controls inherent to their deployment environment. Distinct business processing logic will then add layers of additional cloud security mechanisms or specialized security technologies.

16.2 Cloud Delivery Models: The Cloud Consumer Perspective

This section raises various considerations concerning the different ways in which cloud delivery models are administered and utilized by cloud consumers.

Working with IaaS Environments

Virtual servers are accessed at the operating system level through the use of remote terminal applications. Accordingly, the type of client software used directly depends on the type of operating system that is running at the virtual server, of which two common options are:

- *Remote Desktop (or Remote Desktop Connection) Client* – for Windows-based environments and presents a Windows GUI desktop
- *SSH Client* – for Mac and Linux-based environments to allow for secure channel connections to text-based shell accounts running on the server operating system

Figure 16.4 illustrates a typical usage scenario for virtual servers that are being offered as IaaS services after having been created with management interfaces.

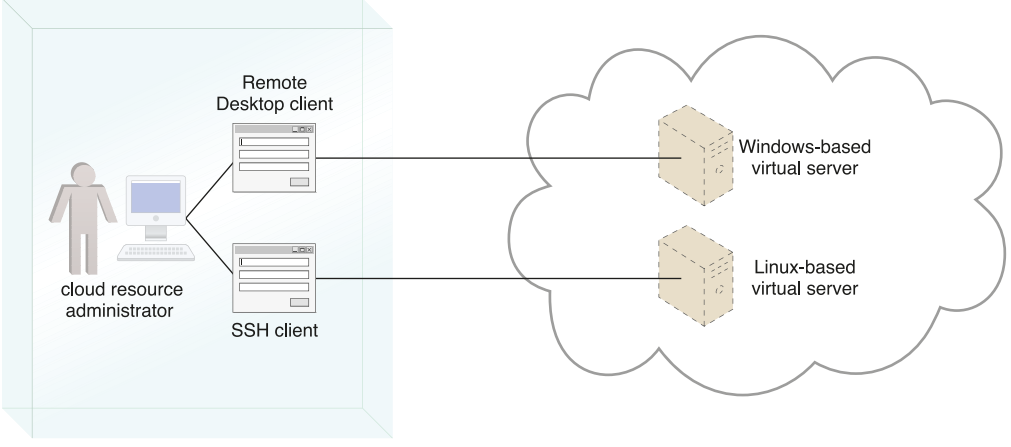


Figure 16.4

A cloud resource administrator uses the Windows-based Remote Desktop client to administer a Windows-based virtual server and the SSH client for the Linux-based virtual server.

A cloud storage device can be attached directly to the virtual servers and accessed through the virtual servers' functional interface for management by the operating system. Alternatively, a cloud storage device can be attached to an IT resource that is being hosted outside of the cloud, such as an on-premises device over a WAN or VPN. In these cases, the following formats for the manipulation and transmission of cloud storage data are commonly used:

- *Networked File System* – system-based storage access, whose rendering of files is similar to how folders are organized in operating systems (NFS, CIFS)
- *Storage Area Network Devices* – block-based storage access collates and formats geographically diverse data into cohesive files for optimal network transmission (iSCSI, Fibre Channel)
- *Web-Based Resources* – object-based storage access by which an interface that is not integrated into the operating system logically represents files, which can be accessed through a web-based interface (Amazon S3)

IT Resource Provisioning Considerations

Cloud consumers have a high degree of control over how and to what extent IT resources are provisioned as part of their IaaS environments.

For example:

- controlling scalability features (automated scaling, load balancing)
- controlling the lifecycle of virtual IT resources (shutting down, restarting, powering up of virtual devices)
- controlling the virtual network environment and network access rules (firewalls, logical network perimeters)
- establishing and displaying service provisioning agreements (account conditions, usage terms)
- managing the attachment of cloud storage devices
- managing the preallocation of cloud-based IT resources (resource reservation)
- managing credentials and passwords for cloud resource administrators
- managing credentials for cloud-based security groups that access virtualized IT resources through an IAM

- managing security-related configurations
- managing customized virtual server image storage (importing, exporting, backup)
- selecting high-availability options (failover, IT resource clustering)
- selecting and monitoring SLA metrics
- selecting basic software configurations (operating system, preinstalled software for new virtual servers)
- selecting IaaS resource instances from a number of available hardware-related configurations and options (processing capabilities, RAM, storage)
- selecting the geographical regions in which cloud-based IT resources should be hosted
- tracking and managing costs

The management interface for these types of provisioning tasks is typically a usage and administration portal, but may also be offered via the use of command-line interface (CLI) tools that can simplify the execution of many scripted administrative actions.

Even though standardizing the presentation of administrative features and controls is typically preferred, using different tools and user interfaces can sometimes be justified. For example, a script can be made to turn virtual servers on and off nightly through a CLI, while adding or removing storage capacity can be more easily carried out using a portal.

Working with PaaS Environments

A typical PaaS IDE can offer a wide range of tools and programming resources, such as software libraries, class libraries, frameworks, APIs, and various runtime capabilities that emulate the intended cloud-based deployment environment. These features allow developers to create, test, and run application code within the cloud or locally (on premises) while using the IDE to emulate the cloud deployment environment. Compiled or completed applications are then bundled and uploaded to the cloud and deployed via the ready-made environments. This deployment process can also be controlled through the IDE.

PaaS also allows for applications to use cloud storage devices as independent data storing systems for holding development-specific data (for example, in a repository that is available outside of the cloud environment). Both SQL and NoSQL database structures are generally supported.

IT Resource Provisioning Considerations

PaaS environments provide less administrative control than IaaS environments, but still offer a significant range of management features.

For example:

- establishing and displaying service provisioning agreements, such as account conditions and usage terms
- selecting software platform and development frameworks for ready-made environments
- selecting instance types, which are most commonly front-end or back-end instances
- selecting cloud storage devices for use in ready-made environments
- controlling the lifecycle of PaaS-developed applications (deployment, starting, shutdown, restarting, and release)
- controlling the versioning of deployed applications and modules
- configuring availability- and reliability-related mechanisms
- managing credentials for developers and cloud resource administrators using IAM
- managing general security settings, such as accessible network ports
- selecting and monitoring PaaS-related SLA metrics
- managing and monitoring usage and IT resource costs
- controlling scalability features such as usage quotas, active instance thresholds, and the configuration and deployment of the automated scaling listener and load balancer mechanisms

The usage and administration portal that is used to access PaaS management features can provide the feature of preemptively selecting the times at which an IT resource

is started and stopped. For example, a cloud resource administrator can set a cloud storage device to turn itself on at 9:00AM then turn off twelve hours later. Building on this system can enable the option of having the ready-made environment self-activate upon receiving data requests for a particular application and turn off after an extended period of inactivity.

Working with SaaS Services

Because SaaS-based cloud services are almost always accompanied by refined and generic APIs, they are usually designed to be incorporated as part of larger distributed solutions. A common example of this is Google Maps, which offers a comprehensive API that enables mapping information and images to be incorporated into websites and web-based applications.

Many SaaS offerings are provided free of charge, although these cloud services often come with data collecting subprograms that harvest usage data for the benefit of the cloud provider. When using any SaaS product that is sponsored by third parties, there is a reasonable chance that it is performing a form of background information gathering. Reading the cloud provider's agreement will usually help shed light on any secondary activity that the cloud service is designed to perform.

Cloud consumers using SaaS products supplied by cloud providers are relieved of the responsibilities of implementing and administering their underlying hosting environments. Customization options are usually available to cloud consumers; however, these options are generally limited to the runtime usage control of the cloud service instances that are generated specifically by and for the cloud consumer.

For example:

- managing security-related configurations
- managing select availability and reliability options
- managing usage costs
- managing user accounts, profiles, and access authorization
- selecting and monitoring SLAs
- setting manual and automated scalability options and limitations

16.3 CASE STUDY EXAMPLE

DTGOV discovers that a number of additional mechanisms and technologies need to be assembled to complete its IaaS management architecture (Figure 16.5):

- Network virtualization is incorporated into logical network topologies, and logical network perimeters are established using different firewalls and virtual networks.
- The VIM is positioned as the central tool for controlling the IaaS platform and equipping it with self-provisioning capabilities.
- Additional virtual server and cloud storage device mechanisms are implemented through the virtualization platform, while several virtual server images that provide base template configurations for virtual servers are created.
- Dynamic scaling is added using the VIM's API through the use of automated scaling listeners.
- High-availability virtual server clusters are created using the resource replication, load balancer, failover system, and resource cluster mechanisms.
- A customized application that directly uses the SSO and IAM system mechanisms is built to enable interoperability between the remote administration system, network management tools, and VIM.

DTGOV uses a powerful commercial network management tool that is customized to store event information gathered by the VIM and SLA monitoring agents in an SLA measurements database. The management tool and database are used as part of a greater SLA management system. To enable billing processing, DTGOV expands a proprietary software tool that is based on a set of usage measurements from a database populated by pay-per-use monitors. The billing software is used as the base implementation for the billing management system mechanism.

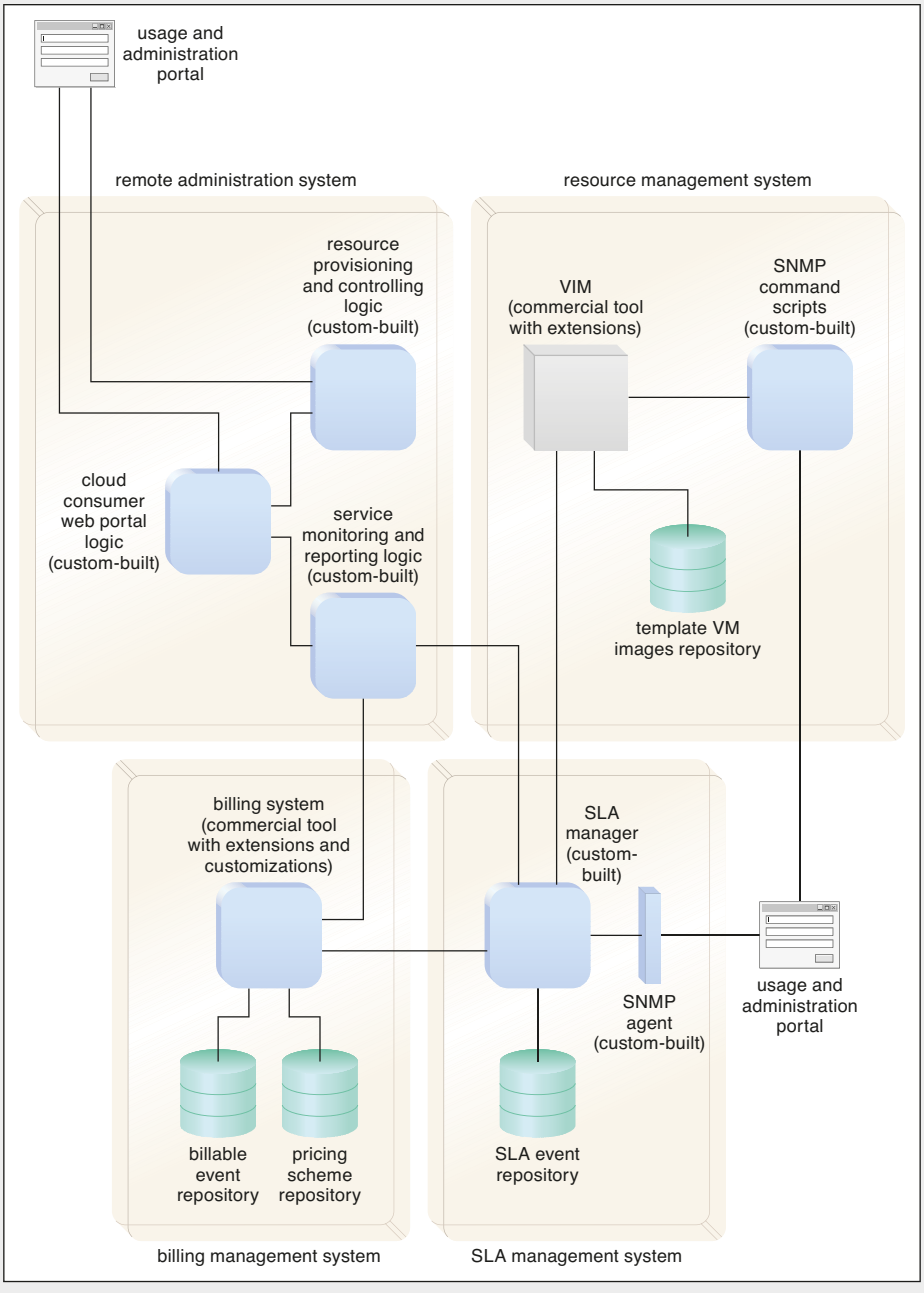


Figure 16.5
An overview of the DTGOV management architecture.

This page intentionally left blank

Chapter 17

Cost Metrics and Pricing Models

Reducing operating costs and optimizing IT environments are pivotal to understanding and being able to compare the cost models behind provisioning on-premises and cloud-based environments. The pricing structures used by public clouds are typically based on utility-centric pay-per-usage models, enabling organizations to avoid up-front infrastructure investments. These models need to be assessed against the financial implications of on-premises infrastructure investments and associated total cost-of-ownership commitments.

This chapter provides metrics, formulas, and practices to assist cloud consumers in performing accurate financial analysis of cloud adoption plans.

17.1 Business Cost Metrics

This section begins by describing the common types of metrics used to evaluate the estimated costs and business value of leasing cloud-based IT resources when compared to the purchase of on-premises IT resources.

Up-Front and Ongoing Costs

Up-front costs are associated with the initial investments that organizations need to make to fund the IT resources they intend to use. This includes the costs associated with obtaining the IT resources as well as expenses required to deploy and administer them.

- Up-front costs for the purchase and deployment of on-premises IT resources tend to be high. Examples of up-front costs for on-premises environments can include hardware, software, and the labor required for deployment.
- Up-front costs for the leasing of cloud-based IT resources tend to be low. Examples of up-front costs for cloud-based environments can include the labor costs required to assess and set up a cloud environment.

Ongoing costs represent the expenses required by an organization to run and maintain the IT resources it uses.

- Ongoing costs for the operation of on-premises IT resources can vary. Examples include licensing fees, electricity, insurance, and labor.
- Ongoing costs for the operation of cloud-based IT resources can also vary, but often exceed the ongoing costs of on-premises IT resources (especially over a longer period of time). Examples include virtual hardware leasing fees, bandwidth usage fees, licensing fees, and labor.

Additional Costs

To supplement and extend a financial analysis beyond the calculation and comparison of standard up-front and ongoing business cost metrics, several more specialized business cost metrics can be taken into account.

For example:

- *Cost of Capital* – The *cost of capital* is a value that represents the cost incurred by raising required funds. For example, it will generally be more expensive to raise an initial investment of $150,000 than it will be to raise this amount over a period of three years. The relevancy of this cost depends on how the organization goes about gathering the funds it requires. If the cost of capital for an initial investment is high, then it further helps justify the leasing of cloud-based IT resources.
- *Sunk Costs* – An organization will often have existing IT resources that are already paid for and operational. The prior investment that has been made in these on-premises IT resources is referred to as *sunk costs*. When comparing up-front costs together with significant sunk costs, it can be more difficult to justify the leasing of cloud-based IT resources as an alternative.
- *Integration Costs* – Integration testing is a form of testing required to measure the effort required to make IT resources compatible and interoperable within a foreign environment, such as a new cloud platform. Depending on the cloud deployment model and cloud delivery model being considered by an organization, there may be a need to further allocate funds to carry out integration testing and additional labor related to enable interoperability between cloud service consumers and cloud services. These expenses are referred to as *integration costs*. High integration costs can make the option of leasing cloud-based IT resources less appealing.

- *Locked-In Costs* – As explained in the *Risks and Challenges* section in Chapter 3, cloud environments can impose portability limitations. When performing a metrics analysis over a longer period of time, it may be necessary to take into consideration the possibility of having to move from one cloud provider to another. Due to the fact that cloud service consumers can become dependent on proprietary characteristics of a cloud environment, there are *locked-in costs* associated with this type of move. Locked-in costs can further decrease the long-term business value of leasing cloud-based IT resources.

CASE STUDY EXAMPLE

ATN performs a total cost-of-ownership (TCO) analysis on migrating two of its legacy applications to a PaaS environment. The report produced by the analysis examines comparative evaluations of on-premises and cloud-based implementations based on a three-year time frame.

The following sections provide a summary from the report for each of the two applications.

Product Catalog Browser

The Product Catalog Browser is a globally used web application that interoperates with the ATN web portal and several other systems. This application was deployed in a virtual server cluster that is comprised of 4 virtual servers running on 2 dedicated physical servers. The application has its own 300 GB database that resides in a separate HA cluster. Its code was recently generated from a refactoring project. Only minor portability issues needed to be addressed before it was ready to proceed with a cloud migration.

The TCO analysis reveals the following:

On-Premises Up-Front Costs

- Licensing: The purchase price for each physical server hosting the application is $7,500, while the software required to run all 4 servers totals $30,500.
- Labor: Labor costs are estimated to be $5,500, including setup and application deployment.

The total up-front costs are: ($7,500 × 2) + $30,500 + $5,500 = $51,000

The configuration of the servers is derived from a capacity plan that accounts for peak workloads. Storage was not assessed as part of this plan, since the application database is assumed to be only negligibly affected by the application's deployment.

On-Premises Ongoing Costs

The following are monthly ongoing costs:

- Environmental Fees: $750
- Licensing Fees: $520
- Hardware Maintenance: $100
- Labor: $2,600

The total on-premises ongoing costs are: $750 + $520 + $100 + $2,600 = $3,970

Cloud-Based Up-Front Costs

If the servers are leased from a cloud provider, there is no up-front cost for hardware or software. Labor costs are estimated at $5,000, which includes expenses for solving interoperability issues and application setup.

Cloud-Based Ongoing Costs

The following are monthly ongoing costs:

- Server Instance: The usage fee is calculated per virtual server at a rate of $1.25/hour per virtual server. For 4 virtual servers, this results in: 4 × ($1.25 × 720) = $3,600. However, the application consumption is equivalent to 2.3 servers when server instance scaling is factored in, meaning the actual ongoing server usage cost is $2,070.
- Database Server and Storage: Usage fees are calculated per database size, at a rate of $1.09/GB per month = $327.
- Network: Usage fees are calculated per outbound WAN traffic at the rate of $0.10/GB and a monthly volume of 420 GB = $42.
- Labor: Estimated at $800 per month, including expenses for cloud resource administration tasks.

The total ongoing costs are: $2,070 + $327 + $42 + $800 = $3,139

The TCO breakdown for the Product Catalog Browser application is provided in Table 17.1.

Up-Front Costs	Cloud Environment	On-Premises Environment
Hardware	$0	$15,000
Licensing	$0	$30,500
Labor	$5,000	$5,500
Total Up-Front Costs	**$5,000**	**$51,000**

Monthly Ongoing Costs	Cloud Environment	On-Premises Environment
Application Servers	$2,070	$0
Database Servers	$327	$0
WAN Network	$42	$0
Environment	$0	$750
Software Licensing	$0	$520
Hardware Maintenance	$0	$100
Administration	$800	$2,600
Total Ongoing Costs	**$3,139**	**$3,970**

Table 17.1
The TCO analysis for the Product Catalog Browser application.

A comparison of the respective TCOs over a three-year period for both approaches reveals the following:

- On-Premises TCO: $51,000 up-front + ($3,970 × 36) ongoing = $193,920
- Cloud-Based TCO: $5,000 up-front + ($3,139 × 36) ongoing = $118,004

Based on the results of the TCO analysis, ATN decides to migrate the application to the cloud.

17.2 Cloud Usage Cost Metrics

This section describes a set of usage cost metrics for calculating costs associated with cloud-based IT resource usage measurements:

- *Network Usage* – inbound and outbound network traffic, as well as intra-cloud network traffic
- *Server Usage* – virtual server allocation (and resource reservation)
- *Cloud Storage Device* – storage capacity allocation
- *Cloud Service* – subscription duration, number of nominated users, number of transactions (of cloud services and cloud-based applications)

For each usage cost metric, a description, measurement unit, and measurement frequency are provided, along with the cloud delivery model most applicable to the metric. Each metric is further supplemented with a brief example.

Network Usage

Defined as the amount of data that is transferred over a network connection, network usage is typically calculated using separately measured *inbound network usage traffic* and *outbound network usage traffic* metrics in relation to cloud services or other IT resources.

Inbound Network Usage Metric

- *Description* – inbound network traffic
- *Measurement* – Σ (sum), inbound network traffic in bytes
- *Frequency* – continuous and cumulative over a predefined period

- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – up to 1 GB free, $0.001/GB up to 10 TB per month

Outbound Network Usage Metric

- *Description* – outbound network traffic
- *Measurement* – Σ, outbound network traffic in bytes
- *Frequency* – continuous and cumulative over a predefined period
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – up to 1 GB free per month, $0.01/GB between 1 GB to 10 TB per month

Network usage metrics can be applied to WAN traffic between the IT resources of one cloud that are located in different geographical regions to calculate costs for synchronization, data replication, and related forms of processing. Conversely, LAN usage and other network traffic among IT resources that reside at the same data center are typically not tracked.

Intra-Cloud WAN Usage Metric

- *Description* – network traffic between geographically diverse IT resources of the same cloud
- *Measurement* – Σ, intra-cloud WAN traffic in bytes
- *Frequency* – continuous and cumulative over a predefined period
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – up to 500 MB free daily and $0.01/GB thereafter, $0.005/GB after 1 TB per month

Many cloud providers do not charge for inbound traffic to encourage cloud consumers to migrate data to the cloud. Some also do not charge for WAN traffic within the same cloud.

Network-related cost metrics are determined by the following properties:

- *Static IP Address Usage* – IP address allocation time (if a static IP is required)
- *Network Load Balancing* – the amount of load-balanced network traffic (in bytes)
- *Virtual Firewall* – the amount of firewall-processed network traffic (as per allocation time)

Server Usage

The allocation of virtual servers is measured using common pay-per-use metrics in IaaS and PaaS environments that are quantified by the number of virtual servers and ready-made environments. This form of server usage measurement is divided into *on-demand virtual machine instance allocation* and *reserved virtual machine instance allocation* metrics.

The former metric measures pay-per-usage fees on a short-term basis, while the latter metric calculates up-front reservation fees for using virtual servers over extended periods. The up-front reservation fee is typically used in conjunction with the discounted pay-per-usage fees.

On-Demand Virtual Machine Instance Allocation Metric

- *Description* – uptime of a virtual server instance
- *Measurement* – Σ, virtual server start date to stop date
- *Frequency* – continuous and cumulative over a predefined period
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – $0.10/hour small instance, $0.20/hour medium instance, $0.90/hour large instance

Reserved Virtual Machine Instance Allocation Metric

- *Description* – up-front cost for reserving a virtual server instance
- *Measurement* – Σ, virtual server reservation start date to expiry date
- *Frequency* – daily, monthly, yearly
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – $55.10/small instance, $99.90/medium instance, $249.90/large instance

Another common cost metric for virtual server usage measures performance capabilities. Cloud providers of IaaS and PaaS environments tend to provision virtual servers with a range of performance attributes that are generally determined by CPU and RAM consumption and the amount of available dedicated allocated storage.

Cloud Storage Device Usage

Cloud storage is generally charged based on the amount of space allocated within a predefined period, as measured by the *on-demand storage allocation* metric. Similar to IaaS-based cost metrics, on-demand storage allocation fees are usually based on short time increments (such as on an hourly basis). Another common cost metric for cloud storage is *I/O data transferred*, which measures the amount of transferred input and output data.

On-Demand Storage Space Allocation Metric

- *Description* – duration and size of on-demand storage space allocation in bytes
- *Measurement* – Σ, date of storage release / reallocation to date of storage allocation (resets upon change in storage size)
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – $0.01/GB per hour (typically expressed as GB/month)

I/O Data Transferred Metric

- *Description* – amount of transferred I/O data
- *Measurement* – Σ, I/O data in bytes
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – $0.10/TB

Note that some cloud providers do not charge for I/O usage for IaaS and PaaS implementations, and limit charges to storage space allocation only.

Cloud Service Usage

Cloud service usage in SaaS environments is typically measured using the following three metrics.

Application Subscription Duration Metric

- *Description* – duration of cloud service usage subscription
- *Measurement* – Σ, subscription start date to expiry date

- *Frequency* – daily, monthly, yearly
- *Cloud Delivery Model* – SaaS
- *Example* – $69.90 per month

Number of Nominated Users Metric

- *Description* – number of registered users with legitimate access
- *Measurement* – number of users
- *Frequency* – monthly, yearly
- *Cloud Delivery Model* – SaaS
- *Example* – $0.90/additional user per month

Number of Transactions Users Metric

- *Description* – number of transactions served by the cloud service
- *Measurement* – number of transactions (request–response message exchanges)
- *Frequency* – continuous
- *Cloud Delivery Model* – PaaS, SaaS
- *Example* – $0.05 per 1,000 transactions

17.3 Cost Management Considerations

Cost management is often centered on the lifecycle phases of cloud services, as follows:

- *Cloud Service Design and Development* – During this stage, the vanilla pricing models and cost templates are typically defined by the organization delivering the cloud service.
- *Cloud Service Deployment* – Prior to and during the deployment of a cloud service, the back-end architecture for usage measurement and billing-related data collection is determined and implemented, including the positioning of pay-per-use monitor and billing management system mechanisms.
- *Cloud Service Contracting* – This phase consists of negotiations between the cloud consumer and the cloud provider with the goal of reaching a mutual agreement on rates based on usage cost metrics.

- *Cloud Service Offering* – This stage entails the concrete offering of a cloud service's pricing models through cost templates, and any available customization options.
- *Cloud Service Provisioning* – Cloud service usage and instance creation thresholds may be imposed by the cloud provider or set by the cloud consumer. Either way, these and other provisioning options can impact usage costs and other fees.
- *Cloud Service Operation* – This is the phase during which active usage of the cloud service produces usage cost metric data.
- *Cloud Service Decommissioning* – When a cloud service is temporarily or permanently deactivated, statistical cost data may be archived.

Both cloud providers and cloud consumers can implement cost management systems that reference or build upon the aforementioned lifecycle phases (Figure 17.1). It is also possible for the cloud provider to carry out some cost management stages on behalf of the cloud consumer and to then provide the cloud consumer with regular reports.

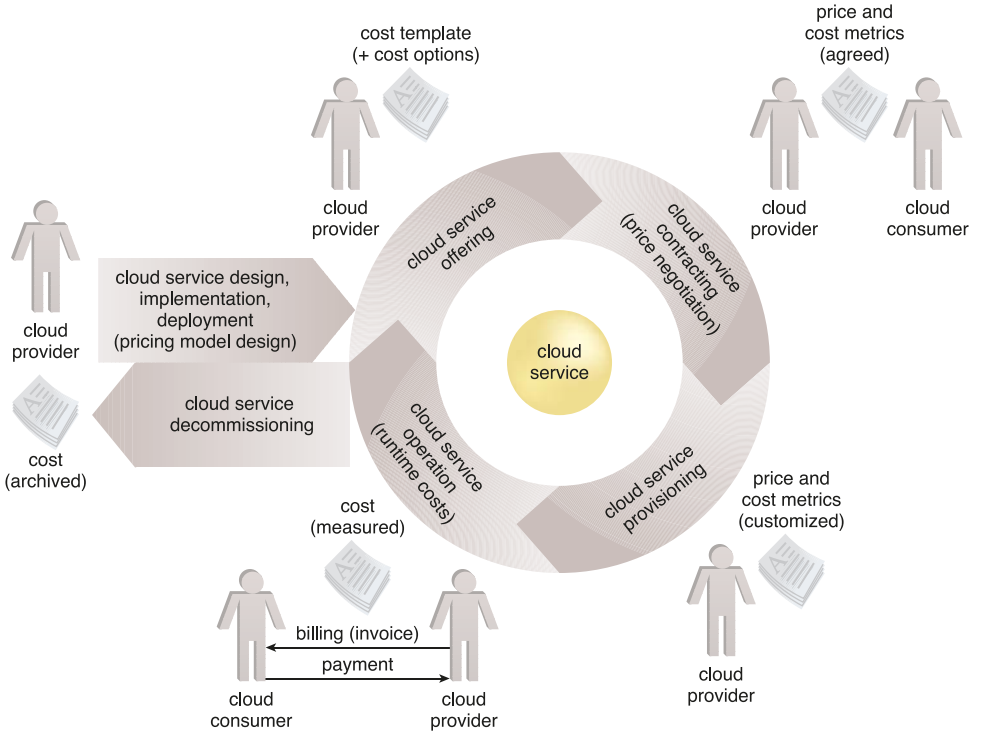


Figure 17.1

Common cloud service lifecycle stages as they relate to cost management considerations.

Pricing Models

The pricing models used by cloud providers are defined using templates that specify unit costs for fine-grained resource usage according to usage cost metrics. Various factors can influence a pricing model, such as:

- market competition and regulatory requirements
- overhead incurred during the design, development, deployment, and operation of cloud services and other IT resources
- opportunities to reduce expenses via IT resource sharing and data center optimization

Most major cloud providers offer cloud services at relatively stable, competitive prices even though their own expenses can be volatile. A price template or pricing plan contains a set of standardized costs and metrics that specify how cloud service fees are measured and calculated. Price templates define a pricing model's structure by setting various units of measure, usage quotas, discounts, and other codified fees. A pricing model can contain multiple price templates, whose formulation is determined by variables such as:

- *Cost Metrics and Associated Prices* – These are costs that are dependent on the type of IT resource allocation (such as on-demand versus reserved allocation).
- *Fixed and Variable Rates Definitions* – Fixed rates are based on resource allocation and define the usage quotas included in the fixed price, while variable rates are aligned with actual resource usage.
- *Volume Discounts* – More IT resources are consumed as the degree of IT resource scaling progressively increases, thereby possibly qualifying a cloud consumer for higher discounts.
- *Cost and Price Customization Options* – This variable is associated with payment options and schedules. For example, cloud consumers may be able to choose monthly, semi-annual, or annual payment installments.

Price templates are important for cloud consumers that are appraising cloud providers and negotiating rates, since they can vary depending on the adopted cloud delivery model.

For example:

- *IaaS* – Pricing is usually based on IT resource allocation and usage, which includes the amount of transferred network data, number of virtual servers, and allocated storage capacity.
- *PaaS* – Similar to IaaS, this model typically defines pricing for network data transferred, virtual servers, and storage. Prices are variable depending on factors such as software configurations, development tools, and licensing fees.
- *SaaS* – Because this model is solely concerned with application software usage, pricing is determined by the number of application modules in the subscription, the number of nominated cloud service consumers, and the number of transactions.

It is possible for a cloud service that is provided by one cloud provider to be built upon IT resources provisioned from another cloud provider. Figures 17.2 and 17.3 explore two sample scenarios.

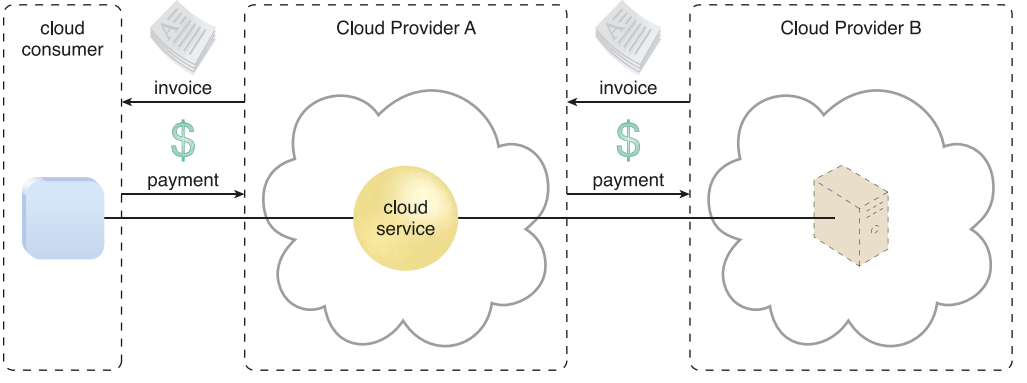


Figure 17.2

An integrated pricing model, whereby the cloud consumer leases a SaaS product from Cloud Provider A, which is leasing an IaaS environment (including the virtual server used to host the cloud service) from Cloud Provider B. The cloud consumer pays Cloud Provider A. Cloud Provider A pays Cloud Provider B.

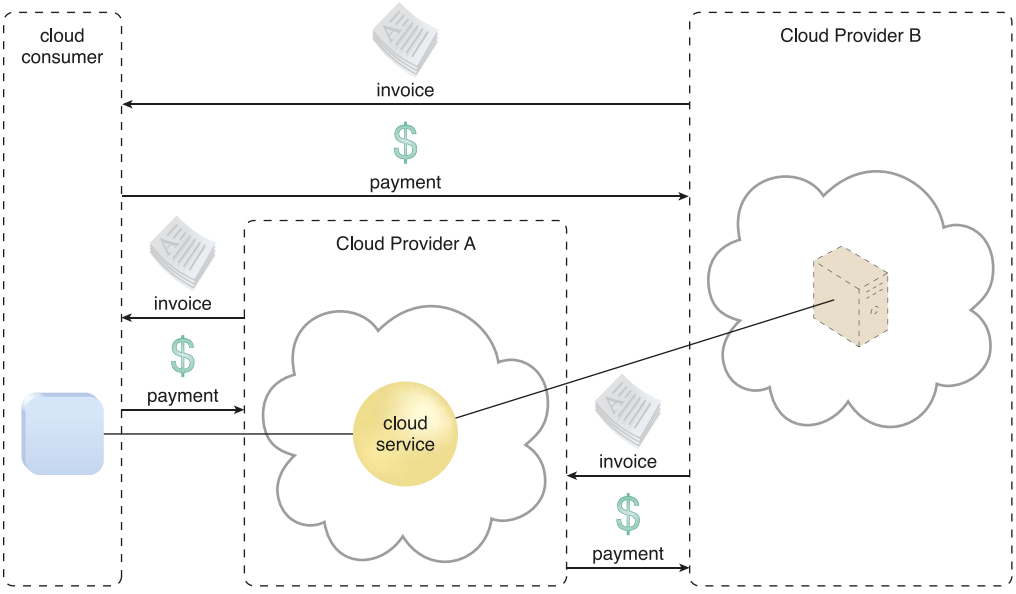


Figure 17.3

Separate pricing models are used in this scenario, whereby the cloud consumer leases a virtual server from Cloud Provider B to host the cloud service from Cloud Provider A. Both leasing agreements may have been arranged for the cloud consumer by Cloud Provider A. As part of this arrangement, there may still be some fees billed directly by Cloud Provider B to Cloud Provider A.

Multicloud Cost Management

Within a multicloud environment, it becomes important to manage the different billing, pricing, and provisioning arrangements that are established with the different cloud providers (Figure 17.4).

Some cloud providers offer reserved IT resources for which the cloud consumer may commit to paying for a fixed period of time, at a discount. Others offer the purchase of "points" or "vouchers" that are calculated to cover estimated costs, allowing for predetermined fixed monthly charges and suitable for periodic budgeting requirements frequently preferred by accounting and financial areas of organizations. A third option is billing based on spot instances that run on spurious capacity for a highly discounted price, which can be used for development or testing purposes for a very low cost. In a multicloud architecture, all of these benefits can be combined from different cloud providers, allowing the organization to select only the most convenient options.

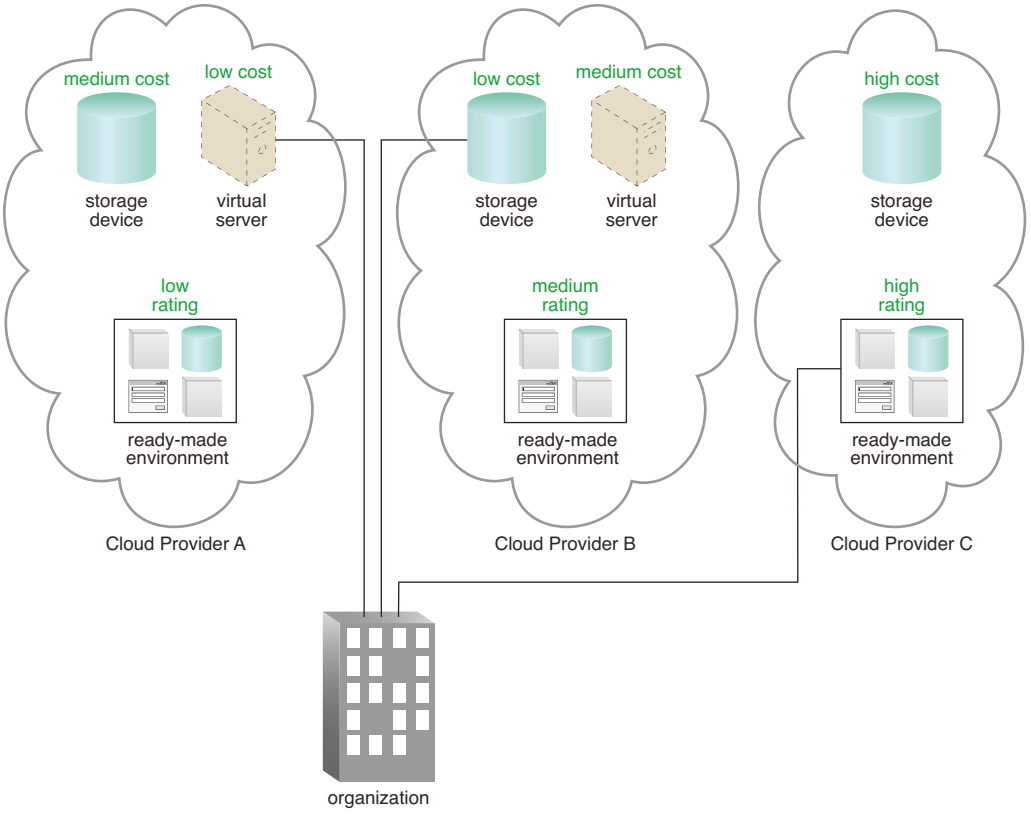


Figure 17.4

An organization using a multicloud architecture identifies and selects from each cloud provider those services that offer an optimal economic advantage.

Before migrating to the cloud, an organization must predict the expenses associated with its new IT resource provisioning. Furthermore, when considering the implementation of a multicloud architecture, specific planning for reduced or discounted expenses must be part of the process. Some of the strategies an organization can follow to achieve this are:

- *Designing a Resource Plan for Each Cloud Provider* – This plan should include specifying the true needs of the cloud consumer and enforcing them through standards that only allow the use of those resources, as well as setting budgets and expense notifications in accordance with each cloud provider's capabilities when budget thresholds are met. Supervising that the plan is completed as designed is a crucial cloud governance task.

- *Tagging Resources* – Utilizing tags enables a business to logically group the resources in its cloud environment for quick identification. It also allows the organization to determine which expenses are associated with each department or business unit. Each cloud provider has its own tagging system. Using a remote administration system, tagging can be standardized for all cloud providers in a multicloud architecture.
- *Establishing Guidelines and Rules on Resource Deployment* – Organizations should specify how, when, and by whom different types of resources are to be deployed for every different cloud provider. The kind of resources intended to be made available should also be standardized, considering the various deployment options that each cloud provider offers.

Additional Considerations

- *Negotiation* – Cloud provider pricing is often open to negotiation, especially for customers willing to commit to higher volumes or longer terms. Price negotiations can sometimes be executed online via the cloud provider's website by submitting estimated usage volumes along with proposed discounts. There are even tools available for cloud consumers to help generate accurate IT resource usage estimates for this purpose.
- *Payment Options* – After completing each measurement period, the cloud provider's billing management system calculates the amount owed by a cloud consumer. There are two common payment options available to cloud consumers: pre-payment and post-payment. With prepaid billing, cloud consumers are provided with IT resource usage credits that can be applied to future usage bills. With the post-payment method, cloud consumers are billed and invoiced for each IT resource consumption period, which is usually on a monthly basis.
- *Cost Archiving* – By tracking historical billing information, both cloud providers and cloud consumers can generate insightful reports that help identify usage and financial trends.

CASE STUDY EXAMPLE

DTGOV structures their pricing model around leasing packages for virtual servers and block-based cloud storage devices, with the assumption that resource allocation is either performed on demand or based on already reserved IT resources.

On-demand resource allocation is measured and charged back by the hour, while reserved resource allocation requires a one- to three-year commitment from the cloud consumer, with fees billed monthly.

As IT resources can scale up and down automatically, any additional capacity used is charged on a pay-per-use basis whenever a reserved IT resource is scaled beyond its allocated capacity. Windows and Linux-based virtual servers are made available in the following basic performance profiles:

- *Small Virtual Server Instance* – 1 virtual processor core, 4 GB of virtual RAM, and 320 GB of storage space in the root file system.
- *Medium Virtual Server Instance* – 2 virtual processor cores, 8 GB of virtual RAM, and 540 GB of storage space in the root file system.
- *Large Virtual Server Instance* – 8 virtual processor cores, 16 GB of virtual RAM, and 1.2 TB of storage space in the root file system.
- *Memory Large Virtual Server Instance* – 8 virtual processor cores, 64 GB of virtual RAM, and 1.2 TB of storage space in the root file system.
- *Processor Large Virtual Server Instance* – 32 virtual processor cores, 16 GB of virtual RAM, and 1.2 TB of storage space in the root file system.
- *Ultra-Large Virtual Server Instance* – 128 virtual processor cores, 512 GB of virtual RAM, and 1.2 TB of storage space in the root file system.

Virtual servers are also available in "resilient" or "clustered" formats. With the former option, the virtual servers are replicated in at least two different data centers. In the latter case, the virtual servers are run in a high-availability cluster that is implemented by the virtualization platform.

The pricing model is further based on the capacity of the cloud storage devices as expressed in multiples of 1 GB, with a minimum of 40 GB. Storage device capacity can be fixed and administratively adjusted by the cloud consumer to increase or decrease by increments of 40 GB, while the block storage has a maximum capacity of 1.2 TB. I/O transfers to and from cloud storage devices are also subject to charges in addition to pay-per-use fees applied to outbound WAN traffic. Inbound WAN and intra-cloud traffic are free of charge.

A complimentary usage allowance permits cloud consumers to lease up to three small virtual server instances and a 60 GB block-based cloud storage device, 5 GB of I/O transfers monthly, as well as 5 GB of WAN outbound traffic monthly, all in the first 90 days. As DTGOV prepares its pricing model for public release, it realizes that setting cloud service prices is more challenging than they expected because:

- Their prices need to reflect and respond to marketplace conditions while staying competitive with other cloud offerings and remaining profitable for DTGOV.
- The client portfolio has not been established yet, as DTGOV is expecting new customers. Their non-cloud clients are expected to progressively migrate to the cloud, although the actual rate of migration is too difficult to predict.

After performing further market research, DTGOV settles on the following price template for virtual server instance allocation:

Virtual Server On-Demand Instance Allocation

- Metric: on-demand instance allocation
- Measurement: pay-per-use charges calculated for total service consumption for each calendar month (an hourly rate is used for the actual instance size when the instance has been scaled up)
- Billing Period: monthly

The price template is outlined in Table 17.2.

Instance Name	Instance Size	Operating System	Hourly
Small Virtual Server Instance	1 virtual processor core 4 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$0.06
		Linux Red Hat	$0.08
		Windows	$0.09
Medium Virtual Server Instance	2 virtual processor cores 8 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$0.14
		Linux Red Hat	$0.17
		Windows	$0.19
Large Virtual Server Instance	8 virtual processor cores 16 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$0.32
		Linux Red Hat	$0.37
		Windows	$0.39
Memory Large Virtual Server Instance	8 virtual processor cores 64 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$0.89
		Linux Red Hat	$0.95
		Windows	$0.99
Processor Large Virtual Server Instance	32 virtual processor cores 16 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$0.89
		Linux Red Hat	$0.95
		Windows	$0.99
Ultra-Large Virtual Server Instance	128 virtual processor cores 512 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$1.29
		Linux Red Hat	$1.69
		Windows	$1.89

Table 17.2
The price template for virtual server on-demand instance allocation.

Surcharge for clustered IT resources: 120%

Surcharge for resilient IT resources: 150%

Virtual Server Reserved Instance Allocation

- Metric: reserved instance allocation
- Measurement: reserved instance allocation fee charged up-front with pay-per-use fees calculated based on the total consumption during each calendar month (additional charges apply for periods when the instance is scaled up)
- Billing Period: monthly

The price template is outlined in Table 17.3.

Instance Name	Instance Size	Operating System	1-Year Term Pricing		3-Year Term Pricing	
			Up-Front	Hourly	Up-Front	Hourly
Small Virtual Server Instance	1 virtual processor core 4 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$57.10	$0.032	$87.97	$0.026
		Linux Red Hat	$76.14	$0.043	$117.30	$0.034
		Windows	$85.66	$0.048	$131.96	$0.038
Medium Virtual Server Instance	2 virtual processor cores 8 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$133.24	$0.075	$205.27	$0.060
		Linux Red Hat	$161.79	$0.091	$249.26	$0.073
		Windows	$180.83	$0.102	$278.58	$0.081

continues

Instance Name	Instance Size	Operating System	1-Year Term Pricing		3-Year Term Pricing	
			Up-Front	Hourly	Up-Front	Hourly
Large Virtual Server Instance	8 virtual processor cores 16 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$304.55	$0.172	$469.19	$0.137
		Linux Red Hat	$352.14	$0.199	$542.50	$0.158
		Windows	$371.17	$0.210	$571.82	$0.167
Memory Large Virtual Server Instance	8 virtual processor cores 64 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$751.86	$0.425	$1158.30	$0.338
		Linux Red Hat	$808.97	$0.457	$1246.28	$0.363
		Windows	$847.03	$0.479	$1304.92	$0.381
Processor Large Virtual Server Instance	32 virtual processor cores 16 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$751.86	$0.425	$1158.30	$0.338
		Linux Red Hat	$808.97	$0.457	$1246.28	$0.363
		Windows	$847.03	$0.479	$1304.92	$0.381
Ultra-Large Virtual Server Instance	128 virtual processor cores 512 GB of virtual RAM 20 GB of storage	Linux Ubuntu	$1132.55	$0.640	$1744.79	$0.509
		Linux Red Hat	$1322.90	$0.748	$2038.03	$0.594
		Windows	$1418.07	$0.802	$2184.65	$0.637

Table 17.3

The price template for virtual server reserved instance allocation.

Surcharge for clustered IT resources: 100%

Surcharge for resilient IT resources: 120%

DTGOV further provides the following simplified price templates for cloud storage device allocation and WAN bandwidth usage.

Cloud Storage Device

- Metric: on-demand storage allocation, I/O data transferred
- Measurement: pay-per-use charges calculated based on total consumption during each calendar month (storage allocation calculated with per-hour granularity and cumulative I/O transfer volume)
- Billing Period: monthly
- Price Template: $0.10/GB per month of allocated storage, $0.001/GB for I/O transfers

WAN Traffic

- Metric: outbound network usage
- Measurement: pay-per-use charges calculated based on total consumption for each calendar month (WAN traffic volume calculated cumulatively)
- Billing Period: monthly
- Price Template: $0.01/GB for outbound network data

This page intentionally left blank

Chapter 18

Service Quality Metrics and SLAs

Service-level agreements (SLAs) are a focal point of negotiations, contract terms, legal obligations, and runtime metrics and measurements. SLAs formalize the guarantees put forth by cloud providers and correspondingly influence or determine the pricing models and payment terms. SLAs set cloud consumer expectations and are integral to how organizations build business automation around the utilization of cloud-based IT resources.

The guarantees made by a cloud provider to a cloud consumer are often carried forward, in that the same guarantees are made by the cloud consumer organization to its clients, business partners, or whomever will be relying on the services and solutions hosted by the cloud provider. It is therefore crucial for SLAs and related service quality metrics to be understood and aligned in support of the cloud consumer's business requirements, while also ensuring that the guarantees can, in fact, be realistically fulfilled consistently and reliably by the cloud provider. The latter consideration is especially relevant for cloud providers that host shared IT resources for high volumes of cloud consumers, each of which will have been issued its own SLA guarantees.

18.1 Service Quality Metrics

SLAs issued by cloud providers are human-readable documents that describe quality-of-service (QoS) features, guarantees, and limitations of one or more cloud-based IT resources.

SLAs use service quality metrics to express measurable QoS characteristics.

For example:

- *Availability* – uptime, outages, service duration
- *Reliability* – minimum time between failures, guaranteed rate of successful responses
- *Performance* – capacity, response time, and delivery time guarantees
- *Scalability* – capacity fluctuation and responsiveness guarantees
- *Resiliency* – mean time to switchover and recovery

SLA management systems use these metrics to perform periodic measurements that verify compliance with SLA guarantees, in addition to collecting SLA-related data for various types of statistical analyses.

Each service quality metric is ideally defined using the following characteristics:

- *Quantifiable* – The unit of measure is clearly set, absolute, and appropriate so that the metric can be based on quantitative measurements.
- *Repeatable* – The methods of measuring the metric need to yield identical results when repeated under identical conditions.
- *Comparable* – The units of measure used by a metric need to be standardized and comparable. For example, a service quality metric cannot measure smaller quantities of data in bits and larger quantities in bytes.
- *Easily Obtainable* – The metric needs to be based on a nonproprietary, common form of measurement that can be easily obtained and understood by cloud consumers.

The upcoming sections provide a series of common service quality metrics, each of which is documented with a description, unit of measure, measurement frequency, and applicable cloud delivery model values, as well as a brief example.

Service Availability Metrics

Availability Rate Metric

The overall availability of an IT resource is usually expressed as a percentage of uptime. For example, an IT resource that is always available will have an uptime of 100%.

- *Description* – percentage of service uptime
- *Measurement* – total uptime / total time
- *Frequency* – weekly, monthly, yearly
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – minimum 99.5% uptime

Availability rates are calculated cumulatively, meaning that unavailability periods are combined to compute the total downtime (Table 18.1).

Availability (%)	Downtime/Week (Seconds)	Downtime/Month (Seconds)	Downtime/Year (Seconds)
99.5	3,024	216	158,112
99.8	1,210	5,174	63,072
99.9	606	2,592	31,536
99.95	302	1,294	15,768
99.99	60.6	259.2	3,154
99.999	6.05	25.9	316.6
99.9999	0.605	2.59	31.5

Table 18.1
Sample availability rates measured in units of seconds.

Outage Duration Metric

This service quality metric is used to define both maximum and average continuous outage service-level targets.

- *Description* – duration of a single outage
- *Measurement* – date/time of outage end – date/time of outage start
- *Frequency* – per event
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 1-hour maximum, 15-minute average

NOTE

In addition to being quantitatively measured, availability can be described qualitatively using terms such as high-availability (HA), which is used to label an IT resource with exceptionally low downtime usually due to underlying resource replication and/or clustering infrastructure.

Service Reliability Metrics

A characteristic closely related to availability, reliability is the probability that an IT resource can perform its intended function under predefined conditions without experiencing failure. Reliability focuses on how often the service performs as expected, which requires the service to remain in an operational and available state. Certain reliability metrics only consider runtime errors and exception conditions as failures, which are commonly measured only when the IT resource is available.

Mean Time Between Failures (MTBF) Metric

- *Description* – expected time between consecutive service failures
- *Measurement* – Σ, normal operational period duration / number of failures
- *Frequency* – monthly, yearly
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – 90-day average

Reliability Rate Metric

Overall reliability is more complicated to measure and is usually defined by a reliability rate that represents the percentage of successful service outcomes. This metric measures the effects of nonfatal errors and failures that occur during uptime periods. For example, an IT resource's reliability is 100% if it has performed as expected every time it is invoked, but only 80% if it fails to perform every fifth time.

- *Description* – percentage of successful service outcomes under predefined conditions
- *Measurement* – total number of successful responses / total number of requests
- *Frequency* – weekly, monthly, yearly
- *Cloud Delivery Model* – SaaS
- *Example* – minimum 99.5%

Service Performance Metrics

Service performance refers to the ability of an IT resource to carry out its functions within expected parameters. This quality is measured using service capacity metrics,

each of which focuses on a related measurable characteristic of IT resource capacity. A set of common performance capacity metrics is provided in this section. Note that different metrics may apply, depending on the type of IT resource being measured.

Network Capacity Metric

- *Description* – measurable characteristics of network capacity
- *Measurement* – bandwidth / throughput in bits per second
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 10 MB per second

Storage Device Capacity Metric

- *Description* – measurable characteristics of storage device capacity
- *Measurement* – storage size in GB
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 80 GB of storage

Server Capacity Metric

- *Description* – measurable characteristics of server capacity
- *Measurement* – number of CPUs, CPU frequency in GHz, RAM size in GB, storage size in GB
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – 1 core at 1.7 GHz, 16 GB of RAM, 80 GB of storage

Web Application Capacity Metric

- *Description* – measurable characteristics of web application capacity
- *Measurement* – rate of requests per minute
- *Frequency* – continuous

- *Cloud Delivery Model* – SaaS
- *Example* – maximum 100,000 requests per minute

Instance Starting Time Metric

- *Description* – length of time required to initialize a new instance
- *Measurement* – date/time of instance up – date/time of start request
- *Frequency* – per event
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – 5-minute maximum, 3-minute average

Response Time Metric

- *Description* – time required to perform synchronous operation
- *Measurement* – (date/time of request – date/time of response) / total number of requests
- *Frequency* – daily, weekly, monthly
- *Cloud Delivery Model* – SaaS
- *Example* – 5-millisecond average

Completion Time Metric

- *Description* – time required to complete an asynchronous task
- *Measurement* – (date of request – date of response) / total number of requests
- *Frequency* – daily, weekly, monthly
- *Cloud Delivery Model* – PaaS, SaaS
- *Example* – 1-second average

Service Scalability Metrics

Service scalability metrics are related to IT resource elasticity capacity, which is related to the maximum capacity that an IT resource can achieve, as well as measurements of its ability to adapt to workload fluctuations. For example, a server can be scaled up to a maximum of 128 CPU cores and 512 GB of RAM, or scaled out to a maximum of 16 load-balanced replicated instances.

The following metrics help determine whether dynamic service demands will be met proactively or reactively, as well as the impacts of manual or automated IT resource allocation processes.

Storage Scalability (Horizontal) Metric

- *Description* – permissible storage device capacity changes in response to increased workloads
- *Measurement* – storage size in GB
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 1,000 GB maximum (automated scaling)

Server Scalability (Horizontal) Metric

- *Description* – permissible server capacity changes in response to increased workloads
- *Measurement* – number of virtual servers in resource pool
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – 1 virtual server minimum, 10 virtual server maximum (automated scaling)

Server Scalability (Vertical) Metric

- *Description* – permissible server capacity fluctuations in response to workload fluctuations
- *Measurement* – number of CPUs, RAM size in GB
- *Frequency* – continuous
- *Cloud Delivery Model* – IaaS, PaaS
- *Example* – 512 core maximum, 512 GB of RAM

Service Resiliency Metrics

The ability of an IT resource to recover from operational disturbances is often measured using service resiliency metrics. When resiliency is described within or in relation to SLA resiliency guarantees, it is often based on redundant implementations and resource replication over different physical locations as well as various disaster recovery systems.

The type of cloud delivery model determines how resiliency is implemented and measured. For example, the physical locations of replicated virtual servers that are implementing resilient cloud services can be explicitly expressed in the SLAs for IaaS environments, while being implicitly expressed for the corresponding PaaS and SaaS environments.

Resiliency metrics can be applied in three different phases to address the challenges and events that can threaten the regular level of a service:

- *Design Phase* – Metrics that measure how prepared systems and services are to cope with challenges.
- *Operational Phase* – Metrics that measure the difference in service levels before, during, and after a downtime event or service outage, which are further qualified by availability, reliability, performance, and scalability metrics.
- *Recovery Phase* – Metrics that measure the rate at which an IT resource recovers from downtime, such as the mean time for a system to log an outage and switch over to a new virtual server.

Two common metrics related to measuring resiliency are described next.

Mean Time to Switchover (MTSO) Metric

- *Description* – the time expected to complete a switchover from a severe failure to a replicated instance in a different geographical area
- *Measurement* – (date/time of switchover completion – date/time of failure) / total number of failures
- *Frequency* – monthly, yearly
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 10-minute average

Mean Time to System Recovery (MTSR) Metric

- *Description* – time expected for a resilient system to perform a complete recovery from a severe failure
- *Measurement* – (date/time of recovery – date/time of failure) / total number of failures
- *Frequency* – monthly, yearly
- *Cloud Delivery Model* – IaaS, PaaS, SaaS
- *Example* – 120-minute average

18.2 CASE STUDY EXAMPLE

After suffering a cloud outage that made their web portal unavailable for about an hour, Innovartus decides to thoroughly review the terms and conditions of their SLA. They begin by researching the cloud provider's availability guarantees, which prove to be ambiguous because they do not clearly state which events in the cloud provider's SLA management system are classified as "downtime." Innovartus also discovers that the SLA lacks reliability and resilience metrics, which had become essential to their cloud service operations.

In preparation for a renegotiation of the SLA terms with the cloud provider, Innovartus decides to compile a list of additional requirements and guarantee stipulations:

- The availability rate needs to be described in greater detail to enable more effective management of service availability conditions.
- Technical data that supports service operations models needs to be included to ensure that the operation of select critical services remains fault tolerant and resilient.
- Additional metrics that assist in service quality assessment need to be included.
- Any events that are to be excluded from what is measured with availability metrics need to be clearly defined.

After several conversations with the cloud provider's sales representative, Innovartus is offered a revised SLA with the following additions:

- The method by which the availability of cloud services is to be measured, in addition to any supporting IT resources on which ATN core processes depend.
- Inclusion of a set of reliability and performance metrics approved by Innovartus.

Six months later, Innovartus performs another SLA metrics assessment and compares the newly generated values with ones that were generated prior to the SLA improvements (Table 18.2).

SLA Metrics	Statistics of Previous SLA	Statistics of Revised SLA
Average Availability	98.10%	99.98%
High-Availability Model	Cold-Standby	Hot-Standby
Average Service Quality **based on customer satisfaction surveys*	52%	70%

Table 18.2
The evolution of Innovartus's SLA evaluation, as monitored by their cloud resource administrators.

18.3 SLA Guidelines

This section provides a number of best practices and recommendations for working with SLAs, the majority of which are applicable to cloud consumers:

- *Mapping Business Cases to SLAs* – It can be helpful to identify the necessary QoS requirements for a given automation solution and to then concretely link them to the guarantees expressed in the SLAs for the IT resources responsible for carrying out the automation. This can avoid situations where SLAs are inadvertently misaligned, or perhaps unreasonably deviate in their guarantees, subsequent to IT resource usage.

- *Working with Cloud and On-Premises SLAs* – Due to the vast infrastructure available to support IT resources in public clouds, the QoS guarantees issued in SLAs for cloud-based IT resources are generally superior to those provided for on-premises IT resources. This variance needs to be understood, especially when building hybrid distributed solutions that utilize both on-premises and cloud-based services or when incorporating cross-environment technology architectures, such as cloud bursting.
- *Understanding the Scope of an SLA* – Cloud environments consist of many supporting architectural and infrastructure layers on which IT resources reside and are integrated. It is important to acknowledge the extent to which a given IT resource guarantee applies. For example, an SLA may be limited to the IT resource implementation but not its underlying hosting environment.
- *Understanding the Scope of SLA Monitoring* – SLAs need to specify where monitoring is performed and where measurements are calculated, primarily in relation to the cloud's firewall. For example, monitoring within the cloud firewall is not always advantageous or relevant to the cloud consumer's required QoS guarantees. Even the most efficient firewalls have a measurable degree of influence on performance and can further present a point of failure.
- *Documenting Guarantees at the Appropriate Granularity* – SLA templates used by cloud providers sometimes define guarantees in broad terms. If a cloud consumer has specific requirements, the corresponding level of detail should be used to describe the guarantees. For example, if data replication needs to take place across particular geographic locations, then these need to be specified directly within the SLA.
- *Defining Penalties for Noncompliance* – If a cloud provider is unable to follow through on the QoS guarantees promised within the SLAs, recourse can be formally documented in terms of compensation, penalties, reimbursements, or otherwise.
- *Incorporating Nonmeasurable Requirements* – Some guarantees cannot be easily measured using service quality metrics, but are relevant to QoS nonetheless, and should therefore still be documented within the SLA. For example, a cloud consumer may have specific security and privacy requirements for data hosted by the cloud provider that can be addressed by assurances in the SLA for the cloud storage device being leased.

- *Disclosure of Compliance Verification and Management* – Cloud providers are often responsible for monitoring IT resources to ensure compliance with their own SLAs. In this case, the SLAs themselves should state which tools and practices are being used to carry out the compliance checking process, in addition to any legal-related auditing that may be occurring.
- *Inclusion of Specific Metric Formulas* – Some cloud providers do not mention common SLA metrics or the metrics-related calculations in their SLAs, instead focusing on service-level descriptions that highlight the use of best practices and customer support. Metrics being used to measure SLAs should be part of the SLA document, including the formulas and calculations that the metrics are based upon.
- *Considering Independent SLA Monitoring* – Although cloud providers will often have sophisticated SLA management systems and SLA monitors, it may be in the best interest of a cloud consumer to hire a third-party organization to perform independent monitoring as well, especially if there are suspicions that SLA guarantees are not always being met by the cloud provider (despite the results shown on periodically issued monitoring reports).
- *Archiving SLA Data* – The SLA-related statistics collected by SLA monitors are commonly stored and archived by the cloud provider for future reporting purposes. If a cloud provider intends to keep SLA data specific to a cloud consumer even after the cloud consumer no longer continues its business relationship with the cloud provider, then this should be disclosed. The cloud consumer may have data privacy requirements that disallow the unauthorized storage of this type of information. Similarly, during and after a cloud consumer's engagement with a cloud provider, it may want to keep a copy of historical SLA-related data as well. It may be especially useful for comparing cloud providers in the future.
- *Disclosing Cross-Cloud Dependencies* – Cloud providers may be leasing IT resources from other cloud providers, which results in a loss of control over the guarantees they are able to make to cloud consumers. Although a cloud provider will rely on the SLA assurances made to it by other cloud providers, the cloud consumer may want disclosure of the fact that the IT resources it is leasing may have dependencies beyond the environment of the cloud provider organization that it is leasing them from.

18.4 CASE STUDY EXAMPLE

DTGOV begins its SLA template authoring process by working with a legal advisory team that has been adamant about an approach whereby cloud consumers are presented with an online web page outlining the SLA guarantees, along with a "click once to accept" button. The default agreement contains extensive limitations to DTGOV's liability in relation to possible SLA noncompliance, as follows:

- The SLA defines guarantees only for service availability.
- Service availability is defined for all the cloud services simultaneously.
- Service availability metrics are loosely defined to establish a level of flexibility regarding unexpected outages.
- The terms and conditions are linked to the Cloud Services Customer Agreement, which is accepted implicitly by all the cloud consumers that use the self-service portal.
- Extended periods of unavailability are to be recompensed by monetary "service credits," which are to be discounted on future invoices and have no actual monetary value.

Provided here are key excerpts from DTGOV's SLA template.

Scope and Applicability

This Service-Level Agreement ("SLA") establishes the service quality parameters that are to be applied to the use of DTGOV's cloud services ("DTGOV cloud") and is part of the DTGOV Cloud Services Customer Agreement ("DTGOV Cloud Agreement").

The terms and conditions specified in this agreement apply solely to virtual server and cloud storage device services, herein called "Covered Services." This SLA applies separately to each cloud consumer ("Consumer") that is using the DTGOV Cloud. DTGOV reserves the right to change the terms of this SLA in accordance with the DTGOV Cloud Agreement at any time.

Service Quality Guarantees

The Covered Services will be operational and available to Consumers at least 99.95% of the time in any calendar month. If DTGOV does not meet this SLA requirement while the Consumer succeeds in meeting its SLA obligations, the Consumer will be

eligible to receive Financial Credits as compensation. This SLA states the Consumer's exclusive right to compensation for any failure on DTGOV's part to fulfill the SLA requirements.

Definitions

The following definitions are to be applied to DTGOV's SLA:

- "Unavailability" is defined as the entirety of the Consumer's running instances as having no external connectivity for a duration that is at least five consecutive minutes in length, during which the Consumer is unable to launch commands against the remote administration system through either the web application or web service API.
- "Downtime Period" is defined as a period of five or more consecutive minutes of the service remaining in a state of Unavailability. Periods of "Intermittent Downtime" that are less than five minutes long do not count toward Downtime Periods.
- "Monthly Uptime Percentage" (MUP) is calculated as: (total number of minutes in a month – total number of downtime period minutes in a month) / (total number of minutes in a month)
- "Financial Credit" is defined as the percentage of the monthly invoice total that is credited toward future monthly invoices of the Consumer, which is calculated as follows:

  99.00% < MUP % < 99.95% – 10% of the monthly invoice is credited in favor of the Consumer's invoice

  89.00% < MUP % < 99.00% – 30% of the monthly invoice is credited in favor of the Consumer's invoice

  MUP % < 89.00% – 100% of the monthly invoice is credited in favor of the Consumer's invoice

Usage of Financial Credits

The MUP for each billing period is to be displayed on each monthly invoice. The Consumer is to submit a request for Financial Credit to be eligible to redeem Financial Credits. For that purpose, the Consumer is to notify DTGOV within thirty days

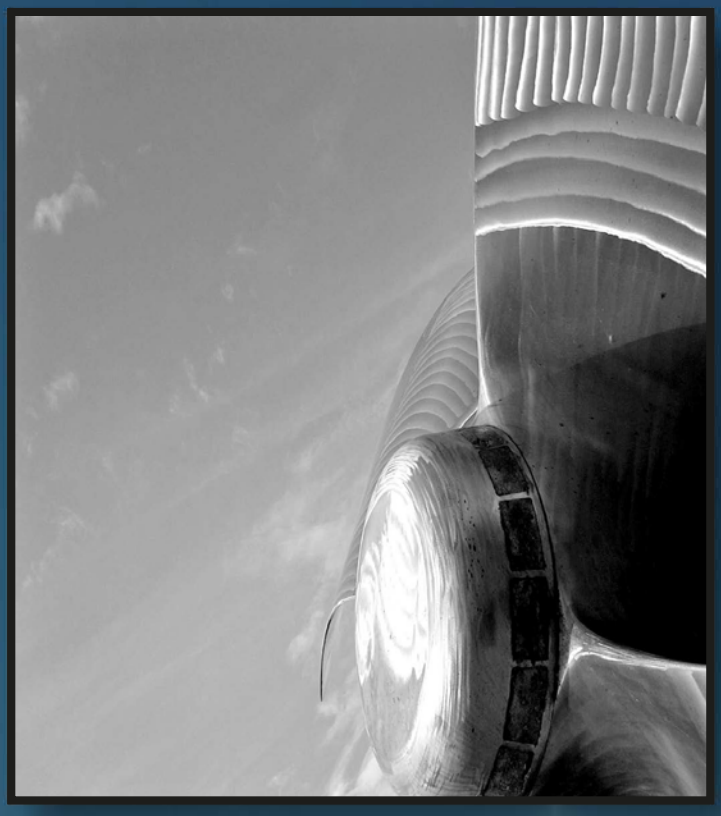
from the time the Consumer receives the invoice that states the MUP beneath the defined SLA. Notification is to be sent to DTGOV via email. Failure to comply with this requirement forfeits the Consumer's right to the redemption of Financial Credits.

SLA Exclusions

The SLA does not apply to any of the following:

- Unavailability periods caused by factors that cannot be reasonably foreseen or prevented by DTGOV.
- Unavailability periods resulting from the malfunctioning of the Consumer's software and/or hardware, third-party software and/or hardware, or both.
- Unavailability periods resulting from abuse or detrimental behavior and actions that are in violation of the DTGOV Cloud Agreement.
- Consumers with overdue invoices or that are otherwise not considered in good standing with DTGOV.

Part V



Appendices

Appendix A Case Study Conclusions

Appendix B Common Containerization Technologies

This page intentionally left blank

Appendix A

Case Study Conclusions

A.1 ATN

A.2 DTGOV

A.3 Innovartus

This appendix briefly concludes the storylines of the three case studies that were first introduced in Chapter 2.

A.1 ATN

The cloud initiative necessitated migrating selected applications and IT services to the cloud, allowing for the consolidation and retirement of solutions in a crowded application portfolio. Not all the applications could be migrated, and selecting appropriate applications was a major issue. Some of the chosen applications required significant re-development effort to adapt to the new cloud environment.

Costs were effectively reduced for most of the applications that were moved to the cloud. This was discovered after six months of expenditures were compared with the costs of the traditional applications over a three-year period. Both capital and operational expenses were used in the ROI evaluation.

ATN's level of service has improved in business areas that use cloud-based applications. In the past, most of these applications showed a noticeable performance deterioration during peak usage periods. The cloud-based applications can now scale out whenever a peak workload arises.

ATN is currently evaluating other applications for potential cloud migration.

A.2 DTGOV

Although DTGOV has been outsourcing IT resources for public sector organizations for more than 30 years, establishing the cloud and its associated IT infrastructure was a major undertaking that took over two years. DTGOV now offers IaaS services to the government sector and is building a new cloud service portfolio that targets private sector organizations.

Diversification of its client and service portfolios is the next logical step for DTGOV, after all the changes they made to their technology architecture to produce a mature cloud. Before proceeding with this next phase, DTGOV produces a report to document aspects of its completed transition to cloud adoption. A summary of the report is documented in Table A.1.

Pre-Cloud Status	Required Change	Business Benefit	Challenges
The data center and related IT resources were not completely standardized.	The standardization of IT resources, including servers, storage systems, network devices, virtualization platform, and management systems.	Required investment costs are reduced by making bulk IT infrastructure acquisitions. Operational costs are reduced by optimizing the IT infrastructure.	Establishing new practices for IT procurement, technology lifecycle management, and data center management.
IT resources were deployed reactively due to long-term client commitment.	Deployment of IT resources supported by infrastructure with large-scale computing capacity.	Investments are reduced by making bulk IT infrastructure acquisitions and scaling IT resources to client demands.	Capacity planning and related ROI calculations are challenging tasks that require on-going training.
IT resources were provisioned through long-term commitment contracts.	Flexible allocation, reallocation, release, and control of available IT resources by comprehensively applying virtualization.	Cloud service provisioning is agile and on-demand for clients, and carried out via flexible (software-based) allocation and management of IT resources.	Establishing the virtualization platform related to IT resource provisioning.
Monitoring capabilities were basic.	Detailed monitoring of cloud service usage and QoS.	Service provisioning is on-demand and pay-per-use for clients. Service charges are proportional to actual IT resource consumption. Service quality management uses business-relevant SLAs.	Establishing SLA monitors, billing monitors, and management mechanisms, which were all new to DTGOV's architecture.

continues

Pre-Cloud Status	Required Change	Business Benefit	Challenges
The resiliency of the overall IT architecture was basic.	Enhanced resiliency of IT architecture, with fully interconnected data centers and cooperative IT resource allocation and management.	Computational resiliency is improved for clients.	Governance and management efforts to regulate and administer large-scale resiliency are significant.
Outsourcing contracts and related provisions were followed on a "per-contract" and "per-client" basis.	New pricing and SLA contracts for cloud service provisioning.	Rapid (agile), on-demand, and scalable services (computational capacity) for clients.	Negotiating contracts with existing clients in the new cloud-based contracting model.

Table A.1
The results of an analysis of DTGOV's cloud initiative.

A.3 Innovartus

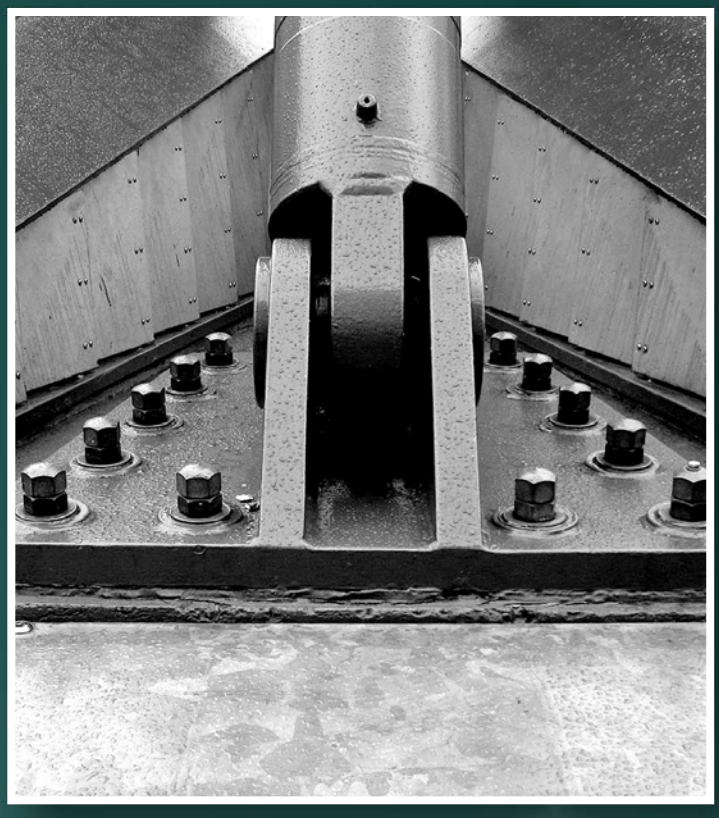
The business objective of increasing company growth required the original cloud to undergo major modifications, since they needed to move from their regional cloud provider to a large-scale global cloud provider. Portability issues were discovered only after the move, and a new cloud provider procurement process had to be created when the regional cloud provider was unable to meet all of their needs. Data recovery, application migration, and interoperability issues were also addressed.

Highly available computing IT resources and the pay-per-use feature were key in developing Innovartus' business feasibility, since access to funding and investment resources were not initially available.

Innovartus has defined several business goals they plan to achieve over the next couple of years:

- Additional applications will be migrated to different clouds, using multiple cloud providers to improve resiliency and reduce dependency on individual cloud provider vendors.
- A new mobile-only business area is to be created, since mobile access to their cloud services has experienced 20% growth.
- The application platform developed by Innovartus is being evaluated as a value-added PaaS to be offered to companies that require enhanced and innovative UI-centric features for both web-based and mobile application development.

This page intentionally left blank



Appendix B

Common Containerization Technologies

As a supplement to Chapter 6, the appendix explores the Docker container engine and the Kubernetes containerization platform and further explains how they are commonly utilized. This content helps illustrate how the terms, concepts, and technologies described in Chapter 6 exist in real-world environments.

Note the following:

- A Kubernetes platform needs to be deployed on a host cluster. The Docker container engine needs to be separately deployed on each host in that cluster.
- Both Docker and Kubernetes introduce distinct terminology. Wherever applicable, the terms established in Chapter 6 are referenced in the upcoming sections. Often, they are shown in parentheses next to the corresponding Docker or Kubernetes terms.

B.1 Docker

Docker is the first containerization engine to have become widely popular in the industry. Docker containers, also known as Dockers, introduce many important benefits and features, which will be covered in the following sections.

From an architecture perspective, a Docker container solution can be divided into the following four key areas:

- Docker Server
- Docker Client
- Docker Registry
- Docker Objects

Docker Server

A *Docker server*, also known as a Docker host, is a host that is running a Docker containerization engine. From a technology architecture perspective, a Docker host is a physical server or virtual machine running a Windows or Linux operating system. A Docker

server can be installed on any Windows or Linux machine supporting X86-64 or ARM and a few other CPU architectures. A Docker container engine can be installed on any system capable of running these operating systems.

Docker servers provide the following key features to a containerization solution:

- Docker servers provide the key functional services required to run the containerization solution and to containerize applications. These services are provided by the *Docker daemon*, which is the containerization engine in the Docker container solution. The Docker daemon has many different components and subsystems designed and deployed as part of each version of the Docker software. It is responsible for scheduling, restarting, and shutting down containers, as well as managing any container interaction.
- Docker servers host the containers that host the applications. Each container is deployed on a container host. A solution may have one or more Docker servers hosting containers and their applications.
- Docker servers also host the images used by the containers that they host, which allows different containers to use and share the same base image without the need to deploy multiple images for multiple containers.

Docker Client

A *Docker client* is a component that runs different tools that enable users and service consumers to interact with the Docker server and its services.

Docker container solutions support the following two types of clients:

- Application Programming Interface (API)
- Command Line Interface (CLI)

A Docker container does not provide a graphical user interface (GUI) for interacting with or configuring services. This reduces its footprint because it does not require heavy GUIs to be rendered and made available for users to interact with. This also results in less code required to be maintained and managed, which further reduces the security risk of the containerization solution.

As shown in Figure B.1, the Docker client can be used to interact with the Docker server or Docker host to deploy, maintain, and manage containers and their applications. All the interactions shown in the diagram occur through the use of the Docker daemon service, which acts as the container engine in Docker solutions. The Docker daemon

controls access to the containers and provides a way for clients to interact with each container.

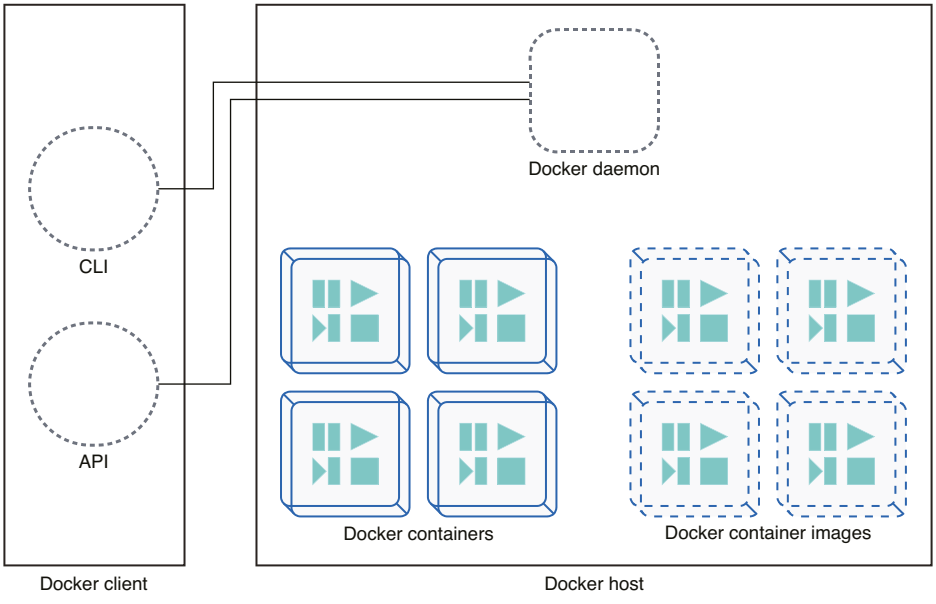


Figure B.1

The Docker client uses an API or CLI to interact with containers via the Docker daemon service.

The Docker daemon service provides REST-based APIs that the Docker client can consume as an API client. The purpose of these APIs is to provide standard interfaces through which the service consumer can interact with the Docker engine to deploy and manage their containers.

A Docker client can be run on a variety of operating systems, including Windows, Linux, and Mac.

Docker Registry

A *Docker registry* is a repository (image registry) that is used to store different types of Docker images, which are used by the Docker host to deploy containers (Figure B.2). The Docker registry is able to host and deploy multiple container images, as well as different versions of the same images. This allows application owners and system administrators to decide what container image or container image version to use when deploying containers and their applications.

The separation of the registry from the host and the Docker daemon service (container engine) has further made it possible for Docker containers to provide a repository of different images without increasing any load or storage footprint on the Docker host.

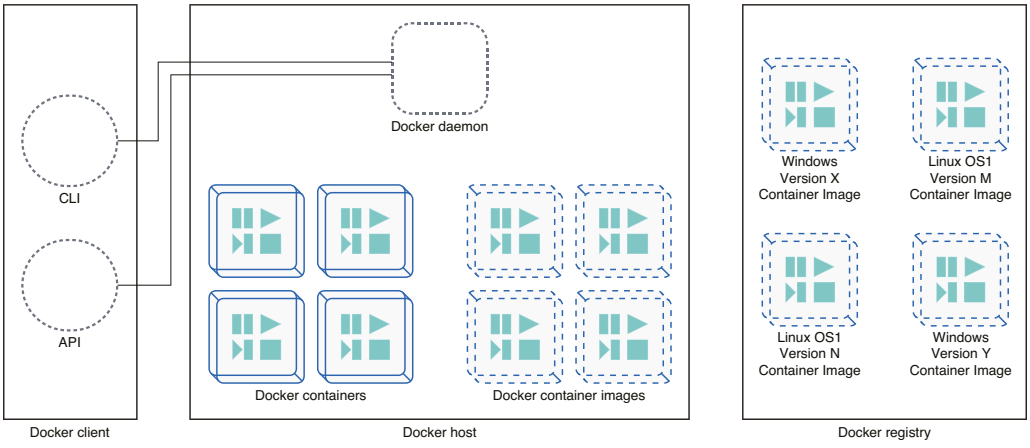


Figure B.2

The Docker registry hosts the container images that can be used to deploy Docker containers.

Docker provides a public repository of different images based on standard operating systems, such as Windows and Linux. This public repository is also known as Docker hub, and the images within the repository can be used to deploy containers. If the standard Docker images provided by the Docker hub are being used, it is not necessary to deploy the Docker registry or allocate any additional storage to build a registry.

Docker further allows users to have private Docker registries. For instance, due to security requirements, Docker images may need to be kept internal to an organization and not accessible to anyone outside of the organization. In this case, a private Docker registry can be deployed to house the Docker images that will be configured and used for the containerization solution.

Docker provides the following three key commands:

- *Docker Push* – used to add an image to the registry
- *Docker Pull* – used to download an image from the Docker registry to run a container
- *Docker Run* – used to run and start the container using a specific image

Docker Objects

A Docker container solution can have several different subcomponents and key elements collectively referred to as *Docker objects*. This section introduces the following key Docker objects:

- *Docker Container* – A *Docker container* is an instance of a container image and represents the actual container that will host the application. A container can be stopped, started, deleted, or scheduled to run at a specific time.
- *Docker Images* – *Images* are read-only templates that are created and deployed in the Docker registry and can be used to develop and run containers. For example, a base image can be created for the Linux operating system and can then be used to deploy several different containers. Each container can then make specific changes on top of the base image to make the container environment suitable for different applications. However, the containers cannot modify the base image.
- *Services* – Docker containers introduce and use many different *services* to run the Docker container solution, many of which are internal to the Docker engine and are not accessible for direct interaction. The most critical Docker services are the Docker daemon (container engine) and swarm (container orchestrator).
- *Namespaces* – To provide the capability of hosting several different containers on the same Docker host, the Docker containerization engine requires a means of isolating containers from each other to provide a secure environment where multiple containers can be deployed. Docker uses a technology called *namespaces* to provide secure isolation of containers from each other. This further enables Docker containers to securely isolate the processes in network interfaces and many other elements of containers from each other, even though they are hosted on the same Docker host.
- *Docker Control Groups* – To ensure that Docker containers use certain system resources to run and become operational and functional, the Docker host and Docker daemon need to enable the container to access the resources provided by the Docker host. This is established through the *control groups*, which limit an application deployed inside a container to a specific set of Docker host resources.
- *Union File System (container image layers)* – *Union file systems*, also referred to as unionFS, are the file systems that enable a Docker container engine to create lightweight and fast writeable layers on top of the base container image to create

a writable environment for applications and containers to make their own specific configurations, as required. This allows the container engine to operate as expected without the need to modify the base image.

- *Docker Orchestrator (container orchestrator)* – Docker containers provide their own embedded orchestration components that can be used to orchestrate the container solution to improve its productivity and automate repeatable tasks that it is required to perform. The *Docker orchestrator* is embedded in the Docker container engine and can be used by system administrators or application developers to orchestrate tasks.

Docker Swarm (Container Orchestrator)

Although deploying multiple different containers on the same Docker host can introduce many benefits in terms of saving and sharing resources among containers and applications, it can also introduce risk whereby if the host is lost, the applications and containers hosting those applications will also be lost. To prevent this issue, Docker containers can use Docker Swarm.

Docker Swarm is a container orchestrator that is deployed as part of a Docker container solution. The Docker Swarm functionality is controlled by the swarm service, a key Docker container service that is used to create and manage a cluster of Docker hosts, also referred to as a swarm, to balance the load across different physical hosts. It can also be used to improve the availability of the Docker containers, allowing application owners to deploy multiple instances of the same container on different Docker hosts while the cluster is managed by Docker Swarm. Each cluster of Docker container hosts inside the swarm is managed by a service known as the cluster manager.

Figure B.3 shows the logical architecture of Docker Swarm.

Docker container solutions can be deployed on private systems in private data centers or servers, or on different cloud platforms from public cloud providers that provide containers as a service with the use of Docker, including Amazon Web Services, Microsoft Azure, and Google Cloud.

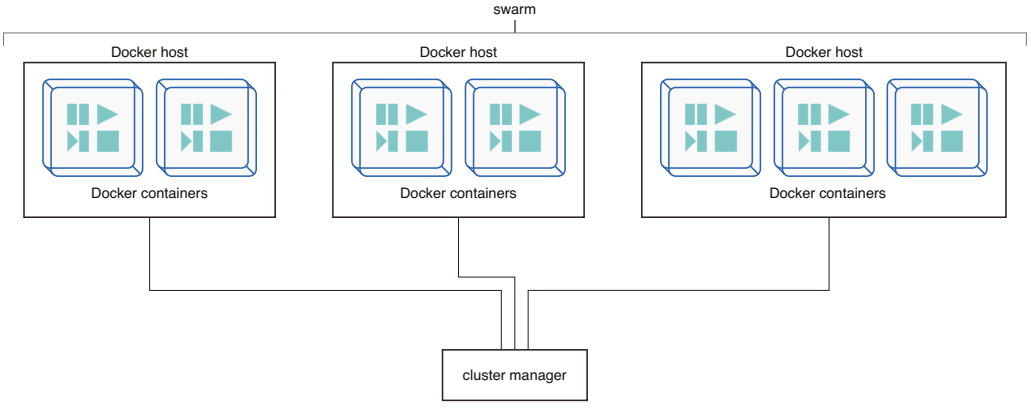


Figure B.3
The logical view of a swarm cluster comprised of three Docker hosts.

B.2 Kubernetes

Kubernetes, also known as K8s, is an open source container orchestrator that provides key benefits and features that enhance Docker. Kubernetes introduces the concept of clusters that can span several different hosts. This system takes the container functionality provided by a containerization engine like Docker to the next level by providing an enterprise-grade architecture and construct that is more suitable for enterprise-grade applications and larger scale distributed systems, which is well-suited for complex applications. This section introduces the key components of a Kubernetes solution.

Kubernetes Node (Host)

In a Kubernetes architecture running Kubernetes software to host its containers, a *Kubernetes node* is the equivalent to the containerization host or Docker host discussed in the preceding *Docker* section. Each Kubernetes cluster (host cluster) can have one or more nodes.

Figure B.4 shows a Kubernetes node, also known as a Kubernetes host, as the key component of a Kubernetes solution.

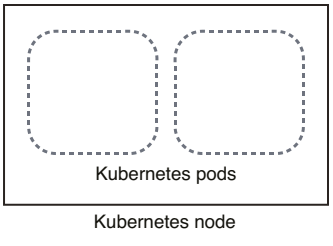


Figure B.4
The Kubernetes node is used to host containers, in this case two instances of Kubernetes pods.

Each Kubernetes node (host) has three key components that enable Kubernetes to host containers in pods, which will be explained later in this section:

- Kubelet
- Kube-Proxy
- Container Runtime

Kubernetes Pod

A *Kubernetes pod* is a logical boundary or logical group of different containers that share storage and network resources on the same Kubernetes node. The containers deployed in each pod also share the same configuration and specifications regarding how to run them. For example, every container hosted inside a pod will always be hosted together on the same Kubernetes node in the cluster.

Figure B.5 shows a logical view of a pod and how the pod is used for the logical separation of containers on the same host.

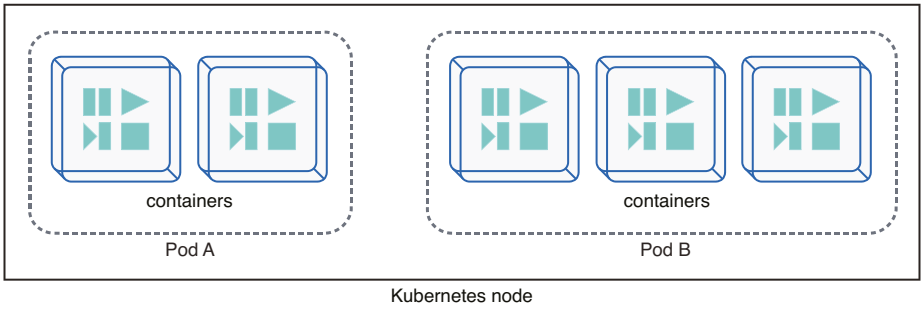


Figure B.5
Pods can be used to logically group and isolate a set of containers from other containers.

Kubelet

A *kubelet* is a service agent that is deployed on each node inside a cluster. It is responsible for ensuring that containers configured to run in each pod are operational and running as expected.

Kube-Proxy

A *kube-proxy* is a service that runs in each Kubernetes node. It acts as a service proxy that enables the containers deployed inside a pod to access the network resources and further communicate with the external world. Each kube-proxy maintains network rules on nodes. These network rules can be defined by system administrators and are used to allow internal and external network communication to each pod in a Kubernetes cluster.

Figure B.6 shows the concept of the Kubernetes node and its kubelet and kube-proxy components.

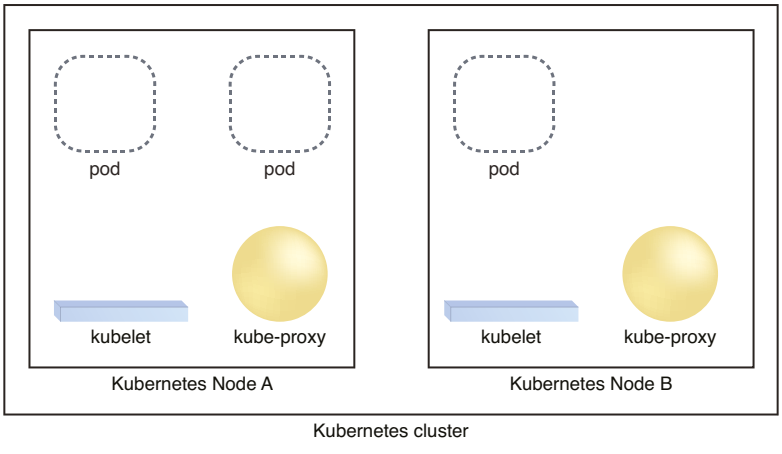


Figure B.6

The Kubernetes cluster is comprised of two nodes: Kubernetes Node A and Kubernetes Node B. Each node has its own kubelet and kube-proxy to serve its pods.

Container Runtime (Container Engine)

In a Kubernetes architecture, the container engine (referred to as the *container runtime*) enables a solution to leverage the Kubernetes technology architecture and its features to deploy a variety of containers. In addition to supporting different container runtimes, Kubernetes also offers its own, known as the *container runtime interface (CRI)*. It is similar to a Docker container engine. As an alternative to using CRI to host containers, a Docker container engine can be deployed to host Docker containers on top of Kubernetes nodes (Figures B.7 and B.8).

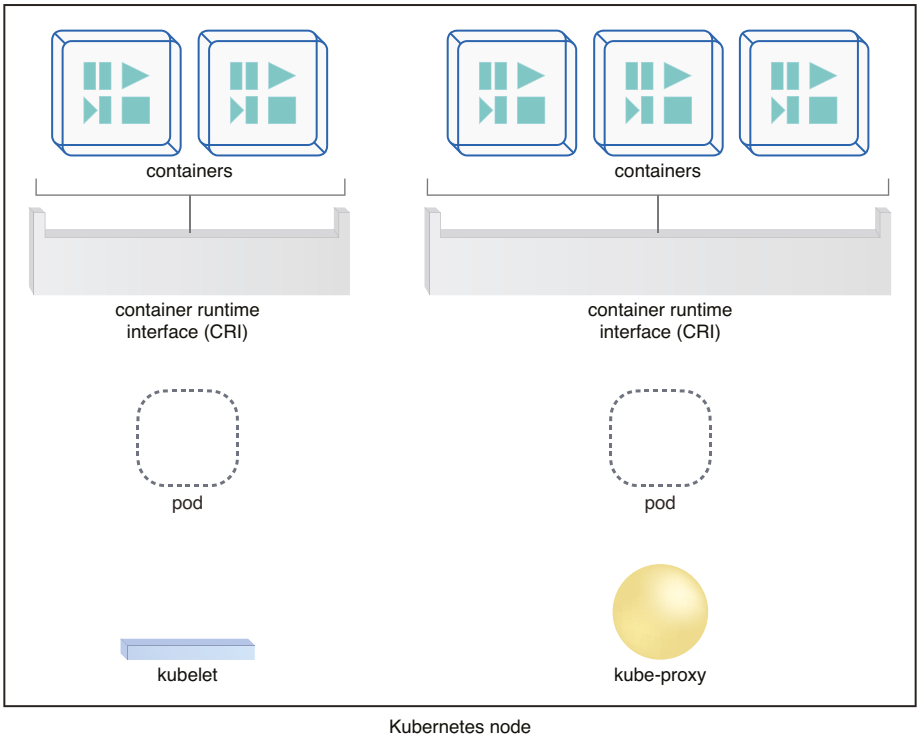


Figure B.7
A Kubernetes node hosting containers using CRI as a container engine.

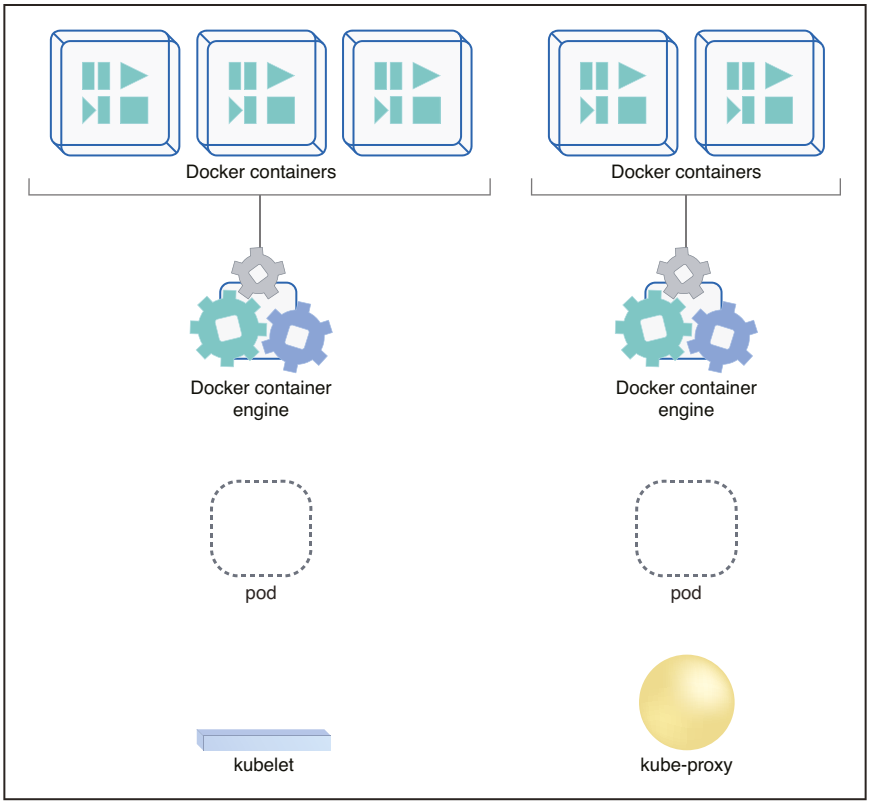


Figure B.8

A Kubernetes node running a Docker container engine at runtime to offer containers.

Cluster

In a Kubernetes architecture, a *cluster* is a group of nodes that work together to provide highly scalable and available solutions for deploying containers to host applications. As shown in Figure B.9, a cluster can contain several different nodes.

Unlike Docker containers, which introduce the concept of a swarm for grouping multiple different Docker container hosts, Kubernetes introduces a much more comprehensive concept and technology architecture for creating a cluster of containerization hosts that is suitable for enterprise applications.

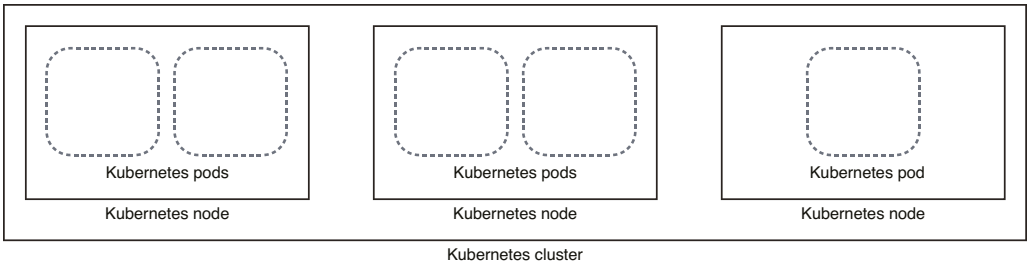


Figure B.9
An example of a Kubernetes cluster containing three different nodes.

Kubernetes Control Plane

A *control plane* can be used in a Kubernetes cluster architecture to offer better services and more capabilities that enable application owners and system administrators to utilize a containerization solution to its full potential. The control plane is responsible for making decisions that are applicable to the entire cluster, granting system administrators or application owners a common set of tools for managing the nodes in a cluster. This section introduces the key components of a control plane within a Kubernetes cluster.

- *Kubernetes API* – The *Kubernetes API* provides a method for application owners, system administrators, and developers to interact with the Kubernetes architecture, its nodes, and the containers deployed in each Kubernetes cluster.
- *kube-apiserver* – The *kube-apiserver* exposes Kubernetes APIs to service consumers so they can interact with the Kubernetes cluster and its components, as well as the containers deployed inside the cluster, through the API. The *kube-apiserver* is deployed as an independent component that can be horizontally scaled to handle a greater volume of API calls and requests from service consumers to accommodate the performance required by a solution as it scales.
- *etcd* – The *etcd* service is used to store the configuration of the cluster data inside the control plane. This does not include any user data or application data from the containerized application.
- *kube-scheduler* – The *kube-scheduler* is responsible for scheduling and running containers. Each time a new container is deployed, the kube-scheduler checks the resource utilization of the nodes inside the clusters, as well as the different pods

deployed on each node, to identify the best place to schedule and run the new container.

- *kube-controller-manager* – The *kube-controller-manager* component is responsible for running and managing the control plane processes. In the context of a Kubernetes solution, each of the preceding control plane components will run as their own independent and separate process. The kube-controller-manager provides a simple way of managing all the preceding components and their associated processes from a central point of view, thus offering system administrators a way to manage the control plane of a Kubernetes solution.
- *cloud-controller-manager* – The *cloud-controller-manager* component comes into play when a solution is deployed in a public cloud or any type of cloud that allows the cloud provider's APIs to be accessed. For example, if a Kubernetes solution is deployed on Amazon Web Services, Microsoft Azure, or Google Cloud, then this component exposes those specific environments' APIs to the cluster. However, if the solution is not deployed in a cloud environment, then this component is not required.

Figure B.10 shows an example of an overall Kubernetes deployment architecture.

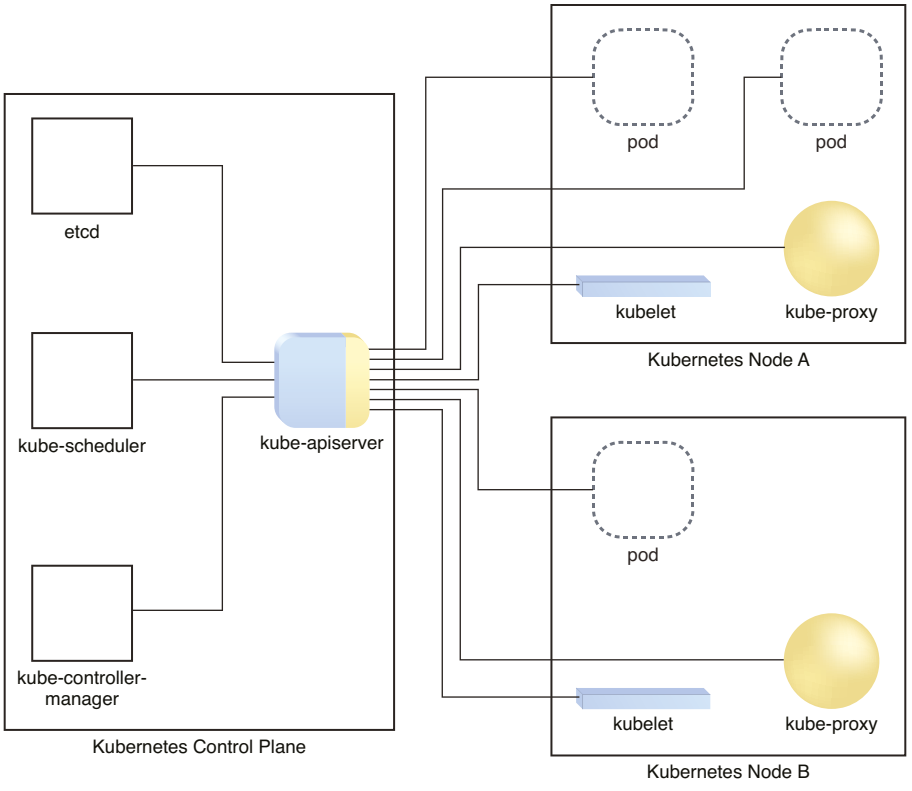


Figure B.10
A Kubernetes cluster with two nodes and a control plane, including the control plane's components.

As illustrated in the previous figure, the cluster control plane is deployed on a separate server than the Kubernetes nodes. This is done with the purpose of eliminating any interdependencies regarding the availability of the components required for managing the cluster, as well as to ensure that the control plane is not impacted if a node fails.

This page intentionally left blank

Index

B

C

D

K

L

M

N

S

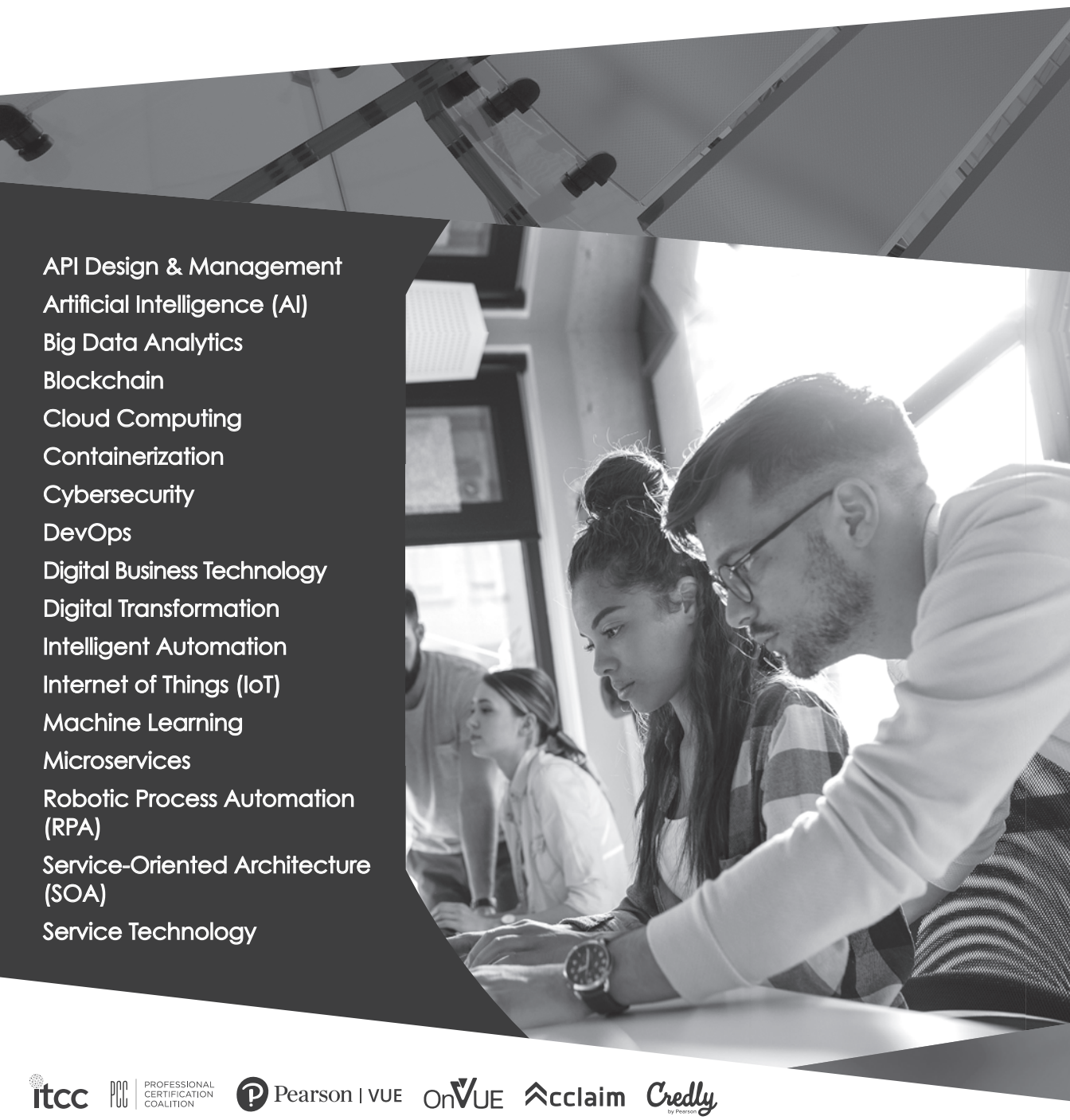
T

U

V

Arcitura®
the IT education company

eLearning Self-Study
Virtual Coaching
Instructor-Led Training



- API Design & Management
- Artificial Intelligence (AI)
- Big Data Analytics
- Blockchain
- Cloud Computing
- Containerization
- Cybersecurity
- DevOps
- Digital Business Technology
- Digital Transformation
- Intelligent Automation
- Internet of Things (IoT)
- Machine Learning
- Microservices
- Robotic Process Automation (RPA)
- Service-Oriented Architecture (SOA)
- Service Technology

itcc | PCC PROFESSIONAL CERTIFICATION COALITION | Pearson | VUE | OnVUE | acclaim | Credly by Pearson

www.arcitura.com