Developing and Hosting Applications on the Cloud

Alex Amies, Harm Sluiman,
Qiang Guo Tong, Guo Ning Liu

Developing and Hosting Applications on the Cloud

This page intentionally left blank

Developing and Hosting Applications on the Cloud

Alex Amies, Harm Sluiman, Qiang Guo Tong, Guo Ning Liu

IBM Press
Pearson plc
Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Ibmpressbooks.com

IBM Press Program Managers: Steven M. Stansel, Ellice Uffer
Cover design: IBM Corporation

Editor-in-Chief: Dave Dusthimer
Marketing Manager: Stephane Nakib
Acquisitions Editor: Mary Beth Ray
Publicist: Heather Fox
Managing Editor: Kristy Hart
Designer: Alan Clements
Project Editor: Betsy Harris
Copy Editor: Krista Hansing Editorial Services, Inc.
Senior Indexer: Cheryl Lenser
Compositor: Nonie Ratcliff
Proofreader: Language Logistics, LLC
Manufacturing Buyer: Dan Uhrig

Published by Pearson plc
Publishing as IBM Press

ISBN-13: 978-0-13-306684-5
ISBN-10: 0-13-306684-3

This book is dedicated to all the members of the IBM® SmartCloud™ Enterprise development team whose hard work and professionalism has made this large and challenging project a reality.

Contents

Preface

We are writing this book to share our experience over the past several years of developing the IBM SmartCloud™ Enterprise. We hope that readers will not just learn more about that cloud, but also be inspired to build solutions using it or other clouds as a platform. We hope that people using other clouds will benefit from this book as well.

This page intentionally left blank

Acknowledgments

Thanks to many dedicated colleagues at IBM who have worked on IBM SmartCloud Enterprise and other related products and projects. In particular, thanks to all the customers and people inside IBM who are using the IBM SmartCloud Enterprise, for their feedback and questions, especially the Rational® team. We gained a great deal of insight about the use of the cloud from these questions and discussions, and it forced us to look at the cloud from an outside-in point of view.

Thanks also to the entire IBM SmartCloud development team for its hard work and dedication in building this wonderful platform, working through unreasonable schedules and difficult technical problems in the process.

Thanks to these specific people who helped with suggestions and review:

- Chris Roach, Program Manager, Cloud Technology, IBM
- Doug Davis, Senior Technical Staff Member, Web Services and Cloud Standards, IBM
- Dikran Meliksetian, Senior Technical Staff Member, Integrated Technology Delivery, IBM
- Jamshid Vayghan, PhD, IBM Distinguished Engineer and Director, CTO Sales Transformation, IBM
- Michael Behrendt, Cloud Computing Architect, IBM
- Prasad Saripalli, PhD, Principal Architect, IBM Cloud Engineering
- Scott Peddle, Advisory Software Engineer, IBM Global Technology Services®
- Shane Weeden, Senior Software Engineer and IBM Tivoli® Federated Identity Manager development lead, who helped us understand OAuth and FIM.
- Stefan Pappe, IBM Fellow, Cloud Services Specialty Area, IBM

This was a personal effort by the authors and is not representative of IBM or its views. IBM did not participate in and does not endorse this work. However, the authors thank IBM for access to the IBM SmartCloud Enterprise system and the opportunity to work on such a challenging and satisfying project.

About the Authors

Alex Amies is a Senior Software Engineer with IBM and an architect on the IBM SmartCloud Enterprise development team.

Harm Sluiman is a Distinguished Engineer with IBM and the technical lead for SmartCloud Enterprise.

Qiang Guo Tong is an Advisory Software Engineer with IBM and one of the lead developers for SmartCloud Enterprise.

Guo Ning Liu is a Staff Software Engineer with IBM and worked on development of the public APIs, provisioning services, and security for SmartCloud Enterprise.

This page intentionally left blank

Introduction

The goal of this book is to help enterprises develop and operate services on the cloud. In particular, we hope that independent software vendors will be inspired to build value-add services on public clouds. Additionally, we hope that developers of applications who make heavy use of Infrastructure as a Service (IaaS), such as developers of Platform as a Service, Software as a Service, and Business as a Service, will find this book useful. The target audience is developers who use cloud-management application programming, architects who are planning projects, and others who want to automate the management of IT infrastructure. The book is intermediate in level but still offers a broad overview of the entire topic of IaaS clouds and aims to give a basic background on most of the prerequisites needed to understand the topics discussed.

The book makes special reference to the IBM SmartCloud Enterprise. However, the principles are general and are useful to anyone planning to automate the management of IT infrastructure using cloud technology. In contrast to technical product documentation, the book tells a story about why you might want to use the technologies described and includes sufficient background material to enable you to build the cloud applications described without having to consult numerous external references. The references are listed as suggestions for further reading, not as prerequisites to understanding the information presented.

Today cloud computing is bringing application development, business, and system operations closer together. This means that software developers need to better understand business process and system operations. It also means that business stakeholders and operations staff have to consume more software. The promise of cloud computing is that centralization, standardization, and automation will simplify the user experience and reduce costs. However, fully achieving these benefits requires a new mindset. The scope of this book is intentionally broad, to cover these aspects of application development and operation. In addition, the book is quite practical, providing numerous code examples and demonstrating system utilities for deployment, security, and maintenance.

The plan of the book runs from simple to more challenging. We hope that it gives application developers an idea of the different possible applications that can be developed. As a result, we look at some adjacent areas and related standards. Many of the topics discussed are not new; however, they are strategic to cloud computing and, when necessary, we review them so that readers do not need to seek background information elsewhere. We also will demonstrate several relatively older technologies, such as Linux services and storage systems, that are finding new uses in cloud computing.

Above all, this book emphasizes problem solving through cloud computing. At times you might face a simple problem and need to know only a simple trick. Other times you might be on the wrong track and need some background information to get oriented. Still other times, you might face a bigger problem and need direction and a plan. You will find all of these in this book.

We provide a short description of the overall structure of a cloud here, to give the reader an intuitive feel for what a cloud is. Most readers will have some experience with virtualization. Using virtualization tools, you can create a virtual machine with the operating system install software, make your own customizations to the virtual machine, use it to do some work, save a snapshot to a CD, and then shut down the virtual machine. An Infrastructure as a Service (IaaS) cloud takes this to another level and offers additional convenience and capability.

Using an IaaS cloud you can create the virtual machine without owning any of the virtualization software yourself. Instead, you can access the tools for creating and managing the virtual machine via a web portal. You do not even need the install image of the operating system; you can use a virtual machine image that someone else created previously. (Of course, that someone else probably has a lot of experience in creating virtual machine images, and the image most likely went through a quality process before it was added to the image catalog.) You might not even have to install any software on the virtual machine or make customizations yourself; someone else might have already created something you can leverage. You also do not need to own any of the compute resources to run the virtual machine yourself: Everything is inside a cloud data center. You can access the virtual machine using secure shell or a remote graphical user interface tool, such as Virtual Network Computing (VNC) or Windows® Remote Desktop. When you are finished, you do not need to save the virtual machine to a CD; you can save it to the cloud storage system. Although you do not have to own any of the infrastructure to do all this yourself, you still have to pay for it in some way. The cloud provider handles that automatically as well, based on the quantity of resources that you have used. This is the cloud pay-as-you-go concept.

The cloud provider has to invest in a lot of infrastructure to support this. Figure I.1 shows a high-level overview of an Infrastructure as a Service cloud.

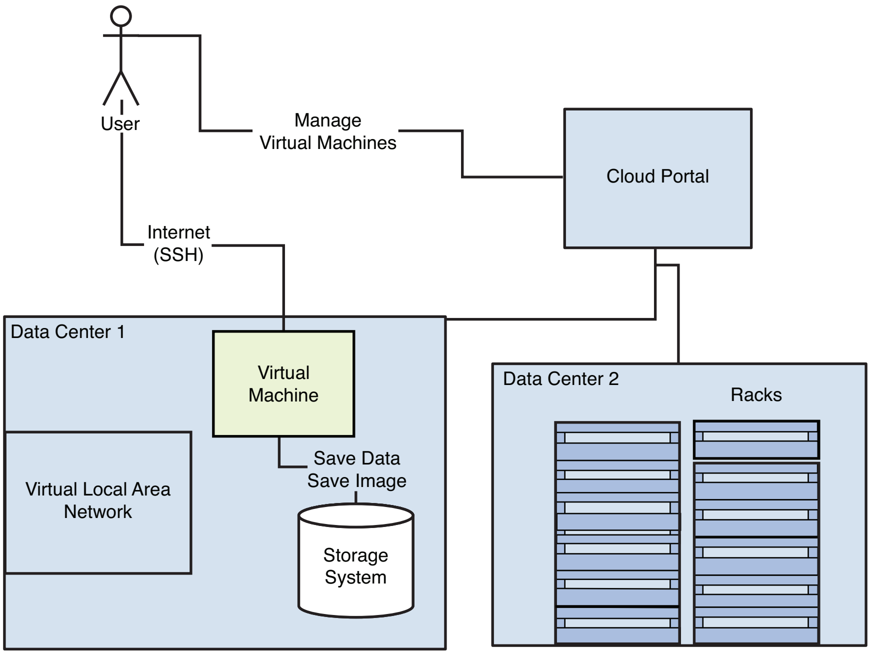


Figure I.1 Conceptual diagram of an Infrastructure as a Service cloud

The figure shows two cloud data centers with rack-based servers. Each server has many CPUs and can support multiple virtual machines of different sizes. This is a major investment for the cloud provider and the first advantage that a cloud user might think of, compared to in-house virtualization: With a cloud, you can have as many computing resources as you need for as short or long of a duration as desired; you are not limited by the computing capacity of your local facilities. We refer to this characteristic as elasticity. You also connect to the cloud via the Internet, which is convenient if you are hosting a web site but requires you to consider security. This is where the virtual local area network shown in Figure I.1 can help you.

The cloud also provides a network storage system, which you can use for storing either virtual machine images or data. Although the cost of ownership of network storage systems is declining, owning your own network storage system is still expensive and affordable to usually only medium to large companies. Blocks of the storage system can be carved off and made available as block storage volumes that can attach to virtual machines. Another aspect of data storage and backup in cloud environments is that multiple data centers are available for making redundant copies of data and providing high availability for mission-critical applications.

The cloud portal provides all this self-service as an additional aspect of cloud computing, which is a great savings for enterprises. No need to ask an administrator every time you need a new server, IP address, or additional storage—the cloud portal provides a control panel that gives you an overview of resources that end users can manage on demand. Not only are fewer administrators needed, but the consumers of the resources also have access to the resources more quickly. This results in both a savings in capital and staff needed and a more agile business.

Another aspect of cloud computing that is immediately apparent to independent software vendors is that public clouds provide a platform for a marketplace. Visibility of resources and services on the cloud can be categorized at three levels: private, shared, and public. Publicly visible resources, especially virtual machine images, provide an opportunity for independent software vendors to sell services.

Terminology

This section gives some of the basic terminology for cloud computing, to give readers a common resource for the terms used. Upcoming chapters explain the terminology in more detail for specialized aspects of cloud computing.

instance—A virtual machine instance. Sometimes referred to as a node.

image—A template for creating a virtual machine. A large file that saves the state of a virtual machine so that a new virtual machine can be created from it.

virtual local area network (VLAN)—An abstraction of the traditional local area network that does not depend on physical connections. A VLAN usually is a resource that a cloud user uses and is isolated from the Internet.

public cloud—A cloud from which multiple enterprises or individuals can consume services. IBM SmartCloud Enterprise is a public cloud that allows only enterprises as customers.

private cloud—A cloud that an enterprise operates for its sole use.

multitenant—A service that multiple tenants share. In this context, a tenant is usually an enterprise, and separation of the tenants' resources is implied.

compute size—The number of virtual CPUs, amount of memory, and hard disks dedicated to a virtual machine.

elasticity—The capability to scale resources on demand, such as dynamically adding virtual machines or IP addresses.

Organization of the Book

The book is divided in to three parts.

Background Information

The first part of the book covers background knowledge on cloud computing. It begins with Chapter 1, "Infrastructure as a Service Cloud Concepts," and covers the basic reasons for using cloud computing by looking at some use cases. This chapter then explains some basic cloud concepts and the resource model of the entities we are managing. The chapter provides a context and language for the chapters that follow. It is followed by a description of how to set up development environments in the cloud. To this point, all the concepts apply equally to any Infrastructure as a Service cloud.

Developing Cloud Applications

The second part of the book describes how to use cloud tools and develop simple cloud applications, and it explores potential cloud application areas. It includes chapters on developing on the cloud, developing with the IBM SmartCloud Enterprise, leveraging standards, and creating cloud services and applications. The chapters also describe the command-line toolkit, Java, and REST APIs for managing resources specifically for IBM SmartCloud Enterprise, as well as provide a number of code examples. In addition, this part discusses standards that relate to cloud computing and some open source projects and covers how to leverage those standards to interoperate between clouds. Following that, this part describes several application areas that are becoming important in cloud computing, such as image customization, network services, software installation and management, storage, and remote desktops.

Exploring Hosting Cloud Applications

The third section of the book discusses hosting applications on the cloud. This includes chapters on security; monitoring, performance, and availability; and operations and maintenance on the cloud. First, we provide an overview of relevant security areas and techniques for hardening applications. We then discuss monitoring, performance, and availability. Finally, we discuss business support systems and maintenance.

The book uses a scenario to illustrate and tie together the different concepts discussed. Throughout, we focus on a hypothetical company called IoT Data that provides a data storage service for Internet-enabled devices.

Disclaimer

Any recommended solutions contained in this book are not guaranteed. Warranty is not implied for any source code. All source code should be understood as sample for illustrative purposes only. IBM does not support or endorse any information in this book.

This page intentionally left blank

CHAPTER 1

Infrastructure as a Service Cloud Concepts

This chapter discusses Infrastructure as a Service (IaaS) concepts with the goal of giving cloud application developers background knowledge and helping them explore why they might want to use cloud computing.

The United States National Institute for Standards and Technology (NIST) defines cloud computing as a model for convenient and rapid network access to a shared pool of computing resources that can be provisioned with minimal management effort [Mell and Grance, 2009]. According to this definition, cloud computing has five essential characteristics:

- On-demand self-service
- Broad network access
- Multitenancy
- Rapid elasticity
- Measured service (pay as you go)

NIST also describes four deployment models:

- **Private cloud**—An organization operates a cloud for its own use. A private cloud can be either on-site at an enterprise's own premises or off-site at the cloud provider's location, with network connectivity and isolation from the outside using a virtual private network (VPN). A private cloud does not need multitenant capability, even though this is one of the five essential characteristics listed earlier.
- **Community cloud**—Several organizations use the cloud. For example, several government organizations might share both goals and resources.
- **Public cloud**—A cloud provider offers cloud services to the public-at-large.
- **Hybrid cloud**—Two or more clouds are federated by some enabling technology.

The content in this book applies to each of these models. However, some of the technologies are more applicable to one of more of the different types of clouds. For private clouds, you will need to operate the cloud itself more independently, so you need a deeper background in virtualization technologies. Public clouds tend to be large in scale, enabling independent software vendors (ISVs) and others to develop innovative services and solutions. To do this successfully, ISVs need to understand how to develop reusable cloud services. Interoperability is important in hybrid clouds, and you might find yourself focusing on standards. Likewise, collaboration is important in community clouds, so open source projects and collaboration techniques might be important.

Workloads

The term *workload* in the context of cloud computing is an abstraction of the use to which cloud consumers put their virtual machines on the cloud. For example, a desktop workload might be supporting a number of users logging on to interactive desktop sessions. An SAP workload might be a system of virtual machines working together to support an enterprise's SAP system. Workloads are a key characteristic differentiating the requirements for cloud computing. Different workloads have different characteristics in terms of computing capacity, variability of load, network needs, back-up services, security needs, network bandwidth needs, and other quality-of-service metrics. At a high level, cloud workloads are divided into three groups: server centric, client centric, and mobile centric. Table 1.1 summarizes the common types of cloud workloads.

Table 1.1 Common Workloads in Cloud Computing

Workload	Description and Examples	Key Quality-of-Service Metrics
Server Centric		
Web sites	Freely available web sites for social networking, informational web sites large number of users	Large amounts of storage, high network bandwidth,
Scientific computing	Bioinformatics, atmospheric modeling, other numerical computations	Computing capacity
Enterprise software	Email servers, SAP, enterprise content management	Security, high availability, customer support
Performance testing	Simulation of large workloads to test the performance characteristics of software under development	Computing capacity
Online financial services	Online banking, insurance	Security, high availability, Internet accessibility

Workload	Description and Examples	Key Quality-of-Service Metrics
E-commerce	Retail shopping	Variable computing load, especially at holiday times
Core financial services	Banking and insurance systems	Security, high availability
Storage and backup services	General data storage and backup	Large amounts of reliable storage
Client Centric		
Productivity applications	Users logging on interactively for email, word processing, and so on	Network bandwidth and latency, data backup, security
Development and testing	Software development of web applications with Rational Software Architect, Microsoft® Visual Studio, and so on	User self-service, flexibility, rich set of infrastructure services
Graphics intensive	Animation and visualization software applications	Network bandwidth and latency, data backup
Rich Internet applications	Web applications with a large amount of JavaScript	
Mobile Centric		
Mobile services	Servers to support rich mobile applications	High availability

It is apparent from Table 1.1 that different workloads are appropriate for different types of clouds. For example, free online social networking web sites need many virtual machines to support many users and save large numbers of media files. Public cloud computing is ideal for supporting online social networking sites. Security and high availability is a top consideration for core financial services that need to be isolated from the Internet. The data integrity provided by a relational database is important for financial applications, to ensure that financial transactions are accounted for accurately. However, social networking web sites often use NoSQL data stores that do not provide full relational integrity.

The workloads can be refined further. For example, desktop needs are different for a handful of developers than they are for a large number of general employees. The developers might use a Linux desktop and set up everything themselves. The general employees might use a standard desktop image maintained from a central point. Support is also important for the general employees, who do not have the expertise to troubleshoot and reinstall, if needed, as developers do.

The paper *MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds* [Saripalli and Pingali, 2011] discusses in detail cloud workloads and decision making for enterprise cloud adoption.

Use Cases

This section explores some of the use cases driving cloud computing. Cloud computing offers many advantages that are important for individual use cases. Infrastructure virtualization also opens up new possibilities and IT assets that traditional computing does not use. Finally, operating in a public Internet environment offers new collaboration possibilities while also introducing security challenges. See "Use Cases and Interactions for Managing Clouds" [Distributed Management Task Force, 2010] for more detail on use cases.

Actors

A number of actors collaborate together in cloud use cases. Consider this basic list.

Cloud service developer—Develops software and other assets for consumption on the cloud.

Cloud service consumer—Requests cloud resources and approves business expenditures. Cloud service consumers can include users, administrators, and business managers.

Cloud provider—Provides a cloud service to consumers.

Web Site Hosting

Operating a web site that requires database access, supports considerable traffic, and possibly connects to enterprise systems requires complete control of one or more servers, to guarantee responsiveness to user requests. Servers supporting the web site must be hosted in a data center with access from the public Internet. Traditionally, this has been achieved by renting space for physical servers in a hosting center operated by a network provider far from the enterprise's internal systems. With cloud computing, this can now be done by renting a virtual machine in a cloud hosting center. The web site can make use of open source software, such as Apache HTTP Server, MySQL, and PHP; the so-called LAMP stack; or a Java™ stack, all of which is readily available. Alternatively, enterprises might prefer to use commercially supported software, such as WebSphere® Application Server and DB2®, on either Linux® or Windows operating systems. All these options are possible in IaaS clouds and, in particular, in the IBM SmartCloud Enterprise.

Figure 1.1 shows a use case diagram for this scenario.

When building the web site, the developer needs to create a virtual machine instance that will host the web and application servers needed. The developer can save an instance to an image when the development of the site reaches a certain point or just for back-up purposes. Usually an administrator does not want to use an instance that a developer created. However, the administrator needs to know the hosting requirements in detail and might use an image that the developer saved or scripts that a developer created, as a starting point. In the process of maintaining the web site, an administrator might need to add storage and clone storage for back-up purposes. After cloning, the administrator might want to copy the data to some other location, so having it offline from the production web site would be an advantage. From the users' perspective, users will be unaware that the web site is hosted in the cloud.

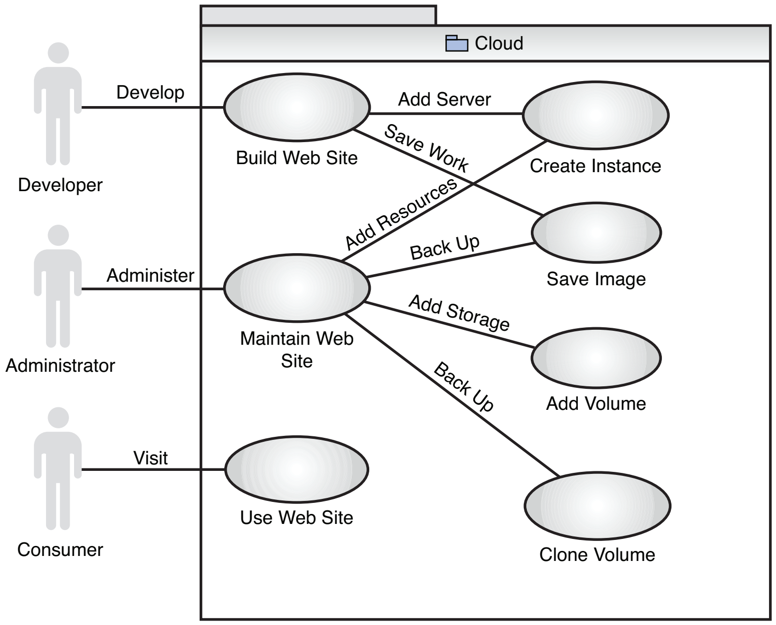


Figure 1.1 Use case diagram for hosting a web site on the cloud

The activities of the developer and administrator can be accomplished via a console with a graphical user interface, such as the one the IBM SmartCloud Enterprise provides. However, as time passes, many regular cloud users will automate with scripts. Command-line tools are ideal for these power users because they execute much faster than a user can click a mouse and navigate pages. Many power users have cheat sheets for common operations, such as installing software and patches, that they can retrieve and edit as needed. They can save scripts for creating instances, saving images, and performing other operations along with the rest of the script collection.

The main advantage of using the cloud for this use case is that renting a virtual machine in a location where it is accessible from the Internet is considerably cheaper than placing physical machines in a data center accessible from the Internet. Other cloud advantages also apply to this use case, including the rapid ability to substitute in a new virtual machine for a server experiencing a hardware fault.

Short-Term Peak Workloads

In the retail industry, workloads come in short peaks at certain times of the year (notably, at Christmas) or coincide with advertising campaigns. Quickly adding capacity during these times is important. With their elastic ability to add servers as desired, clouds are ideal in this situation. Monitoring is important because user traffic varies from year to year based on economic conditions and other factors that make predicting the workload difficult. The IBM SmartCloud Enterprise includes an IBM Tivoli Monitoring image in the catalog that can be helpful. Along with other images in the catalog, it can be rented for as long as needed, and no installation is necessary. Figure 1.2 shows a use case diagram for this scenario.

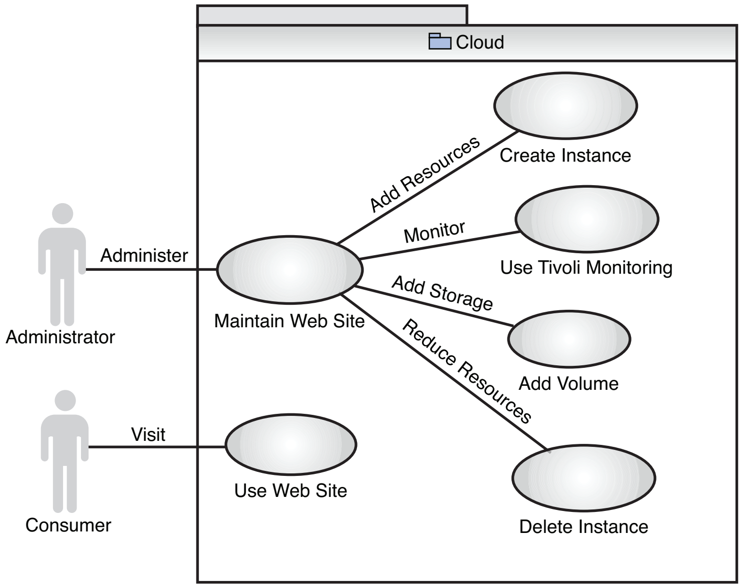


Figure 1.2 Use case diagram for monitoring peak workloads

As in the previous use case, all actions required to do this can be done in the console graphical user interface. However, scripts avoid repetitive work and save administrators time.

The main advantage of the cloud in this use case is its elastic scalability.

Proof-of-Concept

Enterprises usually do proof-of-concept or pilot studies of new technologies before committing to use them. External IT consultants are often invited to do these proof-of-concepts. The consultants are typically under a lot of pressure to deliver a large quantity of computing capacity in a short period of time. If they do not have prior experience in this area, they generally have little hope of succeeding. Assets that they can take from job to job are critical. The cloud can make this easier by allowing saved images to be reused directly and to allow consultants and enterprise users to easily share the same network space. This solution is a better one than requiring the consultant to transport physical machines, install everything on her or his laptop, or install all the software on-site at the enterprise in a short period of time.

Figure 1.3 shows a use case diagram for this scenario.

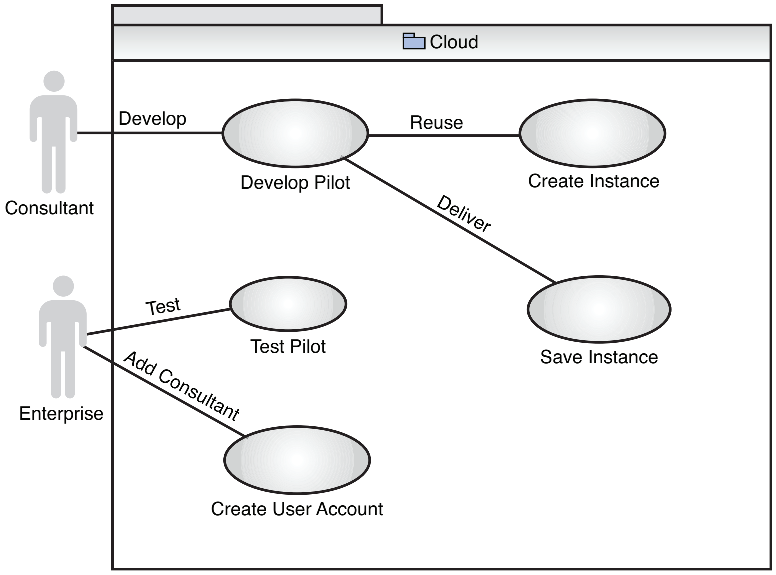


Figure 1.3 Use case diagram for a proof-of-concept on the cloud

Working in a public cloud environment with support for user administration is critical here, to allow the enterprise to add an account for the consultant. Alternatively, the consultant could use his or her account space and simply allow access via a network protocol such as HTTP. If the enterprise likes the proof-of-concept, it might want to use it long term. It can move it to the company's private network by saving an image and starting up an instance on its virtualization LAN. Table 1.2 compares a traditional proof-of-concept and a proof-of-concept on the cloud.

Table 1.2 Comparison of Traditional and Cloud Environments for a Proof-of-Concept

Traditional	Cloud
The consultant travels to the customer site.	The consultant works over the Internet.
The customer gives the consultant access to the enterprise network, subject to an approval workflow.	The customer gives the consultant access to the cloud with account or specific virtual machines with cryptographic keys.
Customer procures hardware for the pilot.	Customer creates an instance with the self-service interface.
The consultant works independently.	The consultant pulls in experts for high availability, performance, security, and so on for a few hours, as needed.

Table 1.2 Comparison of Traditional and Cloud Environments for a Proof-of-Concept (continued)

Traditional	Cloud
The consultant cannot connect his or her laptop to the enterprise network and instead must use only tools that the customer makes available.	The customer can use her or his favorite application lifecycle management tools on a laptop or available on the cloud.
The consultant installs everything from scratch.	The consultant starts up instances from prebuilt images.
The server is repurposed after completion.	Server instances are saved as images, and running instances are deleted.

The cloud enables a different set of deliverables for proof-of-concept, pilot, beta programs, and consulting projects. In traditional environments, enterprise network constraints (especially security issues) often require consultants to work with unfamiliar tools. This results in written reports documenting deployment steps and best practices that customers cannot easily consume. In other situations, consultants are left in a permanent support position long after the project has “finished.” The cloud enables a different set of deliverables, including virtual machine images, deployment topology models, and software bundles, as shown Table 1.3.

Table 1.3 Comparison of Traditional and Cloud Project Artifacts

Traditional	Cloud
Software installation program (time consuming to develop)	Virtual machine image (capturing an instance with the click of a button)
Written reports summarizing deployment steps	Deployment topology models, automation scripts
User documentation written from scratch	Documentation reused from standard images
Configuration files in miscellaneous locations	Asset added to cloud catalog
Difficult support process	Support via remote access to cloud

The primary advantages of the cloud for this use case are elastic scalability, access from the Internet, and the capability to save and reuse projects assets.

Extra Capacity

In this scenario, the IT department runs out of computing resources, delaying in-house projects. The department rents resources on the cloud to meet the shortfall. A virtual private network is used to connect to a private virtual local area network (VLAN) in the cloud to the enterprise network.

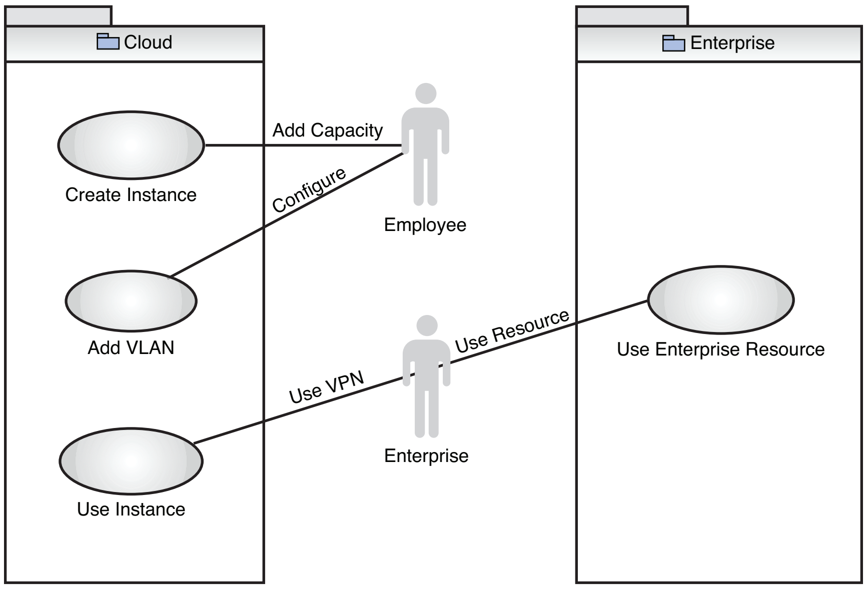


Figure 1.4 Use case diagram for adding extra capacity for enterprise IT infrastructure

Open Source/Enterprise Collaboration

Recently, enterprises have embraced the idea of open source. However, this is best done in a controlled way. An organization might be unwilling to host an open source project on SourceForge or Apache but might want to use open source in a more controlled way. By hosting the project itself on the cloud, the enterprise maintains complete control over the project while still gaining the advantages of an open source model.

Outside contributors can make use of these advantages:

- Be given user accounts without granting access to the enterprise's internal IT systems
- Use a common set of development tools hosted on the cloud

Storage System for Security Videos

Some application domains consume huge amounts of data. Video files are one example. In addition to the files themselves, a management application must allow the videos to be accessed and store additional metadata about them. Hadoop, a freely available open source distributed file system capable of storing huge amounts of data, might fulfill the storage needs of such a security video management and access system. IaaS clouds are an ideal platform for hosting Hadoop and being able to add nodes to the cluster on demand.

Business Scenario: IoT Data Hosting Provider

To tie together the information presented in this book, this section describes how it can be used in a business scenario. In this situation, the company IoT Data provides a hosting service for Internet-connected devices to store data. IoT Data's business services include the following:

- Registering devices
- Storing data from a device using a REST web service
- Conducting HTML and programmatic searches of the data
- Sharing the data in public, community, and commercial modes

IoT Data charges customers by gibibytes (GiB) of data stored and 10% of any data sold. For large customers, IoT Data also provide the entire suite of software for private hosting on the cloud itself. In this case, the changes are per virtual machine hour and depend on the size of the virtual machine (in addition to the per-GiB charge). A diagram showing the main actors and use cases for IoT Data is shown next.

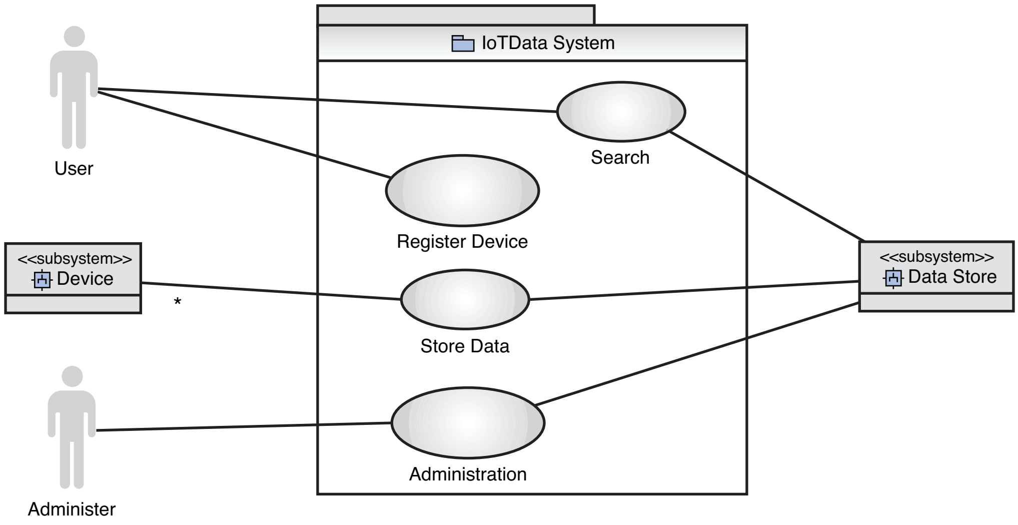


Figure 1.5 IoT Data use case diagram

IoT Data does not have a large budget to hire employees, so as much work as possible has to be automated. IoT Data also cannot afford to buy servers, so it needs a pay-as-you-go model, such as a public cloud provides. In addition, the company has few resources to develop its own software and thus must leverage as much as possible from the cloud provider. This book explains how different technologies can meet IoT Data's business needs (however, we do not actually write the code for doing so).

Virtualization

We briefly discuss virtualization, with the goal of providing a foundation for discussing IaaS clouds and the resource model. The term *virtualization* can apply to a computer (a virtual machine and the resources it uses), storage, network resources, desktops, or other entities. Virtualization of hardware resources and operating systems dates back the 1960s, with IBM mainframes, and was later used on AIX® and other UNIX® platforms. It has been a powerful tool for these platforms for many years. In 1999, VMWare introduced virtualization for low-cost Intel® x-series hardware, based on the research of its founders at Stanford University. This made the practice of virtualization more widespread.

A hypervisor, or virtual machine manager, is a software module that manages virtual machines. The hypervisor resides on the host system on which the virtual machines run. The relationship of the hypervisor to the host operating system and to the virtual machine is one of the key distinguishing characteristics of the different virtualization systems.

Major virtualization systems for x86 hardware include these:

- VMWare, a broad range of virtualization products for x86
- Xen, an open source virtualization system with commercial support from Citrix
- Windows Hyper-V, introduced by Microsoft in Windows Server 2008
- Kernel Virtualization Machine (KVM), a part of the Linux kernel since version 2.6.2

Virtualization became widespread in the early 2000s, several years before the rise of cloud computing. Virtualization offers many practical benefits, including the following:

- The ease of setting up new systems. New systems do not need to installed using installation media.
- No need to buy new hardware to simulate various system environments for debugging and support.
- The capability to recover quickly from system corruption.
- The ease of relocating and migrating systems. For example, a move to a more powerful machine can simply be a matter of taking a snapshot of a virtual machine and starting up a new virtual machine based on that snapshot.
- The ease of remote management. Physical access to data centers is tightly controlled these days. The use of virtual machines greatly reduces the need for physical access.
- The capability to run multiple operating systems simultaneously on one server.

In virtualization of hardware and operating systems, we refer to the *guest* system as the system being virtualized. The system the guest runs on is called the *host*, which uses a *hypervisor* to managing scheduling and system resources, such as memory. Several types of virtualization exist: full virtualization, partial virtualization, and paravirtualization.

Full virtualization is complete simulation of the hardware. Full virtualization is simulating to emulate. In emulation, an emulated system is completely independent of the hardware. The Android smart phone emulator and QEMU in unaccelerated mode are examples of system emulation. Full virtualization differs from emulation in that the virtual system is designed to run on the same hardware architecture as the host system. This enables the instructions of the virtual machine to run directly on the hardware, greatly increasing performance. In full virtualization, no software is needed to simulate the hardware architecture. Figure 1.6 gives a schematic diagram of full virtualization.

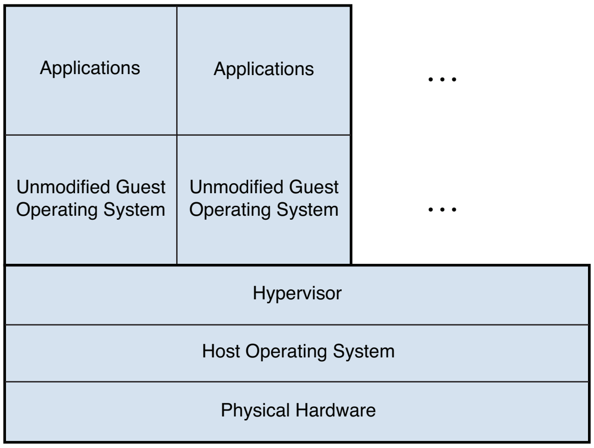


Figure 1.6 Schematic diagram of full virtualization

One of the key characteristics of full virtualization is that an unmodified guest operating system can run on a virtual machine. However, for performance reasons, some modifications are often made. Intel and AMD introduced enhancements to CPUs to allow this: the Intel VT (Virtual Technology) and AMD-V features introduced in 2005. These features support modifications of the guest operating system instructions through variations in their translation to run on the hardware. The Intel VT-x (32-bit processors) and VT-i (IA64 architecture) introduced two new operation levels for the processor, to be used by hypervisors to allow the guest operating systems to run unmodified. Intel also developed a VT-d feature for direct IO, to enable devices to be safely assigned to guest operating systems. VT-d also supports direct memory access (DMA) remapping, which prevents a direct memory access from escaping the bounds of a virtual machine. AMD has a similar set of modifications, although implemented somewhat differently.

Figure 1.6 shows the hypervisor running on top of the host operating system. However, this is not necessary for some hypervisors, which can run in "bare-metal" mode, installed directly on the hardware. Performance increases by eliminating the need for a host operating system.

VMWare Workstation and the IBM System z® Virtual Machine are examples of full virtualization products. VMWare has a wide range of virtualization products for x86 systems. The ESX Server can run in bare-metal mode. VMWare Player is a hosted hypervisor that can be freely downloaded and can run virtual machines created by VMWare Workstation or Server. Xen can run as a full virtualization system for basic architectures with the CPU virtualization features present.

In paravirtualization, the hardware is not simulated; instead, the guest runs in its own isolated domain. In this paradigm, the hypervisor exports a modified version of the physical hardware to the guest operating system. Some changes are needed at the operating system level. Figure 1.7 shows a schematic diagram of paravirtualization.

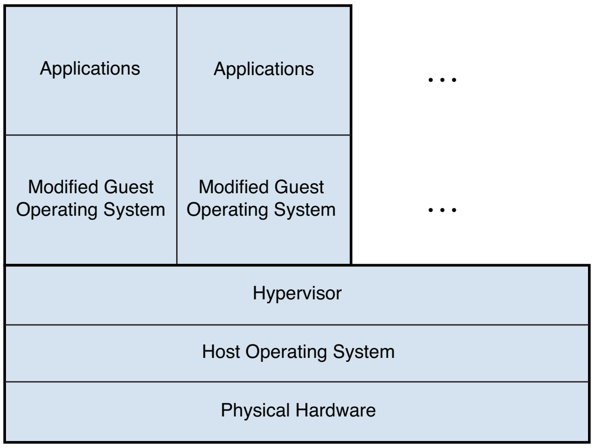


Figure 1.7 Schematic diagram of paravirtualization

Xen is an example of a paravirtualization implementation. VMWare and Windows Hyper-V can also run in paravirtualization mode.

In operating system–level virtualization, the hypervisor is integrated into the operating system. The different guest operating systems still see their own file systems and system resources, but they have less isolation between them. The operating system itself provides resource management. Figure 1.8 shows a schematic diagram of operating system–level virtualization.

One of the advantages of operating system–level virtualization is that it requires less duplication of resources. Logical partitions on the IBM AIX operating system serves as an example of operating system–level virtualization.

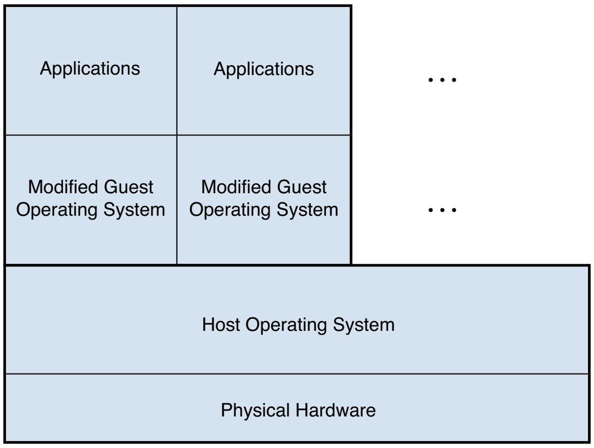


Figure 1.8 Schematic diagram of operating system–level virtualization

KVM can be considered an example of operating system–level virtualization. KVM is a Linux kernel module and relies on other parts of the Linux kernel for managing the guest systems. It was added to the Linux kernel in version 2.6. KVM exports the device `/dev/kvm`, which enables guest operating systems to have their own address spaces, to support isolation of the virtual machines. Figure 1.9 shows the basic concept of virtualization with KVM.

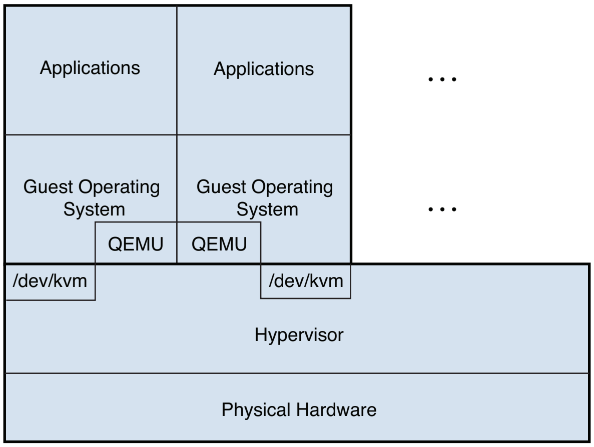


Figure 1.9 Virtualization with KVM

KVM depends on libraries from the open source QEMU for emulation of some devices. KVM also introduces a new process mode, called *guest,* for executing the guest operating systems. It is a privilege mode sufficient to run the guest operating systems but not sufficient to see or interfere with other guest systems or the hypervisor. KVM adds a set of shadow page tables to map memory from guest operating systems to physical memory. The `/dev/kvm` device node enables a userspace process to create and run virtual machines via a set of `ioctl()` operations, including these:

- Creating a new virtual machine
- Allocating memory to a virtual machine
- Reading and writing virtual CPU registers
- Injecting an interrupt into a CPU
- Running a virtual CPU

In addition, guest memory can be used to support DMA-capable devices, such as graphic displays. Guest execution is performed in the loop:

- A userspace process calls the kernel to execute guest code.
- The kernel causes the processor to enter guest mode.
- The processor executes guest code until it encounters an IO instruction or is interrupted by an external event.

Another key difference between virtualization systems is between client-based and server-based virtualization systems. In a client-based virtualization system, such as VMWare Workstation, the hypervisor and virtual machine both run on the client that uses the virtual machine. Server products, such as VMWare ESX, and remote management libraries, such as libvirt, enable you to remotely manage the hypervisor. This has the key advantage of freeing the virtual machine from the client that consumes it. One more step in virtualization is needed in cloud computing, which is to be able to manage a cluster of hypervisors.

Computing capacity is not the only resource needed in cloud computing. Cloud consumers also need storage and network resources. Those storage and network resources can be shared in some cases, but in other cases, they must be isolated. Software based on strong cryptography, such as secure shell (SSH), can be used safely in a multitenant environment. Similarly, some software stores data in encrypted format, but most does not. Thus, storage and network virtualization and tenant isolation are needed in clouds as well.

Storage virtualization provides logical storage, abstracting the details of the storage technology from users and application software. This is often implemented in network-attached storage devices, which can provide multiple interfaces to a large array of hard disks. See the "Storage" section later in this chapter for more details.

Network resources can also be virtualized. This book is most concerned with virtualization at the IP level. In the 1990s, local area networks (LANs) were created by stringing Ethernet cable between machines. In the 2000s, physical network transport was incorporated directly into cabinets that blade servers fit into, to keep the back of the cabinet from looking like a bird's nest of

Ethernet cable. Today we can do the virtual equivalent of that with virtual network management devices in a VLAN, which can be managed conveniently and also provides network isolation for security purposes. See the "Network Virtualization" section later in this chapter for more details.

These virtualization platforms provide great convenience, but management comes at the cost of learning them and developing efficient skills. Some other limitations exist as well:

- The different virtual hosts must be managed separately, and only a limited number of guest machines can be placed on one host. Today 16 dual-core CPU machines are affordable, to support around 32 capable virtual machines, but we need a way to scale to larger numbers.
- End users still need to contact a system administrator when they want a new virtual machine. The administrator then must track these requests and charge for use.
- Virtualization itself does not provide a library of images that can be readily used. A feature of organizations that use a lot of direct virtualization is image sprawl, consisting of a large number of unmanaged virtual machine images.
- The system administrator still must manage the various pieces of the infrastructure. Some small companies cannot afford system administrators, and many large organizations would like to reduce the number of system administrators they currently have.
- Hardware still must be bought. Most enterprises would like to minimize their capital investments in hardware.

Infrastructure as a Service Clouds

An IaaS cloud provides abstractions beyond virtualization so that you do not need to learn how to manage them yourself. In fact, when using an IaaS cloud, you normally are not even aware of what virtualization platform is being used. In this case, the cloud service provider is concerned about the virtualization platform; you do not need to worry about it. Figure 1.10 shows some of the main components of an IaaS cloud.

The user logs into a self-service user interface that is part of a system called the business support services (BSS). The BSS knows how to charge a user for the resources used, and the self-service user interface enables the user to create and manage resources such as virtual machines, storage, and IP addresses. This gives the user a central location for viewing and managing all resources instead of being left to manage a collection of independent virtualization technologies. A programmatic API also is often provided, to enable automation of resource management for a similar set of capabilities as those of the self-service user interface. The operational support system (OSS) manages a collection of hypervisors and federates other virtualized resources. The end result is that the user can have access to the virtual machine without having to know how it was created.

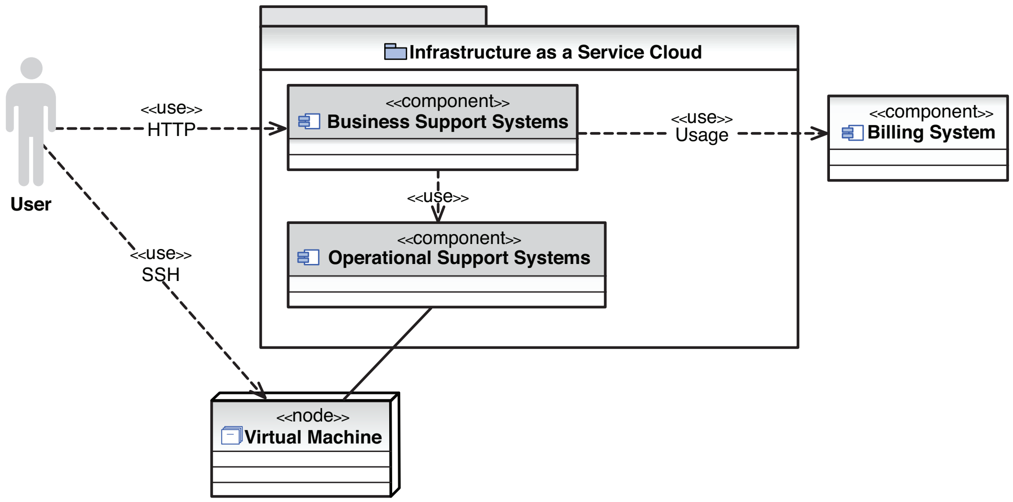

Infrastructure as a Service Cloud
User
<<use>> HTTP
<<component>> Business Support Systems
<<use>> Usage
<<component>> Billing System
<<use>>
<<component>> Operational Support Systems
<<use>> SSH
<<node>> Virtual Machine

Figure 1.10 Basic concepts of an Infrastructure as a Service cloud

Additional features of IaaS clouds, such as the IBM SmartCloud Enterprise, are convenient for enterprises. Importantly, the relationship is between the cloud provider and the enterprise. An enterprise contact can thus manage the users, who can create and use resources that the enterprise is paying for. In addition, the work products created by people using the cloud should belong to the enterprise, and the cloud infrastructure should support this.

One of the most interesting aspects of cloud computing is that it enables a new level of tooling and collaboration. It enables reuse of work products, especially images, by teams. For example, an operating system expert can set up a base operating system image, a software developer can add an installation of a software product on top of it, and an enterprise user can make use of the image by taking snapshots suitable for his or her enterprise's needs. Figure 1.11 shows how a developer can interact with cloud tools to provide assets that an end user can consume.

Business support systems (BSS) are a critical part of the cloud and might be important to your applications if you sell services to customers. Most online systems need a BSS. BSS includes subscriber management, customer management, contract management, catalog management, business partner enablement, metering, and billing. Clearly, BSS is a wider concept than just IaaS. The Apple iPhone AppStore and the Android AppStore are examples of platforms that include a BSS.

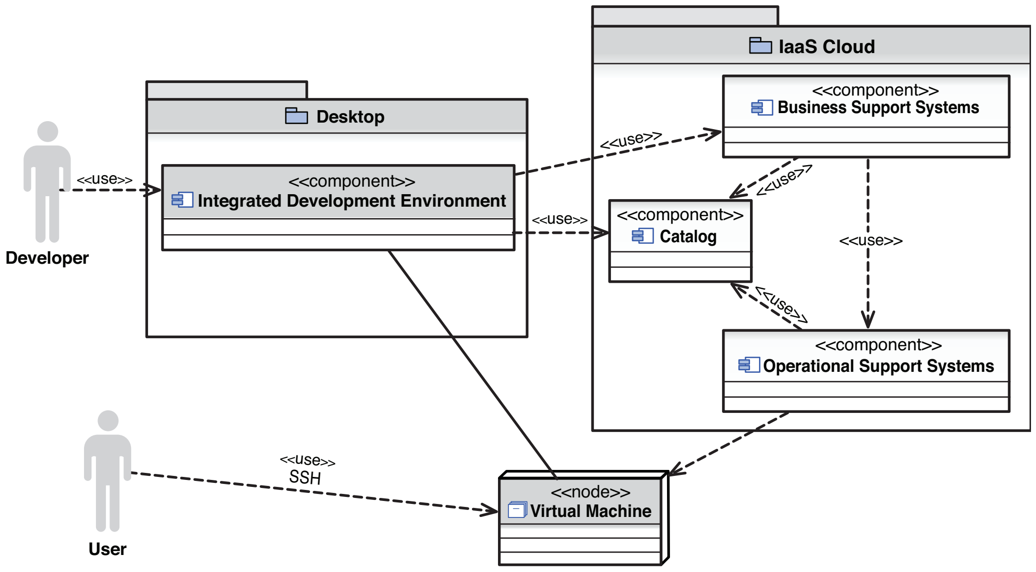


Figure 1.11 Use of development tools in a cloud environment

Other Cloud Layers

Cloud layers operate at a higher level of abstraction than IaaS, including Platform as a Service (PaaS), Software as a Service (SaaS), and Business as a Service (BaaS). In its definition of cloud computing, NIST recognizes three of these layers: IaaS, PaaS, and SaaS. Figure 1.12 illustrates this concept of different layers of cloud computing.

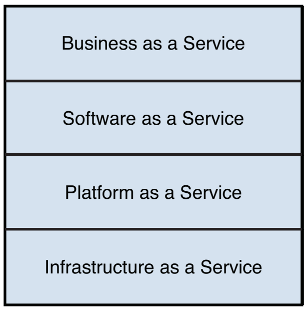


Figure 1.12 Cloud platform layers

Infrastructure as a Service is a term for services that provide a model for dynamically allocating infrastructure and software, starting with an OS, on that infrastructure. Platform as a Service describes concrete services used in the execution of an application or higher-level service. The services provide some generalized and reusable capability to their software consumer and are thus a "platform" service being consumed. They bring their own interface and programming model for the consumer to use, along with their own API, data, messaging, queueing, and so on.

If a platform service is hosted by an infrastructure service provider, the IaaS API is likely used as part of the process to instantiate and access the platform service, but it is a separate concept.

To complete the terminology, Software as a Service is typically a self-sufficient software solution to a consumer need. Typically, this is a tool or business application with on-demand, turn-key characteristics.

If a software service leverages a platform service, it normally does so transparently. If the software service is hosted by an infrastructure service provider, the IaaS application programming interface can be used as part of the process to instantiate and access the software service.

If you look at the service stack from the top down, you can see some of the value the other layers provide. At the very top are business services such as Dunn and Bradstreet, which provides analysis and insight into companies that you might potentially do business with. Other examples of business services are credit reporting and banking. Providing business services such as these requires data stores for storing data. However, a relational database by itself is not sufficient: The data retrieval and storage methods must be integrated into programs that can provide user interfaces for people can use. Relational databases also need to be maintained by database administrators who archive and back up data. This is where Platform as a Service comes in. Platform as a Service provides all the services that enable systems to run by themselves, including scaling, failover, performance tuning, and data retrieval. For example, the SalesforceForce.com platform provides a data store where your programs can store and retrieve data without you ever needing to worry about database or system administration tasks. It also provides a web site with graphical tools for defining and customizing data objects. IBM Workload Deployer is another Platform as a Service that runs on an infrastructure as a service cloud but is aware of the different software running on individual virtual machines; it can perform functions such as elastic scaling of application server clusters.

With Platform as a Service, you still need to write a program that enables a user to interact with it via a graphical user interface. If you do not like that idea, you can use Software as a Service. Salesforce.com enables enterprises to use a customer relationship management (CRM) system without having to do any programming or software installation. Its web site also supports graphical tools for customizing menus, data entry forms, and reports. It works great if you all you need to do is create, retrieve, update, and delete data or use a predefined service, such as email. If you need to do more than that, you need to drop down to the Platform as a Service level.

Virtual Machine Instances

An instance is a running virtual machine, in addition to some data the cloud maintains to help track ownership and status. The cloud manages a large pool of hardware that can be used to create running instances from images. The virtual machine includes a copy of the image that it instantiates and the changes that it saves while it runs. The instance also includes virtualizations of the different hardware that it needs to run, including CPUs, memory, disk, and network interfaces. The cloud manages a pool of hypervisors that can manage the virtual machine instances. However, as a user of the cloud, you do not need to worry about the hypervisors. In fact, the hypervisor you are using—KVM, Xen, VMWare, or any other—makes no difference.

When you delete an instance, that hardware can be reused. The cloud scrubs your hard disk before doing so, to make sure that the next user of the hardware finds no traces of previous data.

Virtual Machine Images

A virtual machine image is a template for creating new instances. You can choose images from a catalog to create images or save your own images from running instances. Specialists in those platforms often create catalog images, making sure that they are created with the proper patches and that any software is installed and configured with good default settings. The images can be plain operating systems or can have software installed on them, such as databases, application servers, or other applications. Images usually remove some data related to runtime operations, such as swap data and configuration files with embedded IP addresses or host names.

Image development is becoming a larger and more specialized area. One of the outstanding features of the IBM SmartCloud Enterprise is the image asset catalog. The asset catalog stores a set of additional data about images, including a "Getting Started" page, a parameters file that specifies additional parameters needed when creating an instance, and additional files to inject into the instance at startup. It also hosts forums related to assets, to enable feedback and questions from users of images to the people who created those images. Saving your own images from running instances is easy, but making images that other people use requires more effort; the IBM SmartCloud Enterprise asset catalog provides you with tools to do this.

Because many users share clouds, the cloud helps you track information about images, such as ownership, history, and so on. The IBM SmartCloud Enterprise knows what organization you belong to when you log in. You can choose whether to keep images private, exclusively for your own use, or to share with other users in your organization. If you are an independent software vendor, you can also add your images to the public catalog.

Some differences between Linux and Windows exist. The filelike description of the Linux operating system makes it easy to prepare for virtualization. An image can be manipulated as a file system even when the instance is not running. Different files, such as a user's public SSH key and runtime parameters, can be injected into the image before booting it. Cloud operators take advantage of this for ease of development and to make optimizations. The same method of manipulating files systems without booting the OS cannot be done in Windows.

Storage

Virtualization of storage can be done in different ways to make physical storage transparent to consumers of I/O services. Block storage is storage handled as a sequence of bytes. In file-based storage systems, the block storage is formatted with a file system so that programs can make use of file-based I/O services to create and manage files. Virtualization can be done at both levels.

Block Storage

Usually, the person installing an operating system partitions physical hard disks in a physical computer. A disk partition is a logical segment of a hard disk. Partitioning a disk can have several advantages, including separating the operating system from user files and providing a storage area for swapping. The disadvantages of partitioning include the need to reorganize or resize if you run out of space on one partition. The classical example is running out of space on your operating system partition (C:) when you still have plenty of space on the other partitions. One advantage of partitions in virtual systems is that you can plan for a large amount of storage space but do not have to actually allocate that space until you need to use it.

Clouds can make use of partitions as well. In the IBM SmartCloud Enterprise, when you provision a virtual machine, you have an option to create only the root file partition. This optimizes startup time. If you have a large amount of storage associated with the image, the time savings can be considerable. Later, when you use the storage, it is then allocated.

A Linux *logical volume manager* (LVM) provides a level of abstraction above block devices, such as hard disks, to allow for flexibility in managing storage devices. This can make it easier to resize physical partitions, among other tasks. The LVM manages *physical volumes*, which can be combined to form a *volume group*. *Logical volumes* can then be created from the volume groups. The logical volumes can span multiple physical volumes, allowing them to be any size up to the total size of the volume group.

Copy on write is a technique for efficiently sharing large objects between two or more clients. Each client appears to have its own writable copy of the object, but each client actually has only a read-only copy of the shared object. When a client tries to write to the object, a copy of the block is made and the client is given its own copy. This is efficient when the object is only rarely changed by client programs, such as an operating system when a virtual machine loads and runs it. This technique can make starting the virtual machine much faster than first copying the operating system to a separate storage area before booting the virtual machine. In this context, copy on write is often used with a network-based file system.

The term *direct attached storage* is usually used to contrast local block-based storage with *network attached storage*. Direct attached storage is simple, cheap, and high performance. Its high-performance characteristics are due to the fact that it is directly attached. Its disadvantages include that its lifetime is usually tied to the lifetime of the virtual machine. In addition, it might not be scalable if you do not have physical access to the machine. In a cloud environment, you often have no way of increasing direct attached storage, so be sure to start with enough.

In an Infrastructure as a Service cloud, you do not need to be concerned with the different storage implementations the cloud provider uses. Instead, you should be concerned with the amount of storage and the level of performance the storage service provides. Cloud consumers need a basic understanding of the concepts to do informed planning. Generally, local storage comes and goes with virtual machines, and remote storage can be managed as an independent entity that can be attached to or detached from a virtual machine. In general, local and remote storage have a large difference in performance. Remote storage is not suitable for some applications, such as relational databases.

File-Based Storage

File systems provide a level of abstraction over block storage, to allow software to more easily use and manage files. As with block-based storage, a fundamental difference exists between local and remote file systems. Common local file systems in clouds are ext3 and ext4 on Linux and NTFS on Windows. Common remote file systems are NFS on Linux and CIFS on Windows. One huge difference between remote files systems and network attached storage, such as AoE and iSCSI, is that remote file systems are designed for multiple clients with simultaneous write access. This is not possible with remote block devices provided by network attached storage.

Some distributed file systems can span many servers. Apache Hadoop is an example of such a distribute file system used by many large web sites with huge storage requirements. Hadoop is discussed in the upcoming "Hadoop" section in Chapter 5, "Open Source Projects."

Table 1.4 compares different basic storage options.

Table 1.4 Comparison of Different Storage Options

Storage Option	Advantages	Disadvantages
Local block based	High performance	Lifetime tied to a virtual machine
Remote block based	Can be managed independently, with a lifetime not tied to a virtual machine	Cannot be shared among multiple virtual machines
Local file based	High performance	Lifetime tied to a virtual machine
Remote file based	Can be shared among different clients	Relatively lower performance

The persistence of virtual machines and their local storage can vary with different virtualization methods. Some virtual machines' local storage disappears if the virtual machine is deleted. In other implementations, the local storage is kept until the owner deletes the virtual machine. The IBM SmartCloud Enterprise uses this model. Some virtualization implementations support the concept of a persistent virtual machine. In a third model, some implementations boot the operating system from network attached storage and do not have any local storage. Be sure to understand the storage model your cloud provider uses so that you do not lose data.

Network Virtualization

Networking is one of the fundamental elements of cloud computing and also one of the hazards to users of cloud computing. Network performance degradation and instability can greatly affect the consumption of cloud resources. Applications that are relatively isolated or are specially designed to deal with network disruptions have an advantage running in the cloud.

From a different perspective, network resources can be virtualized and used in cloud computing just as other resources are. In this section, we first discuss basic use of IP addresses in a cloud context and then cover virtual networks.

Delivery of cloud services takes place over networks at different levels using different protocols. This is one of the key differences in cloud models. In PaaS and SaaS clouds, delivery of services is via an application protocol, typically HTTP. In IaaS, cloud services can be delivered over multiple layers and protocols—for example, IPSec for VPN access and SSH for command-line access.

Management of the different layers of the network system also is the responsibility of either the cloud provider or the cloud consumer, depending on the type of cloud. In a SaaS model, the cloud provider manages all the network layers. In an IaaS model, the cloud consumer manages the network levels, except for the physical and data link layers. However, this is a simplification because, in some cases, the network services relate to the cloud infrastructure and some services relate to the images. The PaaS model is intermediate between IaaS and SaaS.

Table 1.5 summarizes the management of network layers in different cloud scenarios.

Table 1.5 Management for Network Layers

OSI Layer	Example Protocols	IaaS	PaaS	SaaS
7 Application	HTTP, FTP, NFS, SMTP, SSH	Consumer	Consumer	Provider
6 Presentation	SSL, TLS	Consumer	Provider	Provider
5 Session	TCP	Consumer	Provider	Provider
4 Transport	TCP	Consumer	Provider	Provider
3 Network	IP, IPSec	Consumer	Provider	Provider
2 Data link	Ethernet, Fibre Channel	Provider	Provider	Provider
1 Physical	Copper, optical fiber	Provider	Provider	Provider

This table is a simplification of the many models on the market. However, it shows that an IaaS gives cloud consumers considerably more flexibility in network topology and services than PaaS and SaaS clouds (but at the expense of managing the tools that provide the flexibility).

IP Addresses

One of the first tasks in cloud computing is determining how to connect to the virtual machine. Several options exist when creating a virtual machine: system generated, reserved, and VLAN IP address solutions. System-generated IP addresses are analogous to Dynamic Host Control Protocol (DHCP)–assigned addresses. They are actually static IP addresses, but the IaaS cloud assigns them. This is the easiest option if all you need is a virtual machine that you can log into and use.

Reserved IP addresses are addresses that can be provisioned and managed independently of a virtual machine. Reserved IP addresses are useful if you want to assign multiple IP addresses to a virtual machine.

IPv6 is an Internet protocol intended to supersede IPv4. The Internet needs more IP addresses than IP v4 can support, which is one of the primary motivations for IPv6. The last top-level block of IPv4 addresses was assigned in February 2011. The Internet Engineering Task Force (IETF) published *Request for Comments: 2460 Internet Protocol, Version 6 (IPv6)*, which was the specification for IPv6 in 1998. IPv6 also provides other features not present in IPv4. Network security is integrated into the design of IPv6, which makes IPSec a mandatory part of the implementation. IPv6 does not specify interoperability with IPv4 and essentially creates an independent network. Today usage rates of IPv6 are very low, and most providers operate in compatibility/tolerance mode. However, that could change.

Network Virtualization

When dealing with systems of virtual machines and considering network security, you need to manage networks. Network resources can be virtualized just like other cloud resources. To do this, a cloud uses virtual switches to separates a physical network into logical partitions. Figure 1.13 shows this concept.

VLANs can act as an extension of your enterprise's private network. You can connect to it via an encrypted VPN connection.

A hypervisor can share a single physical network interface with multiple virtual machines. Each virtual machine has one or more virtual network interfaces. The hypervisor can provide networking services to virtual machines in three ways:

- Bridging
- Routing
- Network address translation (NAT)

Bridging is usually the default mode. In this mode, the hypervisor works at the data link layer and makes the virtual network interface externally visible at the Ethernet level. In routing mode, the hypervisor works at the network layer and makes the virtual network interface externally visible at the IP level.

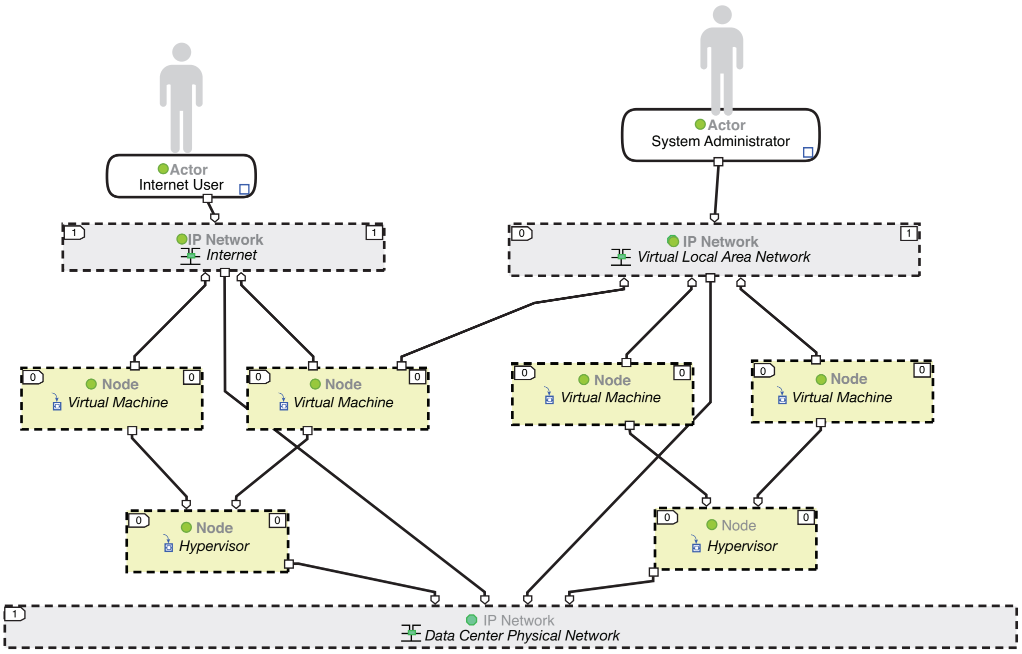


Figure 1.13 Physical and virtual networks in a cloud

In network address translation, the virtual network interface is not visible externally. Instead, it enables the virtual machine to send network data out to the Internet, but the virtual machine is not visible on the Internet. Network address translation is typically used to hide virtualization network interfaces with private IP addresses behind a public IP address used by a host or router. The NAT software changes the IP address information in the network packets based on information in a routing table. The checksum values in the packet must be changed as well.

NAT can be used to put more servers on the network than the number of virtual machines you have. It does this by port translation. This is one reason IPv6 is still not in wide use: Even though the number of computers exceeds the number of IP addresses, you can do some tricks to share them. For example, suppose that you have a router and three servers handling HTTP, FTP, and mail, respectively. You can assign a public IP address to the router and private IP addresses to the HTTP, FTP, and mail servers, and forward incoming traffic (see Table 1.6).

Table 1.6 Example of Network Address Translation

Public IP	Port	Private IP
9.0.0.1 (router)	80, 443	192.168.0.1 (HTTP server)
	21	192.168.0.2 (FTP server)
	25	192.168.0.3 (mail server)

Desktop Virtualization

Desktops are another computing resource that can be virtualized. Desktop virtualization is enabled by several architectures that allow remote desktop use, including the X Window System and Microsoft Remote Desktop Services. The X Window System, also known as X Windows, X, and X11, is an architecture commonly used on Linux, UNIX, and Mac OS X that abstracts graphical devices to allow device independence and remote use of a graphical user interface, including display, keyboard, and mouse. X does not include a windowing system—that is delegated to a window manager, such as KDE or Gnome. X is based on an MIT project and is now managed by the X.Org Foundation. It is available as open source software based on the MIT license. X client applications exist on Linux, UNIX, Mac OS X, and Windows. The X server is a native part of most Linux and UNIX systems and Mac OS X and can be added to Windows with the Cygwin platform. The X system was designed to separate server and client using the X protocol and lends itself well to cloud computing. X Windows is complex and can involve some troubleshooting, but because it supports many varied scenarios for its use, it has enjoyed a long life since it was first developed in 1984.

The Remote Desktop Service is a utility that enables users to use a Microsoft Windows graphical user interface on a remote computer. Remote Desktop Service makes use of the Remote Desktop Protocol (RDP), a protocol that Microsoft developed to support it. Client implementations exist for Windows (including most variants, such as Windows 7, XP, and Mobile), Linux, UNIX, Mac OS X, and others. Remote Desktop Service was formerly known as Terminal Services. The Remote Desktop Service implementation of RDP is highly optimized and efficient over remote network connections.

In addition to X and RDP, two other remote graphical user interface platforms worth mentioning are Virtual Network Computing (VNC) and the NX Client. VNC is a system that uses a remote control paradigm that uses Remote Framebuffer Protocol (RBP). Because it is based at the framebuffer level, it can operate on all Windows systems, including Linux/UNIX and Windows. VNC is open source software available under the GNU license. Setup of VNC on a Linux system is described in the section "Linux, Apache, MySQL, and PHP" in Chapter 2, "Developing on the Cloud."

NX is commercial/open source developed by NoMachine. NX Server and Client are the components of the platform, which operates over an SSH connection. The big advantage of the MoMachine NX system is that it works over a secure channel. NX also compresses the display data and uses a client proxy to make optimal use of network bandwidth. Future versions of the tool from NoMachine might be commercial only, with the FreeNX project producing the open source version. Figure 1.14 shows the basic concepts of VNC and NX.

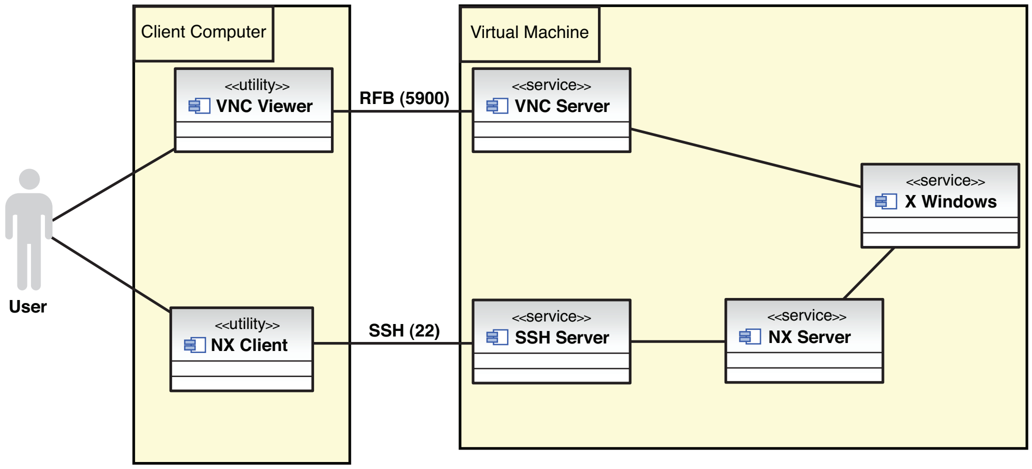


Figure 1.14 Remote Desktop Management with VNC and NX Client

Commercial distributions of NX can support many desktops centrally. Linux desktops, such as KDE and Gnome, work over the top of X to enable users to manage Windows in a multi-tasking environment and personalize their desktop settings. You can use the desktop environment of your choice with either VNC or NX.

In addition to VNC and NX, several open source and commercial implementations of X are available for Microsoft Windows, including Cygwin X server, Hummingbird Exceed, Reflection X, and Xming.

See the sections "Linux, Apache, MySQL, and PHP" in Chapter 2 for basic use of VNC and the section "Remote Desktop Management" in Chapter 6, "Cloud Services and Applications," for more details on using virtual desktops.

This page intentionally left blank

CHAPTER 2

Developing on the Cloud

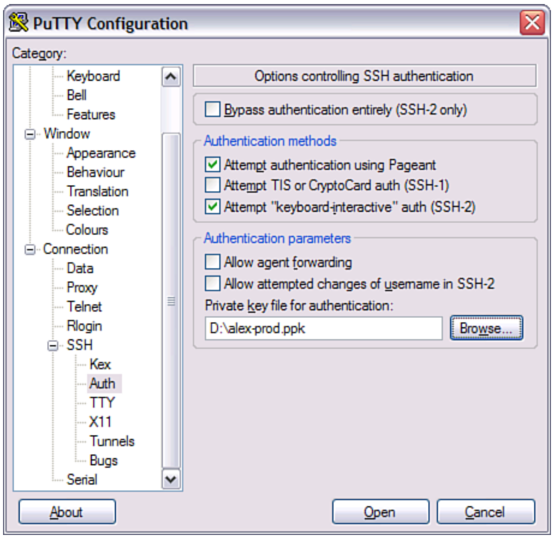
This book is geared to developers, so we have written this chapter to give application developers some tools for application development. Even if you don't use these tools, you might learn something by experimenting with the methods described here.

Linux, Apache, MySQL, and PHP

The so-called LAMP stack is a great open source platform for running web applications. To set up a LAMP environment, first provision a Red Hat Enterprise Linux 5.4 64-bit copper instance on the IBM Cloud. Many other Linux images will do just as well. We also describe some basic tools for working with Linux in the cloud here, including PuTTY and VNC. Some of the settings are specific to Red Hat. The Java 2 Enterprise Edition example that follows uses SUSE Linux Enterprise.

On a Windows client, use the PuTTY SSH client to connect to the Linux server. To import the key, go to Connection, SSH, Auth, click the Browse button, and add your private key, as shown in Figure 2.1.

Enter the host name or IP address and use the other default settings in the main screen (see Figure 2.2). Save it with a name for future use.



Options controlling SSH authentication

[ ] Bypass authentication entirely (SSH-2 only)

Authentication methods

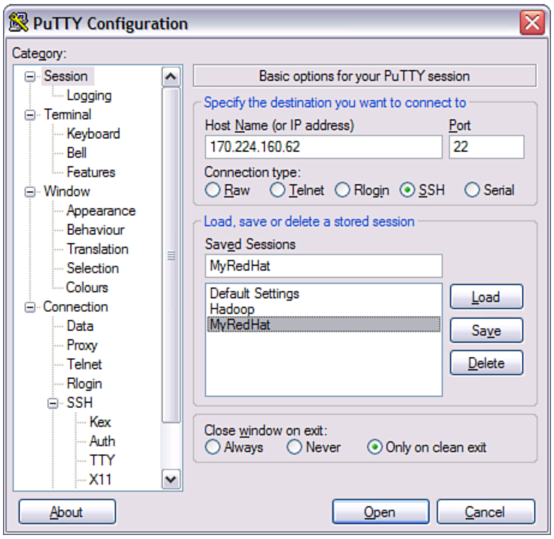
[✓] Attempt authentication using Pageant
[ ] Attempt TIS or CryptoCard auth (SSH-1)
[✓] Attempt "keyboard-interactive" auth (SSH-2)

Authentication parameters

[ ] Allow agent forwarding
[ ] Allow attempted changes of username in SSH-2

Private key file for authentication: D:\alex-prod.ppk

Figure 2.1 PuTTY configuration



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 170.224.160.62
Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: MyRedHat

Default Settings
Hadoop
MyRedHat

Close window on exit: Always Never [✓ Only on clean exit]

Figure 2.2 PuTTY session information

See the “Secure Shell (SSH)” section of Chapter 7, “Security,” for more details on SSH. Click the Open button to log in. Enter the user name, idcuser. No password is required because you are authenticating with your private key. To install software, you often must be acting as root. The image is set up so that you cannot log in as root, which is a good security practice. However, to perform administration work, you can execute `sudo` as idcuser. Type the following command to get root access:

```
> sudo /bin/bash
```

If you have any problems in Linux, the first place to check is the logs directory `/var/log`. The system log messages are stored in the file `/var/log/messages`. To view them as they occur, use the `tail` command:

```
tail -f messages
```

Red Hat Enterprise Linux distributions mostly come with the Apache HTTP server installed. If yours does not, use this command to install it:

```
# yum install apache
```

To start Apache, type this command:

```
# /usr/sbin/apachectl start
```

The image is set up with `iptables` blocking all inbound traffic, which stops you from seeing web pages served by the Apache HTTP server. To allow traffic on port 80, use this command:

```
# /sbin/iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

See the “Services” section in Chapter 6, “Cloud Services and Applications,” for more details on system services. These commands are specific to Red Hat. See the “Java 2 Enterprise Edition” section, later in this chapter, for the basic steps on SUSE Linux Enterprise. See the “Firewalls” section of Chapter 7 for more background on firewalls in general.

Now enter the address of your server into your web browser:

```
http://<ip or hostname>
```

You should see the Apache web server test page. If you have any problems, check the HTTP server logs in `/var/log/httpd`. The Apache configuration file is located at `/etc/httpd/conf.d`.

To install PHP and MySQL, use the yum package install tool to install it from the Red Hat software repository. Using the yum installer is a best practice on Red Hat to check and download the dependencies properly. Type this command:

```
# yum install php-mysql
```

Restart the HTTP server with this command:

```
# /usr/sbin/apachectl restart
```

Now create a basic PHP page to check your PHP installation. You can use vi, edit the file on your local system, or install Emacs. We prefer to use Emacs. Install it with this command:

```
# yum install emacs
```

The document root of the HTTP server is at `/var/www/html`. Create a file called `test.php` using these commands:

```
# cd /var/www/html
# emacs test.php
```

Cut and past the following text into PuTTY:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head><title>PHP Test</title></head>
    <body>
        <?php
          phpinfo();
        ?>
    </body>
</html>
```

`phpinfo()` is a function that lists the PHP installation and configuration settings. Type Ctrl+X Ctrl+S to save the document, and type Ctrl+X Ctrl+C to exit Emacs. Enter this URL in your browser:

```
http://<ip>/test.php
```

You should see a page like Figure 2.3, as shown in the VNC viewer.

A more useful editor for PHP scripts than Emacs is the Eclipse PHP plug-in, which you can download free from the Eclipse Project using the software update installer. It provides a color-coded editor, context completion, and integration with a debugger. Figure 2.4 shows the PHP plug-in configuration panel.

To use the desktop, you need to make a few edits to enable VNC. If it is not installed already, use yum to install VNC:

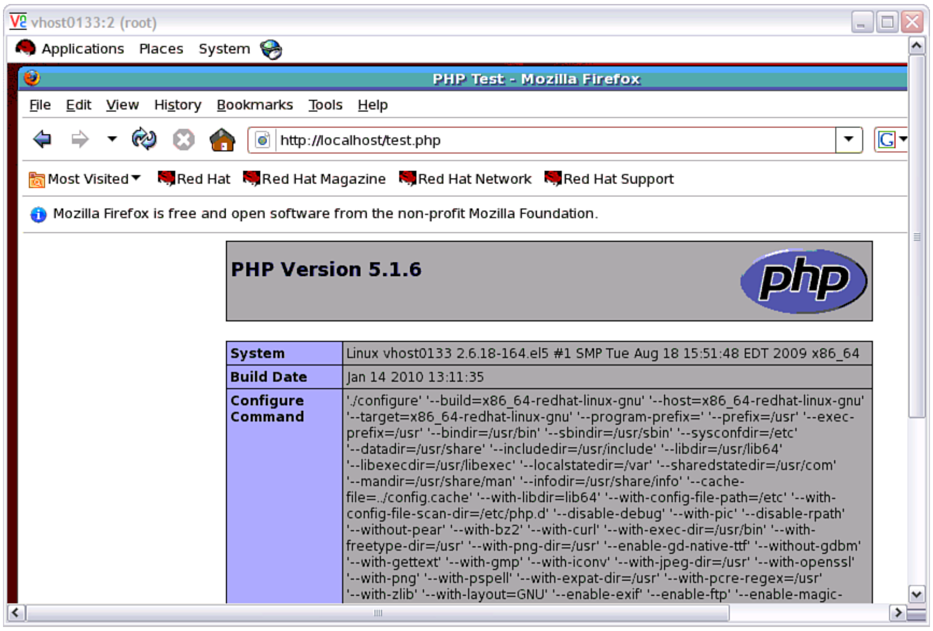
```
# yum install vnc
```

Run this command to set a VNC password:

```
# /usr/bin/vncpasswd
```

Uncomment these two lines in the file `~idcuser/.vnc/xstartup`:

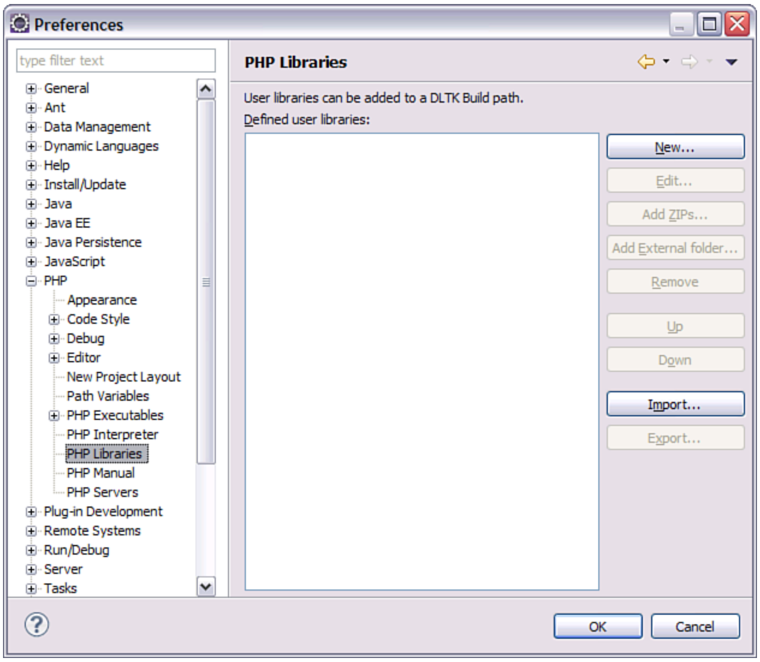
```
unset SESSION_MANAGER
exec /etc/X11/xinit/xinitrc
```


vhost0133:2 (root)
Applications Places System
PHP Test - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/test.php
Most Visited Red Hat Red Hat Magazine Red Hat Network Red Hat Support
Mozilla Firefox is free and open software from the non-profit Mozilla Foundation.

PHP Version 5.1.6

System	Linux vhost0133 2.6.18-164.el5 #1 SMP Tue Aug 18 15:51:48 EDT 2009 x86_64
Build Date	Jan 14 2010 13:11:35
Configure Command	'./configure' '--build=x86_64-redhat-linux-gnu' '--host=x86_64-redhat-linux-gnu' '--target=x86_64-redhat-linux-gnu' '--program-prefix=' '--prefix=/usr' '--exec-prefix=/usr' '--bindir=/usr/bin' '--sbindir=/usr/sbin' '--sysconfdir=/etc' '--datadir=/usr/share' '--includedir=/usr/include' '--libdir=/usr/lib64' '--libexecdir=/usr/libexec' '--localstatedir=/var' '--sharedstatedir=/usr/com' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--cache-file=../config.cache' '--with-libdir=lib64' '--with-config-file-path=/etc' '--with-config-file-scan-dir=/etc/php.d' '--disable-debug' '--with-pic' '--disable-rpath' '--without-pear' '--with-bz2' '--with-curl' '--with-exec-dir=/usr/bin' '--with-freetype-dir=/usr' '--with-png-dir=/usr' '--enable-gd-native-ttf' '--without-gdbm' '--with-gettext' '--with-gmp' '--with-iconv' '--with-jpeg-dir=/usr' '--with-openssl' '--with-png' '--with-pspell' '--with-expat-dir=/usr' '--with-pcre-regex=/usr' '--with-zlib' '--with-layout=GNU' '--enable-exif' '--enable-ftp' '--enable-magic-

Figure 2.3 PHP information page



User libraries can be added to a DLTK Build path.

Defined user libraries:

Figure 2.4 Eclipse PHP plug-in configuration screen

If the `xstartup` file does not exist, it will be created when the VNC server is first run. If VNC is running as a service, you can check the status of the VNC server using this command:

```
# /sbin/service vncserver status
Xvnc (pid 379) is running...
```

This shows that the VNC server is running. If it is not running, use this command to start it:

```
# /usr/bin/vncserver
```

This command also creates a new authority file, `/home/idcuser/.Xauthority`, if it does not already exist. See the section "Remote Desktop Management" in Chapter 6 for more details on VNC.

Add a firewall rule for the ports VNC needs by adding a line to the file `/etc/sysconfig/iptables`:

```
-A RH-Firewall-1-INPUT --protocol tcp --destination-port 5900:5902 -j
ACCEPT
```

This is an alternate method to the command described previously and saves the rule. Restart the firewall with the `service` command, as root:

```
# /sbin/service iptables restart
```

See the "Firewalls" section in Chapter 7 for more on managing firewall rules. Download the VNC client to your PC and start it up, entering the address `<IP>:1` to connect. If you do not achieve success, stop the VNC server, go back and edit the xstartup file, and restart VNC. To stop the VNC server, enter this command:

```
# /usr/bin/vncserver -kill :1
```

To log into VNC as a different user, run the command `vncserver` as that user.

Windows

Working with Microsoft Windows virtual machines is possible on many Infrastructure as a Service (IaaS) clouds. However, some of the tools for making use of virtual machine instances vary. The notes here relate to the IBM SmartCloud Enterprise with Windows 2008 Server.

You can provision a Windows virtual machine instance the same as a Linux instance with the self-service user interface. However, you will be prompted for an administrator user password instead of a Secure Shell key name. After provisioning has completed, you can connect to the Windows instance with Remote Desktop.

Java 2 Enterprise Edition

You can set up a Java 2 Enterprise Edition (J2EE) environment on one of the base OS images, or you can use one that is already preconfigured, such as WebSphere Application Server on Red Hat Enterprise Linux (RHEL) or SUSE Linux Enterprise. This section helps you get started working with a Java 2 Enterprise environment, including setting up a Java SDK, creating a simple application on WebSphere Application Server with data persistence to DB2, and working with messaging and scheduled events. The tools used in this section are leveraged in later chapters of the book for more complex examples.

Java SDK

The RHEL image comes bundled with a JDK. However, the version is old, so you might want to download a newer version from the IBM Java web site. In fact, if you want to use the IBM Smart Business Cloud Enterprise APIs, you need Java SE version 6 (JDK 1.6). Download the 64-bit AMD/Opteron/EM64T or 32-bit x86. *Tip:* Using a browser in VNC directly to download files from the Internet saves double handling, compared with downloading to your local machine and uploading to the cloud. Install the JDK with the following commands:

```
# chmod +x ibm-java-x86_64-jre-6.0-9.1.bin
# ./ibm-java-x86_64-jre-6.0-9.1.bin
```

Check the version with this command:

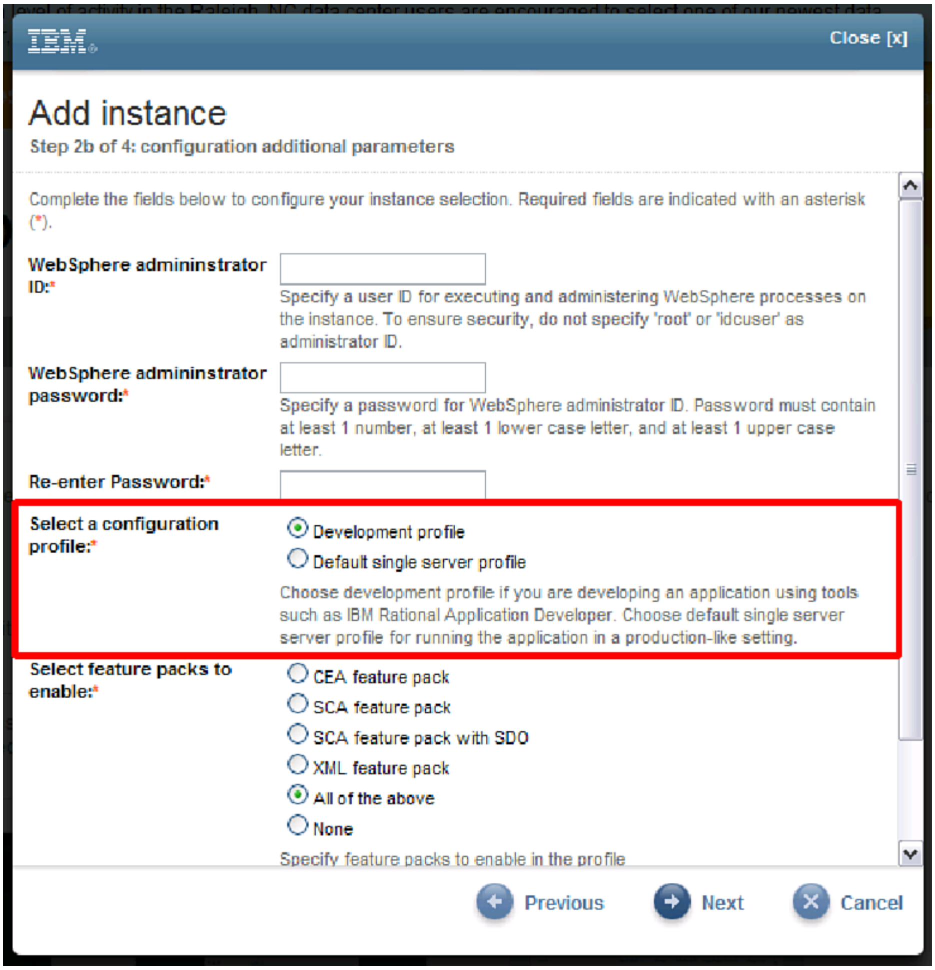
```
# export JAVA_HOME=/opt/ibm/java-x86_64-60/jre
# $JAVA_HOME/bin/java -version
```

If you do not adjust the TCP window scaling, you might encounter some difficulties using some API functions (mainly the `describeInstance` method) because of the large amount of data generated from the instances in the cloud itself. This is a known problem for some Linux versions with certain default settings. If you have this problem, you need to modify the `tcp_window_scaling` parameter with the `sysctl` command (which modifies kernel parameters at runtime), restart network services, and set `tcp_window_scaling` to `0` in the `/etc/sysctl.conf` configuration file.

WebSphere Application Server

The IBM SmartCloud Enterprise catalog includes instances of WebSphere Application Server (WAS) 7.0. With this, you can use a J2EE environment without needing to install any software and can also enjoy the extra features that WAS provides above other J2EE environments. When requesting a WAS server, you are prompted for a user name and password for the WAS administration console. Write these down; you will need them later. In addition, you are prompted to choose the profile, as shown in Figure 2.5.

A WebSphere profile defines a runtime environment, including all the files that the server processes to determine and store configuration information. Choose the Development profile. It makes WAS more convenient to manage through IBM Rational Software Architect, as discussed in the following sections. We return to profiles in the section “J2EE Application Performance and Scalability” in Chapter 8, “Performance, Availability, Monitoring, and Metering.”



Close [x]

Add instance

Step 2b of 4: configuration additional parameters

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (*).

WebSphere admininstrator ID:*	Specify a user ID for executing and administering WebSphere processes on the instance. To ensure security, do not specify 'root' or 'idcuser' as administrator ID.
WebSphere admininstrator password:*	Specify a password for WebSphere administrator ID. Password must contain at least 1 number, at least 1 lower case letter, and at least 1 upper case letter.
Re-enter Password:*	
Select a configuration profile:*	[✓ Development profile] Default single server profile. Choose development profile if you are developing an application using tools such as IBM Rational Application Developer. Choose default single server server profile for running the application in a production-like setting.
Select feature packs to enable:*	CEA feature pack, SCA feature pack, SCA feature pack with SDO, XML feature pack, [✓ All of the above], None. Specify feature packs to enable in the profile

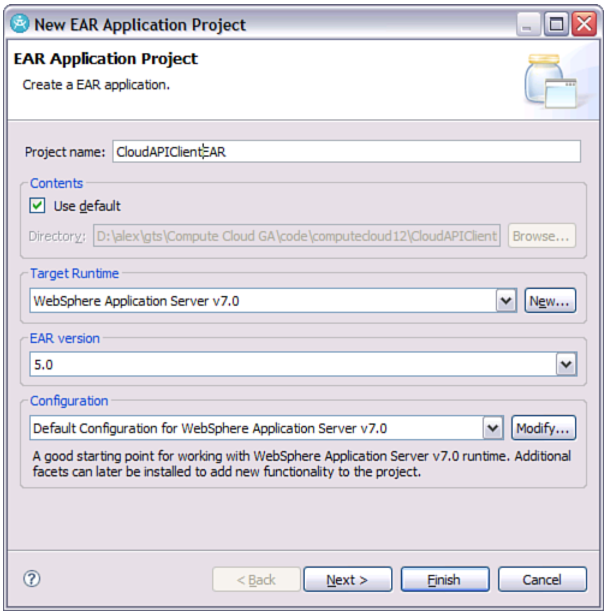
Figure 2.5 WebSphere Image Provisioning Wizard additional parameters screen

The WAS server starts up a short time after the instance is provisioned and has booted up. You can bring up the administration console with this URL:

```
https://host:9043/ibm/console/logon.jsp
```

You can do all the management of WAS, except for starting and stopping the server itself, from the web administration console. You can use IBM Rational Application Developer, Eclipse, or other integrated development environments (IDEs) to create J2EE applications that can run in WAS. For Eclipse, use *J2EE Standard Tools*, also known as the *J2EE Developer Tools* version. You can also use the IBM Rational Application Developer images in the Cloud catalog, to take advantage of a prebuilt environment.

To try it in Rational Application Developer (RAD) or Rational Software Architect (RSA), start a new Enterprise Application in Eclipse using the EAR Application Project wizard, as shown in Figure 2.6.



EAR Application Project

Create a EAR application.

Project name: CloudAPIClientEAR

Contents

✓ Use default

Directory: D:\alex\gts\Compute Cloud GA\code\computecloud12\CloudAPIClient Browse...

Target Runtime

WebSphere Application Server v7.0 New...

EAR version

5.0

Configuration

Default Configuration for WebSphere Application Server v7.0 Modify...

A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

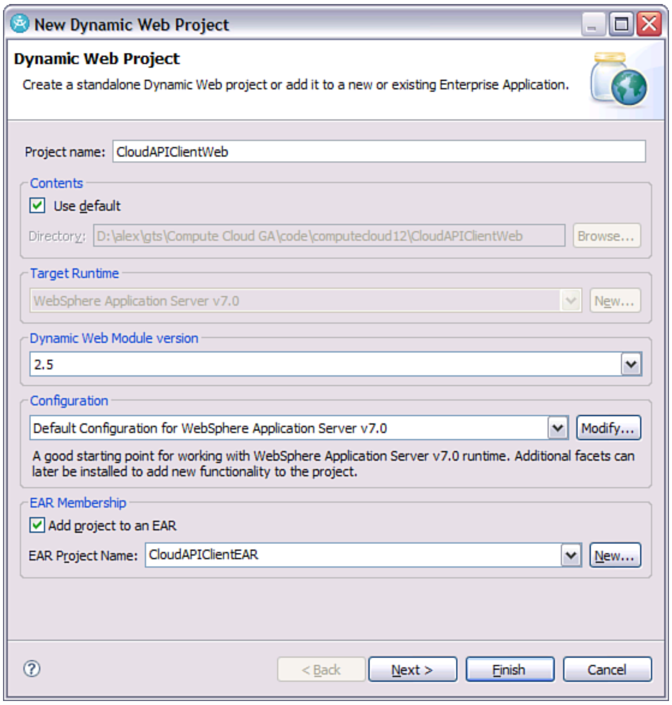
Figure 2.6 Eclipse EAR Application Project Wizard

Call the project `CloudAPIClientEAR`. Click the Finish button. Next, create a Dynamic Web Project and call it `CloudAPIClientWeb`. Add the Dynamic Web project to the `CloudAPIClientEAR` project, as shown in Figure 2.7.

Now create an HTML file and a servlet to test the application. Add a file called `index.html` to the web content directory with some text such as "My Web Application." Also add a servlet with code such as the class `TestServlet`, shown here:

```
package com.ibm.cloud.examples.servlet;

import java.io.IOException;
import java.io.Writer;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
/**
 * Servlet implementation class TestServlet
 */
public class TestServlet extends HttpServlet {
      private static final long serialVersionUID = 1L;

      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
            Writer writer = response.getWriter();
            writer.write("My web application");
      }
}
```


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: CloudAPIClientWeb

Contents
- [✓] Use default
- Directory: D:\alex\gts\Compute Cloud GA\code\computecloud12\CloudAPIClientWeb

Target Runtime: WebSphere Application Server v7.0

Dynamic Web Module version: 2.5

Configuration: Default Configuration for WebSphere Application Server v7.0

A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

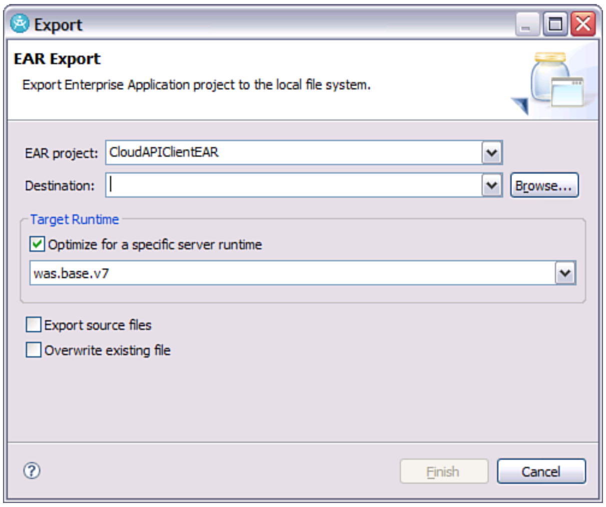
EAR Membership
- [✓] Add project to an EAR
- EAR Project Name: CloudAPIClientEAR

Figure 2.7 Eclipse Dynamic Web Project Wizard

If you used RSA or Eclipse, your web application archive (war) `web.xml` file should be already configured for you. If not, make it look like this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
      <display-name>CloudAPIClientWeb</display-name>
      <servlet>
            <display-name>TestServlet</display-name>
            <servlet-name>TestServlet</servlet-name>
            <servlet-class>com.ibm.cloud.examples.servlet.
TestServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>TestServlet</servlet-name>
            <url-pattern>/TestServlet</url-pattern>
      </servlet-mapping>
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

Next, export from RSA by right-clicking the EAR project and choosing Export to EAR. Name the file `CloudAPIClientEAR.ear`, as shown in Figure 2.8.



Export Enterprise Application project to the local file system.

EAR project: CloudAPIClientEAR

Destination:

Target Runtime

[✓] Optimize for a specific server runtime

was.base.v7

[ ] Export source files

[ ] Overwrite existing file

Figure 2.8 Rational Software Architect EAR Export Wizard

To add your enterprise application EAR file, follow these steps:

1. Go to the WebSphere administration console in your browser.
2. Navigate to Applications, New Application, New Enterprise Application, and choose Upload from Local File System. Upload the EAR file and click the Next button.
3. Choose the Fast Path option.
4. Choose any directory to install the application.
5. In the Map Modules to Servers panel, select CloudAPIClientWeb.
6. Click Save Changes.
7. Go to Applications, Application Types, WebSphere enterprise applications. Start the application.

Navigate to the following URL to view the HTML page:

```
http://host:9080/CloudAPIClientWeb/
```

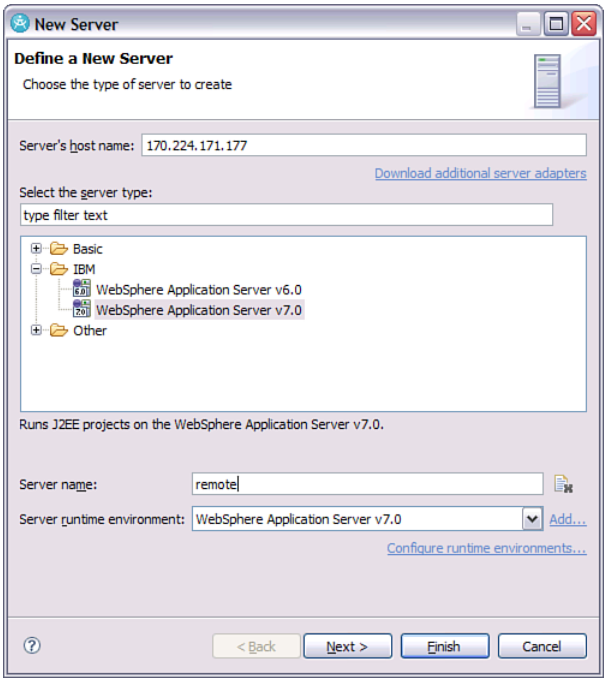
You should see the text "My web application" added to the HTML page. To see the servlet, enter the following the URL:

```
http://host:9080/CloudAPIClientWeb/TestServlet
```

You should see the text that printed from the Writer in the previous TestServlet.

Deploying in this way can get old if you do it more than a few times. You can save a lot of time by using the remote deployment capabilities in Eclipse or IBM Rational Application Developer (RAD). Using either automatically publishes your J2EE application to the server instead of requiring you to step through all the screens in the WebSphere administration console. Use the New Server Wizard to add the remote server; you must supply the host name or IP address and administration user ID and password. Figure 2.9 shows the New Server Wizard from RAD.

You might need to use the Linux desktop to perform certain tasks. See the section "Remote Desktop Management" in Chapter 6 for more details on this topic.


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Basic
IBM
WebSphere Application Server v6.0
WebSphere Application Server v7.0
Other
Add...
Configure runtime environments...
< Back
Next >
Finish
Cancel

Server's host name: 170.224.171.177

Select the server type: type filter text

Runs J2EE projects on the WebSphere Application Server v7.0.

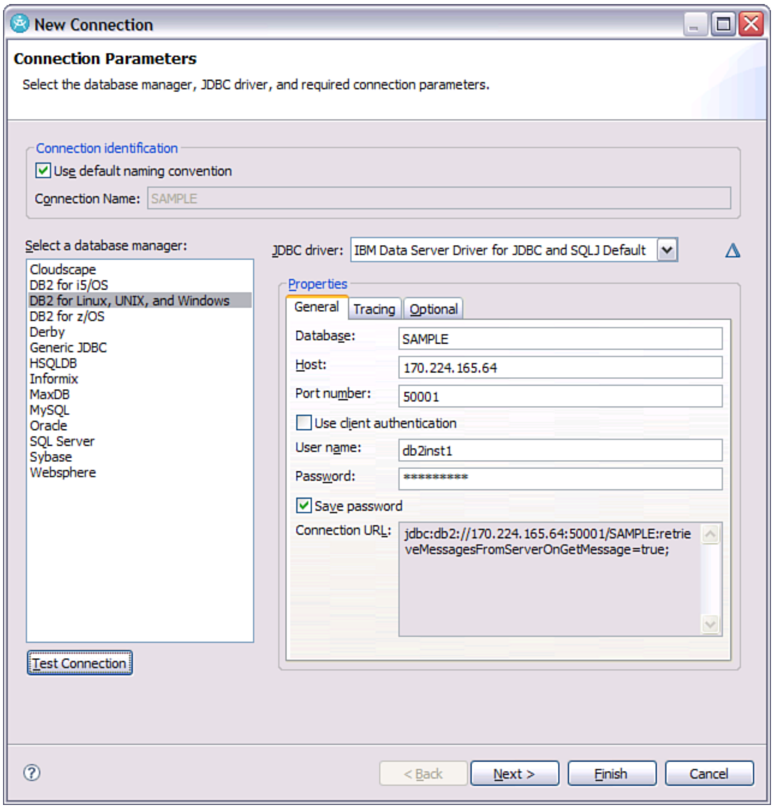
Server name: remote

Server runtime environment: WebSphere Application Server v7.0

Figure 2.9 New Server Wizard in IBM Rational Application Server

Relational Database

Most enterprise and web applications need a relational database to store data. In this section, we look at using the IBM DB2 Enterprise database. Images are available for DB2 Enterprise in the catalog. We use the *IBM DB2 Enterprise Developer Ed SLES 9.7.1–BYOL* image to demonstrate. (It has an image ID of 20005750.) The "Getting Started" page for the image lists the different ports and other parameters used. Add a firewall rule allowing traffic through port 50002 using YAST, as explained in the Chapter 7 section "Firewalls." You can create a connection to the SAMPLE database from your Eclipse or Rational Application Developer environment using the New Connection Wizard in Data view. Figure 2.10 shows this in Rational Application Developer.



New Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: SAMPLE

Select a database manager:

- Cloudscape
- DB2 for i5/OS
- DB2 for Linux, UNIX, and Windows
- DB2 for z/OS
- Derby
- Generic JDBC
- HSQLDB
- Informix
- MaxDB
- MySQL
- Oracle
- SQL Server
- Sybase
- Websphere

JDBC driver: IBM Data Server Driver for JDBC and SQLJ Default

Properties

Database:	SAMPLE
Host:	170.224.165.64
Port number:	50001
☐ Use client authentication	
User name:	db2inst1
Password:	*********
☑ Save password	
Connection URL:	jdbc:db2://170.224.165.64:50001/SAMPLE:retrieveMessagesFromServerOnGetMessage=true;

Figure 2.10 New Connection Wizard in Rational Application Developer

Enter the IP address from the IBM Cloud Enterprise Control Panel, 50001 for the port, db2inst1 for the user name, and the password that you supplied when creating the DB2 instance. Click the Test Connection button to test connectivity. We show how to use the DB2 command line in the next section.

The database needs to run from a local disk, for I/O performance reasons. However, because the local disk is not persistent, you need to back up the database somewhere else. An attached disk is probably the best option for backup location initially. Many tools can help back up data. In development mode, data-movement tools such as `copy`, `db2move`, `import`, and `export` might be the most convenient to work with at first. For example, this command exports the database `{dbname}` to a file:

```
> db2move {dbname} export
```

The DB2 command-line tool is discussed in the section, “Business Scenario: Developing the IoT Data Portal,” later in this chapter.

In operations mode, you probably will also use the backup, recover, and restore utilities. You can easily automate the backup utility, and you can use the rollforward utility to save changes made. You can also use the HADR function that is built into DB2 for high availability, to minimize any data lost if a machine crashes.

DB2 supports different types of clients. The Rational Database Connection Wizard previously used the DB2 JDBC driver to connect. The examples that follow also use JDBC. The DB2 JDBC drivers are located in this directory:

```
/opt/ibm/db2/V9.5/java
```

You may have a different version of DB2. The DB2 JDBC driver is a type 4 driver, so to install it, you only need to copy the JAR file. The driver requires a JDK 1.4.2 or later. We use the `db2jcc4.jar` driver, which includes JDBC 4.0 functions; JDK 6 is required for this. You will need also `db2jcc_license_cisuz.jar` file on the classpath.

DB2 also has drivers for ODBC, .NET, PHP, and other platforms, but we do not discuss those.

DB2 also comes with a native client (IBM Data Server Runtime Client) and administrative console (IBM Data Server Client). You can access these using remote desktop where DB2 is installed, or you can install them locally on your own desktop computer. The most convenient way to manage DB2 in a cloud environment (at least for the examples we are running) is via the DB2 Command Line Processor (CLP) client. To use this, connect to the DB2 server instance with SSH and enter these commands:

```
> sudo su - db2inst1
> db2 -t
```

The script first changes to the dbinst1 user, who is the owner of the database instance with the environment of that owner (`su -` option). Then we start up a DB2 command-line client. The `-t` option enables multiline commands terminated by a semicolon.

Data Persistence

Several options provide the capability to store and retrieve data from relational databases in J2EE applications. The original way of accessing relational data is to use the Java Database Connectivity (JDBC) API. With this approach, you have to write SQL statements and create a mapping to Java objects. Object-relational mapping is a paradigm developed to save you from having to hand-craft code for translating. The Java Persistence API (JPA) does object-relational mapping in a way that is part of the J2EE standard. It also hides some of the specifics of dealing with different databases, at least for standard applications.

The JPA EntityManager class provides a way to store, update, insert, and delete objects in the database. XML configuration files specify the mapping from the SQL schema to Java objects. Alternatively, you can use Java annotations. JPA provides a query language that enables you to query the data in the database. WebSphere uses the Apache OpenJPA implementation of the JPA specification. You can use JPA both within and outside a J2EE application. The minimum Java version is Java 5 SE.

For applications outside a J2EE container, the connection parameters can be set in a `persistence.xml` file in the `META-INF` directory of the JAR file. Consider this example:

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="openjpa">

<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</prov
ider>
    <class>com.ibm.cloud.examples.iotdata.services.Customer</class>
    <class>com.ibm.cloud.examples.iotdata.services.Device</class>
    <properties>
      <property name="openjpa.ConnectionDriverName"
value="com.ibm.db2.jcc.DB2Driver"/>
      <property name="openjpa.ConnectionUserName" value="db2inst1"/>
      <property name="openjpa.ConnectionPassword" value="xxx"/>
      <property name="openjpa.ConnectionURL"
value="jdbc:db2://host:50001/iotdata"/>
      <property name="openjpa.Log" value="DefaultLevel=WARN,
Tool=INFO"/>
    </properties>
  </persistence-unit>
</persistence>
```

In this file, the classes `Customer` and `Device` are the classes to be enhanced for JPA access. These example classes are described later in the section “Business Scenario: Developing the IoT Data Portal.” The connection properties are listed. The database name given in the JDBC URL is *`iotdata`*. The entry point for JPA programming is the `EntityManagerFactory` class.

```
package com.ibm.cloud.examples.iotdata.services;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerFactoryHolder {

      private static EntityManagerFactoryHolder instance = new
EntityManagerFactoryHolder();
      private EntityManagerFactory entityManagerFactory;
```

```
        private EntityManagerFactoryHolder() {}

        public static EntityManagerFactoryHolder getInstance() {
              return instance;
        }

        public EntityManagerFactory getEntityManagerFactory() {
              if (entityManagerFactory == null) {
                    entityManagerFactory =
  Persistence.createEntityManagerFactory("openjpa");
              }
              return entityManagerFactory;
        }
  }
```

The `EntityManagerFactory` class is an expensive resource, and only a single instance of it should exist. The `EntityManagerFactoryHolder` class is a singleton class and maintains a single instance of `EntityManagerFactory`.

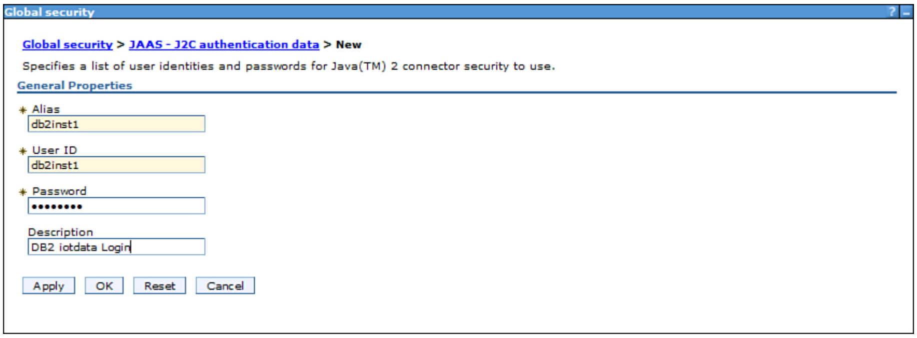
For running outside the container, you need to copy the OpenJPA library and the DB2 database driver to the classpath. You can download the OpenJPA library from the Apache OpenJPA site. Copy the DB2 driver from the DB2 instance. Copy the `db2jcc4.jar db2jcc_license_cisuz.jar` files from the DB2 server using the SCP commands explained in the "Secure Shell (SSH)" section in Chapter 7.

```
# scp -i mykey db2server:/opt/ibm/db2/V9.5/java/db2jcc4.jar
driver/db2jcc4.jar
# scp -i mykey db2server:/opt/ibm/db2/V9.5/java/db2jcc_license_cu.jar
driver/db2jcc_license_cu.jar
```

Here, `db2server` is the host name or IP address of the DB2 server, and `driver` is the local directory where you want to copy the files.

Running inside the WebSphere Application Server, you need to configure a JDBC data source. First, create a J2C authentication alias by following these steps.

1. Select Security, Global Security from the left menu.
2. Click the J2C Authentication data link.
3. Enter the user name and password, as shown in Figure 2.11.



Global security > JAAS - J2C authentication data > New

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

General Properties

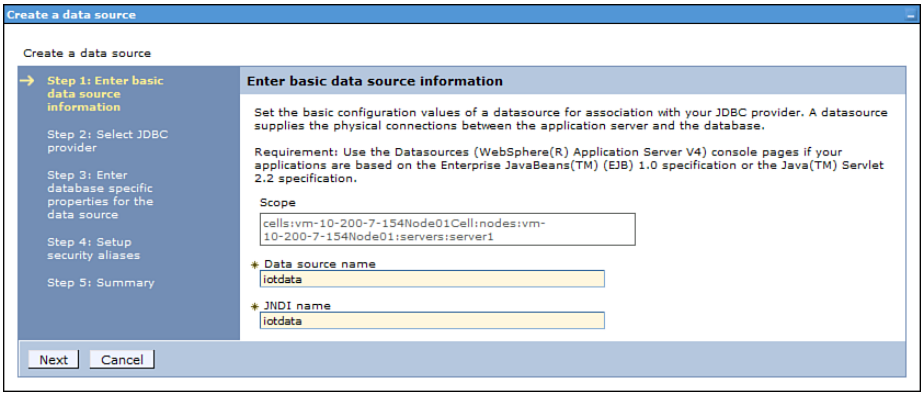
* Alias: db2inst1
* User ID: db2inst1
* Password: ••••••••

Description: DB2 iotdata Login

Figure 2.11 Creating a new WebSphere authentication alias

To set the JPA default provider and create the data source, follow these steps:

1. Navigate to Servers, Server Types, WebSphere Application Server, on the left menu, and then go to server1, Container Services, Default Java Persistence API settings.
2. Select org.apache.openjpa.persistence.PersistenceProviderImpl as the default JPA provider.
3. Click Related Items, Data Sources. Select Node ...Server=server1 and click New button to add a new data source. Select the options shown in Figure 2.12 for Data Source Name and JNDI.


Create a data source
Create a data source
Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Setup security aliases
Step 5: Summary

Enter basic data source information

Set the basic configuration values of a datasource for association with your JDBC provider. A datasource supplies the physical connections between the application server and the database.

Requirement: Use the Datasources (WebSphere(R) Application Server V4) console pages if your applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

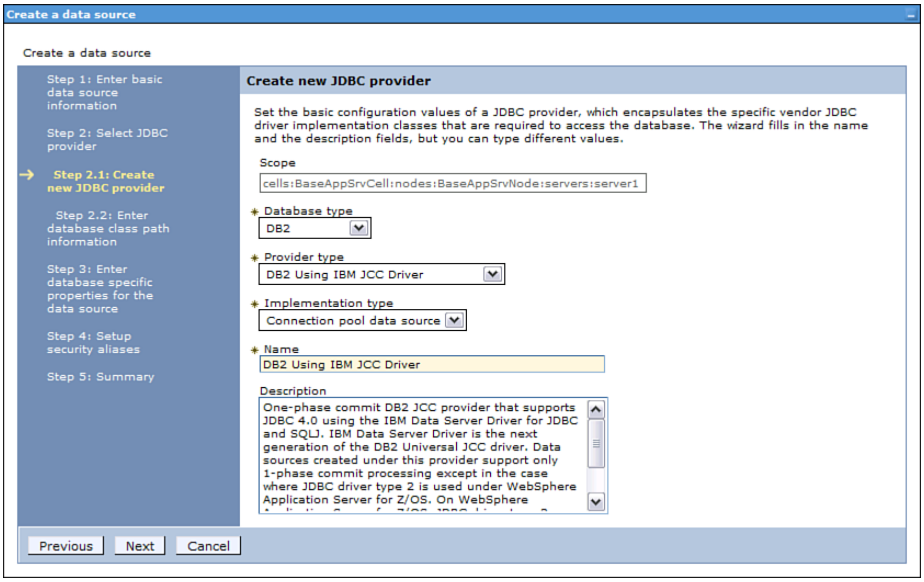
Scope: cells:vm-10-200-7-154Node01Cell:nodes:vm-10-200-7-154Node01:servers:server1

* Data source name: iotdata

* JNDI name: iotdata

Figure 2.12 WAS Console: Data Source Wizard

Both the data source name and the JNDI name are iodata. Click Next and select Create a New JDBC Provider. Enter the data in the next step, as shown in Figure 2.13.



Create a data source

Create a data source

Step 1: Enter basic data source information

Step 2: Select JDBC provider

→ **Step 2.1: Create new JDBC provider**

Step 2.2: Enter database class path information

Step 3: Enter database specific properties for the data source

Step 4: Setup security aliases

Step 5: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope
cells:BaseAppSrvCell:nodes:BaseAppSrvNode:servers:server1

* Database type
DB2

* Provider type
DB2 Using IBM JCC Driver

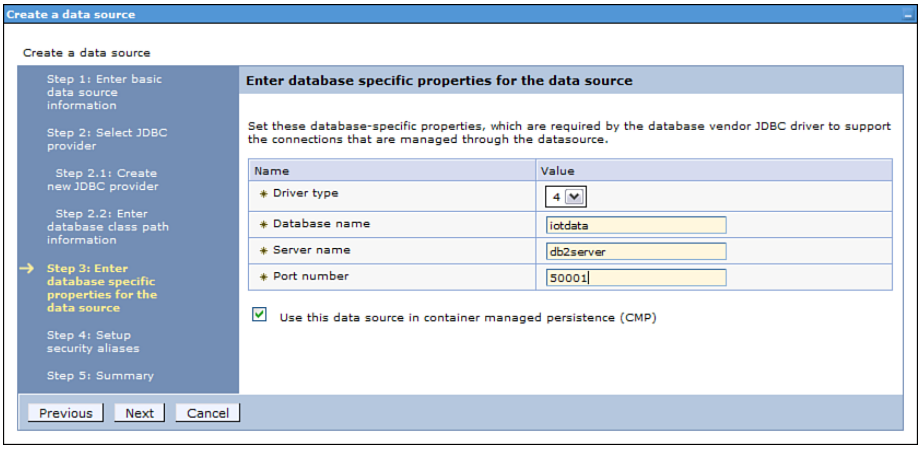
* Implementation type
Connection pool data source

* Name
DB2 Using IBM JCC Driver

Description
One-phase commit DB2 JCC provider that supports JDBC 4.0 using the IBM Data Server Driver for JDBC and SQLJ. IBM Data Server Driver is the next generation of the DB2 Universal JCC driver. Data sources created under this provider support only 1-phase commit processing except in the case where JDBC driver type 2 is used under WebSphere Application Server for Z/OS. On WebSphere

Figure 2.13 WebSphere Application Server New Data Source Wizard

For Database Type, select DB2. Also select DB2 Using IBM JCC Driver and select Connection Pool Data Source. On the next screen, enter the path to which you copied the driver and license file. Enter the database-specific properties that correspond to the database you created earlier.



Create a data source

Create a data source

Step 1: Enter basic data source information

Step 2: Select JDBC provider

Step 2.1: Create new JDBC provider

Step 2.2: Enter database class path information

→ **Step 3: Enter database specific properties for the data source**

Step 4: Setup security aliases

Step 5: Summary

Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through the datasource.

Name	Value
* Driver type	4
* Database name	iotdata
* Server name	db2server
* Port number	50001

☑ Use this data source in container managed persistence (CMP)

Figure 2.14 WebSphere Data Source Wizard: database-specific properties

Select the J2C Authentication Alias that you created previously. Finally, click Finish. After you have finished adding the JDBC provider and data source, also set the default JPA provider properties to use this data source.

Messaging

Messaging is an important concept in cloud computing. It is typically used when some action will take longer than a user can conveniently wait for a page to load, such as a minute or two. Often the cloud service takes care of the messaging for you, such as when provisioning a new virtual machine. However, other times you need messaging in your own programs. For example, suppose that you want to provision a storage volume and then add that storage to a virtual machine that you provision. You must wait for the storage to be provisioned and in an available state before you can use it in a virtual machine provisioning request. This is an example in which you can use messaging to track long-running operations. In many more examples, you might take longer to perform an operation than you would expect a user to wait for a page to return. These operations need to be done in the background after the user has submitted the request.

Also, in general, workloads that can be processed in batch mode are good candidates for cloud computing. Because these workloads are not needed rapidly in an interactive mode for users, you can afford to keep the servers for the work offline until they are needed. For example, consider a document indexing job that takes about an hour and needs to be run every day. You can save the virtual machine needed for this work, start it up when it is needed, and shut it down when you finish. Messaging systems are ideal for processing this kind of work. This section focuses on the IBM WebSphere Application Server 7.0 messaging system. However, the concepts also apply to open source implementations, such as Apache Active MQ.

J2EE supports asynchronous messaging via the Java Messaging Service (JMS). JMS enables programs to create, send, receive, and read asynchronous messages. Messages can be delivered in point-to-point or publish/subscribe modes. WebSphere Application Server supports JMS using either an embedded messaging provider or a third-party provider.

A JMS queue is an instance of the JMS interface `javax.jms.Queue` and represents a message destination. Administrators can define a JMS queue using the WebSphere administration console, and clients can look it up using the Java Naming and Directory Interface (JNDI). Message-driven beans can simplify programs responding to events. They eliminate the need for having to poll the message queue for new messages. Applications can use message-driven beans to consume messages asynchronously.

JMS leverages the J2EE Connector Architecture (JCA) to enable different JMS implementations to interoperate. It wraps the JMS client library in a resource adapter. Our message-driven beans can use an `ActivationConfigProperty` annotation attributes that relate to ActivationSpec objects.

In WebSphere 7.0, a service integration bus provides asynchronous messaging services as a JMS 1.1 provider. The messaging engine is a component of the service integration bus that handles messages submitted to the bus and requests to retrieve messages from the bus. The service integration bus can be distributed over several servers or clusters. Bus members are the individual servers or clusters that participate in the bus and include message stores where the messages are saved. Figure 2.15 illustrates these concepts.

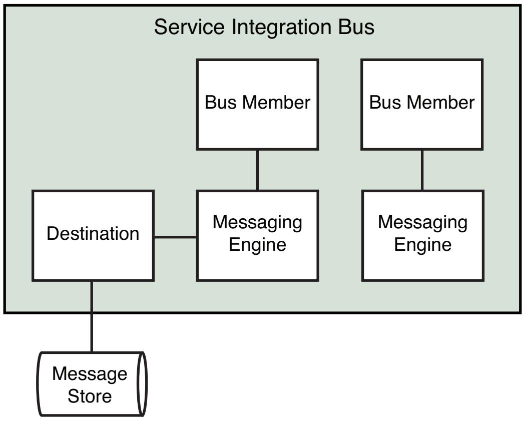
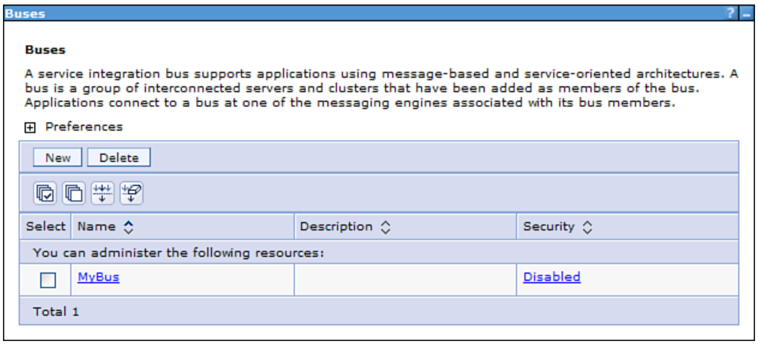


Figure 2.15 WebSphere service integration bus schematic diagram

Follow these steps to add a bus:

1. Select Service integration, Busses on the left menu.
2. Click New.
3. Enter the name MyBus; deselect the Bus Security option.
4. Click Next.
5. Click Finish.

The results should look Figure 2.16.



Buses

Buses

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

Select	Name	Description	Security
You can administer the following resources:			
☐	MyBus		Disabled
Total 1			

Figure 2.16 WebSphere service integration bus management

Follow these steps to add a bus member:

1. Select Service Integration, Buses.
2. Click the MyBus you just created.
3. Under Additional Properties, click Bus Members.
4. Click Add.
5. Select Server in the first step of the wizard.
6. Select File Store as the message store type.
7. Enter 10 MB for the log size, 20 MB for the minimum store size, and 50 MB for the maximum store size. Accept the other configuration defaults.
8. Accept the performance tuning parameters.
9. Click the Finish button.
10. Click Save.

Out of the box, WebSphere does not start a messaging server. However, if you add a bus member, it starts the messaging server when WebSphere starts up.

Restart WebSphere now to start up with the messaging engine running. Use the following commands:

```
# /opt/IBM/WebSphere/AppServer/profiles/AppSrv01/bin/stopServer.sh
server1
# /opt/IBM/WebSphere/AppServer/profiles/AppSrv01/bin/startServer.sh
server1
```

You are prompted for the administrator password when shutting down the server.

Next, add a destination with these steps:

1. Select Service Integration, Buses.
2. Click the MyBus link.
3. Under Additional Properties, click the Destinations link.
4. Click the New button.
5. Select Queue and then click Next.
6. Enter the text ‘MyQueue’ for the name.
7. Select the bus member that you created previously.
8. Click the Finish button.

Check the messaging engine with these steps:

1. Select Service Integration, Buses.
2. Click the IBASampleBus link.
3. Under Additional Properties, click Messaging Engines.

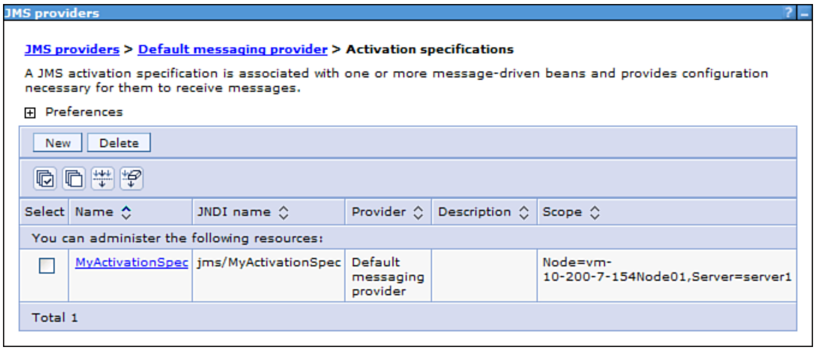
You should see a green arrow next to the messaging engine.
Next, add a JMS queue connection factory with these steps:

1. Add queue.
2. Restart WebSphere.
3. Add activation spec: jms/MyActivationSpec.

To add an activation specification, follow these steps:

1. Select Resources, JMS, Activation Specifications from the left menu.
2. Select Scope to Node … Server=server1.
3. Click the New button.
4. Select the default messaging provider.
5. In the General Properties panel, enter a name of ConnectionFactory, a JNDI name of jms/ConnectionFactory, a destination type of Queue, a destination JNDI name of jms/requests, and a bus name of MyBus.
6. Click Finish.

The result should look like Figure 2.17.



JMS providers > Default messaging provider > Activation specifications

A JMS activation specification is associated with one or more message-driven beans and provides configuration necessary for them to receive messages.

⊞ Preferences

Select	Name	JNDI name	Provider	Description	Scope
You can administer the following resources:					
☐	MyActivationSpec	jms/MyActivationSpec	Default messaging provider		Node=vm-10-200-7-154Node01,Server=server1
Total 1					

Figure 2.17 WebSphere administration console: activation specifications

The previous steps are needed to host the application code in the upcoming section "Business Scenario: Developing the IoT Data Portal."

Scheduled Events

Provisioning requests submitted to the clouds take a finite time to process, depending on the size of the instance requested and various other factors. The entities go through a lifecycle, including states that they cannot be accessed in. For example, on the IBM SmartCloud Enterprise, a new instance goes through the NEW and PROVISIONING states before reaching an ACTIVE state, when it can be used. For these reasons, it might be necessary to be able to check on their status at scheduled intervals. In stand-alone programs, you can use the `java.util.concurrent` utilities to do this; in WebSphere, you can use the WebSphere scheduler service. Following is an example of a stand-alone program that uses the `java.util.concurrent` utilities to schedule a job at 5-minute intervals for an hour:

```
package com.ibm.cloud.examples.schedule;

import java.util.concurrent.*;

public class Scheduler {
      private static int count = 0;

      public static void main(String[] args) {
            ScheduledExecutorService scheduler =
Executors.newSingleThreadScheduledExecutor();
            final Runnable iaaSJob = new Runnable() {
                  public void run() {
                        System.out.println("Check status or submit
request " + count);
                        count++;
                  }
            };
            // Schedule at 5 minute intervals
            final ScheduledFuture<?> handle =
            scheduler.scheduleAtFixedRate(iaaSJob, 0, 5,
TimeUnit.MINUTES);
            // Cancel after an hour
        scheduler.schedule(new Runnable() {
                public void run() { handle.cancel(true); }
            }, 60, TimeUnit.MINUTES);
      }
}
```

The program uses the `Executors` class to create a single-thread `ScheduledExecutorSer-vice` object. The variable `iaaSJob` implements the `Runnable` interface and prints a message for demonstration purposes. It is scheduled to execute at 5-minute intervals and to be canceled after 60 minutes.

Business Scenario: Developing the IoT Data Portal

IoT Data needs to host its web site, which has a web user interface and public API and is data intensive. Many factors need to be considered when choosing a platform for developing and hosting a new web site. No single correct answer exists. The IoT Data web site has one point of complexity: It is data intensive and needs to expand automatically as storage is needed. IoT Data will use Hadoop for data management. This is a Java project, so the choice of hosting leans toward Java.

We chose Java 2 Enterprise Edition hosted on WebSphere Application Server, with the deployment architecture shown in Figure 2.18.

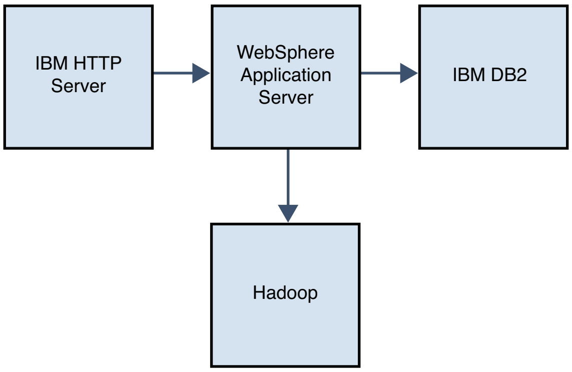


Figure 2.18 IoT Data hosting: deployment architecture

This example uses the Hadoop file system for data storage. This is a scalable file system that can span many servers and locations in a fault-tolerant way. Later sections go into more detail.

Basic customer and device data will be stored in DB2. Log onto the DB2 instance with SSH and execute the following script to create the tables required:

```
> sudo su - db2inst1
> db2 -t
>
> CREATE DATABASE iotdata;
>
> CONNECT TO iotdata;
>
> CREATE TABLE customer (
>  name VARCHAR(80) NOT NULL,
>  id INTEGER GENERATED ALWAYS AS IDENTITY,
>  PRIMARY KEY (id)
> );
>
> CREATE TABLE device (
>  name VARCHAR(80) NOT NULL,
>  id INTEGER GENERATED ALWAYS AS IDENTITY,
>  location VARCHAR(80) NOT NULL,
>  customer_id INTEGER NOT NULL,
>  PRIMARY KEY (id),
>  FOREIGN KEY (customer_id) references customer(id)
> );
```

The script first creates the database `iotdata` and then connects to it. Then you can create the customers and devices tables. Both tables have a `name` and an `id` field; the latter is the primary key. Table devices have a foreign key reference to the customer table to reference the customer to which the device belongs. You can add some sample data to experiment. Enter the following commands in the DB2 command-line tool:

```
> INSERT INTO customer VALUES ('National Weather Bureau', DEFAULT);
> INSERT INTO device VALUES ('Weather Station', DEFAULT, '41', 1);
> INSERT INTO device VALUES ('Radar', DEFAULT, '41', 1);
```

You can check this with these `SELECT` statements:

```
> SELECT * from customers;
> SELECT * from devices;
```

Here we use JPA to manage the data in DB2 from the J2EE application. Entity classes in JPA are plain old Java objects (POJOs). The relationships to database structures can be specified either in XML files or using Java annotations. The following entity class represents the customer table in DB2:

```java
package com.ibm.cloud.examples.iotdata.services;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Customer {

      @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
      private long id;
      private String name;

      public Customer() {}
```

```
        public long getId() {
                return id;
        }

        public void setId(long id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
}
```

The `@Id` annotation indicates that it is the primary key. The `@GeneratedValue` annotation indicates that the database generates the field automatically. The `CustomerService` service can use the `Customer` class, as shown here:

```
package com.ibm.cloud.examples.iotdata.services;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;

/**
 * Service class for accessing customer data
 */
public class CustomerService {


        public void addCustomer(Customer customer) {
                EntityManagerFactory factory =
                        EntityManagerFactoryHolder.getInstance().
getEntityManagerFactory();
                EntityManager em = factory.createEntityManager();
                em.getTransaction().begin();
                em.persist(customer);
                em.getTransaction().commit();
                em.close();
        }
        public static void main(String[] args) {
                CustomerService service = new CustomerService();

                System.out.println("Adding a customer");
                Customer newCustomer = new Customer();
                newCustomer.setName("Air Quality Management District");
                service.addCustomer(newCustomer);
        }
}
```

This adds a customer record to the data store using JPA. After getting an `EntityManager` from the `EntityManagerFactory`, it begins a transaction, adds a customer record, and commits the transaction. The class could be improved by closing the `EntityManager` in a `finally` block. The main block exercises the method with example data. To run the class outside a J2EE container, you need to set JPA Java agent as a JVM argument, as shown here:

```
-javaagent:${JPA_LOCATION}\openjpa-all-2.1.0.jar
```

You can retrieve individual records using the `EntityManager.find` method, as shown in this method:

```
public Customer getCustomer(long id) {
      EntityManagerFactory factory =
      EntityManagerFactoryHolder.getInstance()
.getEntityManagerFactory();
      EntityManager em = factory.createEntityManager();
      Customer customer = em.find(Customer.class, id);
      em.close();
      return customer;
}
```

With these building blocks, you can execute a JPA query to find all the devices in the cloud. Consider an example of this:

```
public List getCustomers() {
      EntityManagerFactory factory =
      EntityManagerFactoryHolder.getInstance()
.getEntityManagerFactory();
      EntityManager em = factory.createEntityManager();
      em.getTransaction().begin();
      Query query = em.createQuery("SELECT c from Customer c");
      List results = query.getResultList();
      em.getTransaction().commit();
      em.close();
      return results;
}
```

The query is similar to SQL, but with some differences. Instead of using a `SELECT * FROM`, you select the results into a Java object. You can exercise this method using the following code:

```
public static void main(String[] args) {
      CustomerService service = new CustomerService();
      System.out.println("Looking up all customers");
      List customers = service.getCustomers();
      for (Object res : customers) {
            Customer c = (Customer)res;
            System.out.println(c.getId() + " : " + c.getName());
      }
}
```

If you run this program, you see output like this:

```
Looking up all customers
1 : National Weather Bureau
2 : Air Quality Management District
```

You can model the device resource with the following Java class:

```
package com.ibm.cloud.examples.iotdata.services;

import javax.persistence.*;

@Entity
public class Device {

      @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
      private long id;
      private String name, location;
      @ManyToOne private Customer customer;

      public Device() { }
      public Customer getCustomer() { return customer; }
      public void setCustomer(Customer c) { this.customer = c; }
      public long getId() { return id; }
      public void setId(long id) { this.id = id; }
      public String getLocation() { return location; }
      public void setLocation(String l) { this.location = l; }
      public String getName() { return name; }
      public void setName(String name) { this.name = name; }
}
```

The `Device` class is similar to the `Customer` class. The main difference lies in the use of a many-to-one entity relationship, which is expressed with the `@ManyToOne` annotation. The class can be used by the `DeviceService` service-tier class, shown here:

```
package com.ibm.cloud.examples.iotdata.services;

import java.util.List;
import javax.persistence.*;

public class DeviceService {

      public void addDevice(Device device) {
            EntityManagerFactory factory =
                  EntityManagerFactoryHolder.getInstance()
.getEntityManagerFactory();
            EntityManager em = factory.createEntityManager();
            em.getTransaction().begin();
            em.persist(device);
            em.getTransaction().commit();
            em.close();
      }

      public static void main(String[] args) {
            DeviceService service = new DeviceService();
            System.out.println("Adding a new device");
            Device newDevice = new Device();
            newDevice.setName("Air Quality Station");
            newDevice.setCustomer(device.getCustomer());
            newDevice.setLocation("41");
            service.addDevice(newDevice);
            }
      }
}
```

This program adds a new device record to the data store. You can list all devices for a customer with this method:

```
public List getDevicesByCustomer(long customerId) {
      EntityManagerFactory factory =
      EntityManagerFactoryHolder.getInstance()
.getEntityManagerFactory();
      EntityManager em = factory.createEntityManager();
      em.getTransaction().begin();
      Query query = em.createQuery("SELECT d FROM Device d WHERE
d.customer.id = " + customerId);
      List results = query.getResultList();
      em.getTransaction().commit();
```

```
            em.close();
            return results;
      }
```

This method uses the JPA query language with a `where` clause.

You can begin to put the web application together by writing a page that lists all the devices that a customer has registered. This is the `doGet` method of the servlet:

```
      protected void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
            String customerIdStr = request.getParameter("customerid");
            long customerId = Long.parseLong(customerIdStr);
            CustomerService customerService = new CustomerService();
            Customer customer = customerService.getCustomer(customerId);
            PrintWriter writer = response.getWriter();
            writeTitle(writer, customerId, "Device List");

            DeviceService deviceService = new DeviceService();
            List devices = deviceService.getDevicesByCustomer(customerId);
            LocationService locationService = LocationService.getInstance();
            writer.println("<table><tbody>");
            writer.println("<tr><th>Device</th><th>Location</th></tr>");
            for (Object res : devices) {
                  Device d = (Device)res;
                  Location l = locationService.getLocation(d.getLocation());
                  writer.println("<tr><td><a href='" + d.getId() + "'>" +
                         d.getName() + "</a></td>");
                  writer.println("<td><a href='" + d.getLocation() + "'>" +
                         l.getName() + "</td></tr>");
            }
            writer.println("</tbody></table>");
            writer.println("<p><a href='NewDeviceServlet?customerid=" +
                        customer.getId() + "'>Add a New Device</a></p>");
            writeFooter(writer);
      }
```

The servlet expects an HTTP parameter called `customerid`, which it uses to look up the customer record and the devices registered for the customer. It writes a header that includes a breadcrumbs trail with links back to the web site home page `index.html` and the customer home page `CustomerServlet`. It iterates over all the devices and prints the name and device location to an HTML table. A `LocationService` looks up the name of the location, based on its ID. This uses an IBM SmartCloud Enterprise API that finds and caches all the relevant locations. We discuss it later in the chapter.

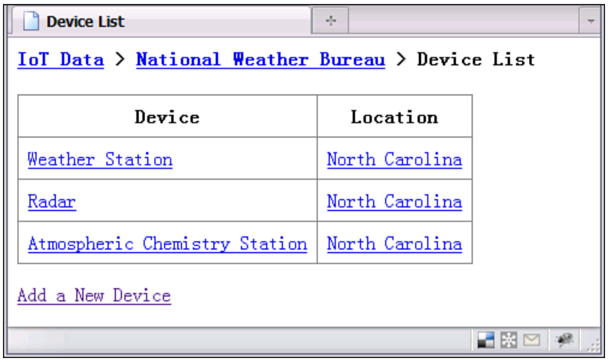
The `persistence.xml` file needs to be updated to take its settings from the container instead of directly in the file. The updated file is shown here:

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="iotdata">

<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</prov
ider>
    <class>com.ibm.cloud.examples.iotdata.services.Customer</class>
    <class>com.ibm.cloud.examples.iotdata.services.Device</class>
    <properties>
      <property name="openjpa.ConnectionFactoryName" value="iotdata"/>
    </properties>
  </persistence-unit>
</persistence>
```

The `EntityManagerFactoryHolder` object needs to be updated to refer to the new `persistence-unit iotdata`. Bring up the page by entering this URL:

```
http://host:9080/CloudAPIClientWeb/DeviceListServlet?customerid=1
```

Figure 2.19 shows this page.



IoT Data > National Weather Bureau > Device List

Device	Location
Weather Station	North Carolina
Radar	North Carolina
Atmospheric Chemistry Station	North Carolina

Add a New Device

Figure 2.19 DeviceListServlet

We continue this scenario in the section “Business Scenario: Using Elastic Cloud Services to Scale” in Chapter 3, “Developing with IBM SmartCloud Enterprise APIs.”

Integration of Application Lifecycle Management Tools with Clouds

In this section we discuss application lifecycle management (ALM) tools and their connection with cloud computing. A recent trend in ALM is building a better connection between development and operations and, in particular, integration of these tools with clouds. We mentioned before that one of the neat things about cloud resources compared to physical resources is that you can create cloud resources on demand with APIs. The emerging generation of ALM tools takes advantage of this. These tools can be run on the cloud, leverage the cloud in some way, and can be used to develop applications for the cloud.

ALM tools help plan, design, collaborate, develop, test, deliver, and maintain software projects. When we scale up to multiperson software projects with build artifacts to deploy into application servers, we need some tools to help us. These tools include the following:

- Requirements management tools to document and track functional and nonfunctional requirements. An example is Rational Requirements Composer.
- Planning tools that help you plan the time and resources needed for a project and help you track the execution of that project. An example is IBM Rational Team Concert™.
- Design tools, such as for creating UML diagrams.
- Source code management system that enables code to be managed centrally. Some examples of open source code management systems are Concurrent Versions System (CVS) and Subversion, and an example of a commercial product that includes a source code management system is IBM Rational Team Concert.
- Integrated development environments (IDEs) for developers to edit and debug source code. Examples are the open source Eclipse IDE and the commercial tool IBM Rational Application Developer.
- Various testing tools to assist in executing and tracking functional, performance, integration, globalization, and regression testing, such as Rational Functional Tester and Rational Performance Tester.
- Defect-tracking systems, such as Bugzilla and IBM Rational Team Concert.
- Ticket-tracking systems for customer support. In open source and cloud projects, forums are replacing these in many cases.

Probably the most fundamental need as a project becomes larger than a single person is the need to share source code in a central repository. CVS is an open source source code–management system that had been popular from many years. It is scalable and can be conveniently used from within integrated development environments using plug-ins. More recently, some improved open source source code–management systems, notably Subversion, have begun to compete with and replace CVS. A commercial alternative with considerably more functionality than these is IBM Rational Team Concert (RTC). The main impact of cloud computing on source code–management tools is that they can be conveniently run on the cloud.

When you get the source code into a central repository, you need to build it into deployable artifacts. You can do this with open source build tools such as Ant or Maven. We discuss this in detail in the section "Build and Deployment Automation," later in this chapter.

When you know how to develop and deliver code in a project team of more than one person, you need to make sure that your code meets requirements. Requirements come in several flavors:

- **Business requirements**—Requirements from specific business stakeholders
- **Marketing requirements**—Business-level requirements that the marketing team creates after analysis of many customers and business stakeholders
- **Functional requirements**—Lower-level requirements to which developers can code
- **Nonfunctional requirements**—Requirements that relate to software in general, such as performance, usability, accessibility, and maintainability
- **Use cases**—A specific way of writing requirements in a way that a detailed design can be more easily derived from
- **Agile stories**—Another specific way of writing requirements so that they can be developed as complete units in agile stories

For medium to large projects, you usually have requirements at multiple levels and may have more than one tool for tracking them. IBM Rational Requirements Composer is a tool for managing requirements that gives stakeholders visibility into the requirements and integrates with development and test tools for traceability of requirements.

After you have collected requirements, it is desirable to create a design to translate this into working software. Software designs can also come at different levels:

- **Conceptual designs**—These document the decisions for use of key technology, including the reasons for adopting the technologies and how those technologies should be applied to the project. The conceptual design should also outline each area of the problem to be solved and describe the approach to be used in developing the software. These designs should include high-level component diagrams.
- **Detailed design**—These document the specific software components to be used, what needs to be developed, public interfaces, and package structure. They should include lower-level UML diagrams.

- **User interface design**—This is a sketch of the user interface to be developed, created by a user interface designer or user experience specialist.

Many tools are useful for the different aspects of software design. The IBM RSA Design Manager, including the Design Manager server, is a tool to enable collaborative design among different participants and stakeholders in the design process. It has both web and desktop user interfaces that support traceability, offer insight into design reasoning, and allow stakeholders to enter comments corresponding to specific areas of the design artifacts. It also supports a design review and approval process, which is important for larger projects.

For quality code development, you need to verify that the code is working properly. Unit testing is where this begins. JUnit is a simple and useful tool for driving unit tests and can give helpful reports. However, it can be difficult to do unit tests when the code integrates with different systems. Mock objects can represent these dependencies. The open source EasyMock tool is useful for this. You will soon find that creating mock objects is an intensive job in itself. Working as a team and sharing resources on the cloud and with ALM tools helps address this.

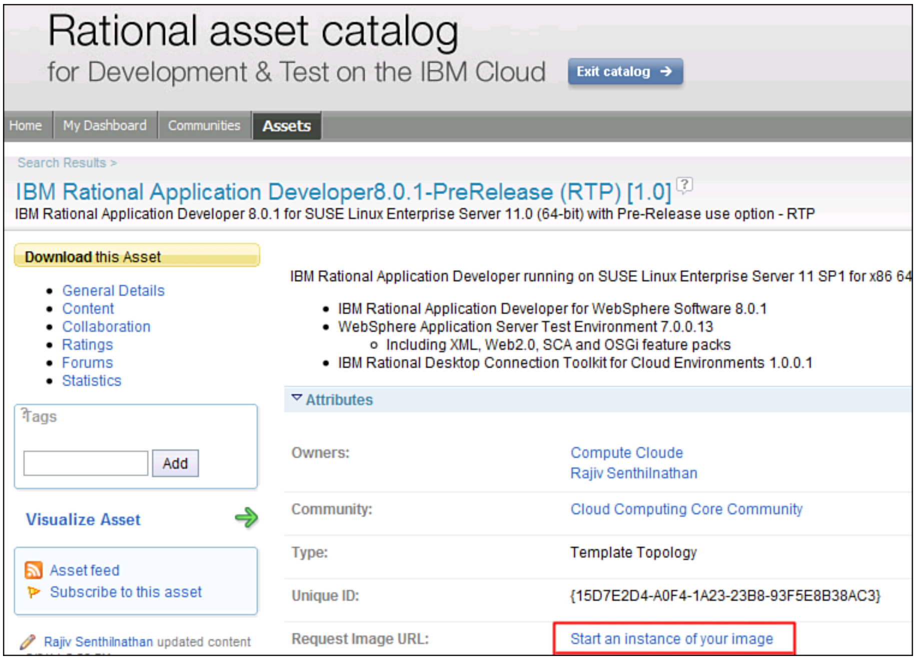
Functional verification testing is testing to verify that the code can execute all the different functions it is supposed to in an operational environment. This is done by following different web screens in a web browser. Ideally, it should be automated by recording the steps the user takes. IBM Rational Functional Tester (RFT) is a commercial product that can be used to do this.

Rational Application Developer

Rational Application Developer is an integrated development environment for Java, J2EE, Web 2.0, and other platforms. The IBM SmartCloud Enterprise has images for Rational Application Developer in the catalog. Rational Application Developer also has a cloud plug-in that enables you to provision and integrate with servers, such as WebSphere Application Server on the cloud. This enables you to achieve a scenario like this:

1. Find the Rational Application Developer Virtual Image in catalog.
2. Select virtual machine properties.
3. Set up a connection to a virtual desktop with NX client.
4. Start up Rational Application Developer on a remote desktop.
5. Add a new WebSphere cloud server.
6. Set cloud credentials.
7. Request a virtual machine for WebSphere on the cloud.

First, find Rational Application Developer in the catalog, as shown in Figure 2.20.


Rational asset catalog

for Development & Test on the IBM Cloud

IBM Rational Application Developer8.0.1-PreRelease (RTP) [1.0]

IBM Rational Application Developer 8.0.1 for SUSE Linux Enterprise Server 11.0 (64-bit) with Pre-Release use option - RTP

IBM Rational Application Developer running on SUSE Linux Enterprise Server 11 SP1 for x86 64

- IBM Rational Application Developer for WebSphere Software 8.0.1
- WebSphere Application Server Test Environment 7.0.0.13
  - Including XML, Web2.0, SCA and OSGi feature packs
- IBM Rational Desktop Connection Toolkit for Cloud Environments 1.0.0.1

Attributes

Owners:	Compute Cloude Rajiv Senthilnathan
Community:	Cloud Computing Core Community
Type:	Template Topology
Unique ID:	{15D7E2D4-A0F4-1A23-23B8-93F5E8B38AC3}
Request Image URL:	Start an instance of your image

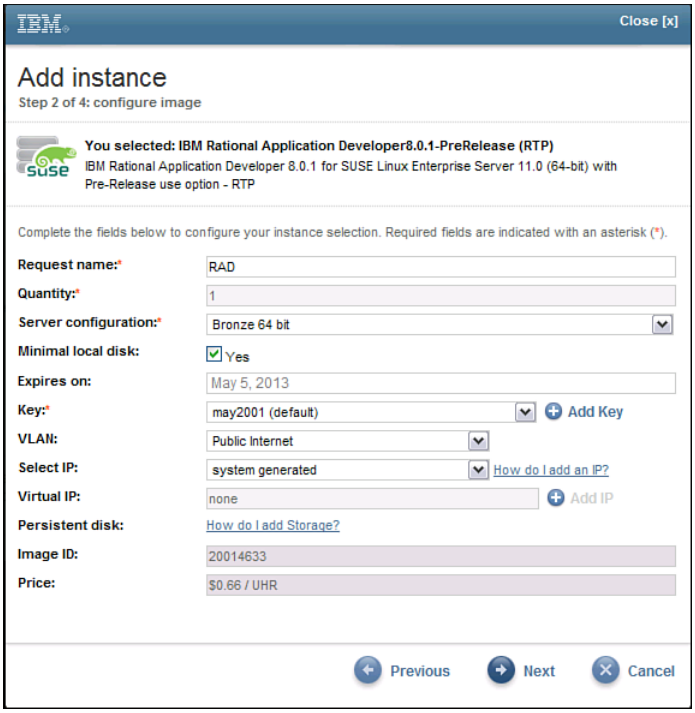
Rajiv Senthilnathan updated content

Figure 2.20 Find Rational Application Developer Virtual Image in catalog

Click the Start an Instance of Your Image link. This brings up the wizard to select the characteristics of the virtual machine, as shown in Figure 2.21.

Click Next. The next steps prompt you for a desktop password. Provision the virtual machine; this takes a few minutes. NX Client is an open source/commercially supported remote desktop client that tunnels over SSH. Download the NX Client from the NoMachine web site. Start the Connection Wizard from the Windows Start menu (or equivalent on Linux). In the Connection Wizard, import and save your SSH key. The key does not need conversion by PuTTYgen. Enter the user name and the password that you entered in the provisioning wizard. You should see a desktop like in Figure 2.22.

If you have trouble connecting via the NX Client, see the section “NX Remote Desktop” in Chapter 6.



Add instance
Step 2 of 4: configure image

You selected: IBM Rational Application Developer8.0.1-PreRelease (RTP)
IBM Rational Application Developer 8.0.1 for SUSE Linux Enterprise Server 11.0 (64-bit) with Pre-Release use option - RTP

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (*).

Request name:*	RAD
Quantity:*	1
Server configuration:*	Bronze 64 bit
Minimal local disk:	✓ Yes
Expires on:	May 5, 2013
Key:*	may2001 (default) Add Key
VLAN:	Public Internet
Select IP:	system generated How do I add an IP?
Virtual IP:	none Add IP
Persistent disk:	How do I add Storage?
Image ID:	20014633
Price:	$0.66 / UHR

Figure 2.21 Provisioning wizard for virtual machine properties

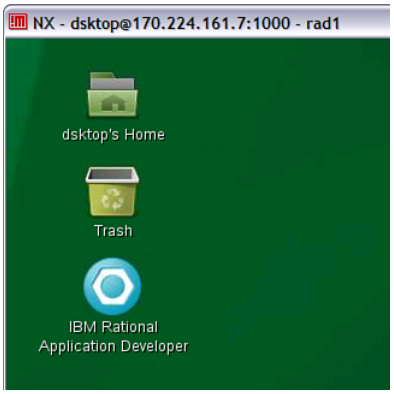
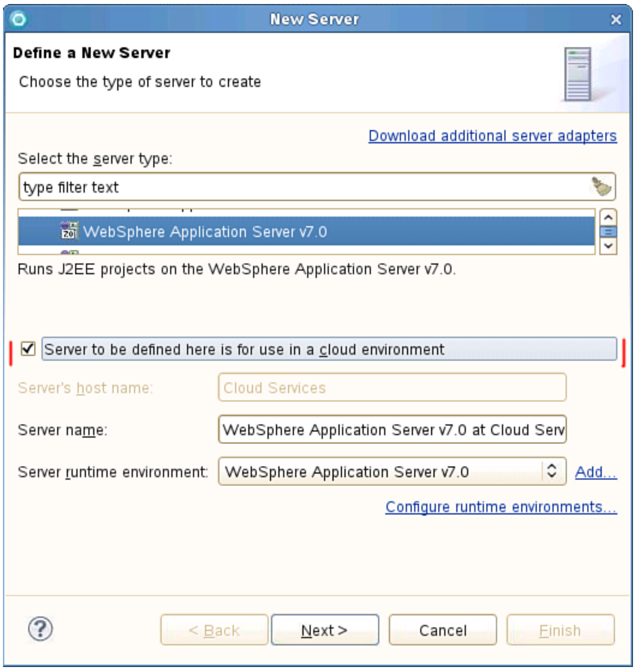

NX - dsktop@170.224.161.7:1000 - rad1
dsktop's Home
Trash
IBM Rational Application Developer

Figure 2.22 Remote Desktop with Rational Application Developer launch icon

Double-click the desktop launch icon. Close the Welcome page. You should see a Servers tab. Right-click in the blank space and select New, Server. Select the check box Server to Be Defined Here Is for Use in a Cloud Environment, as shown in Figure 2.23.


New Server

Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

WebSphere Application Server v7.0

Runs J2EE projects on the WebSphere Application Server v7.0.

☑ Server to be defined here is for use in a cloud environment

Server's host name: Cloud Services

Server name: WebSphere Application Server v7.0 at Cloud Serv

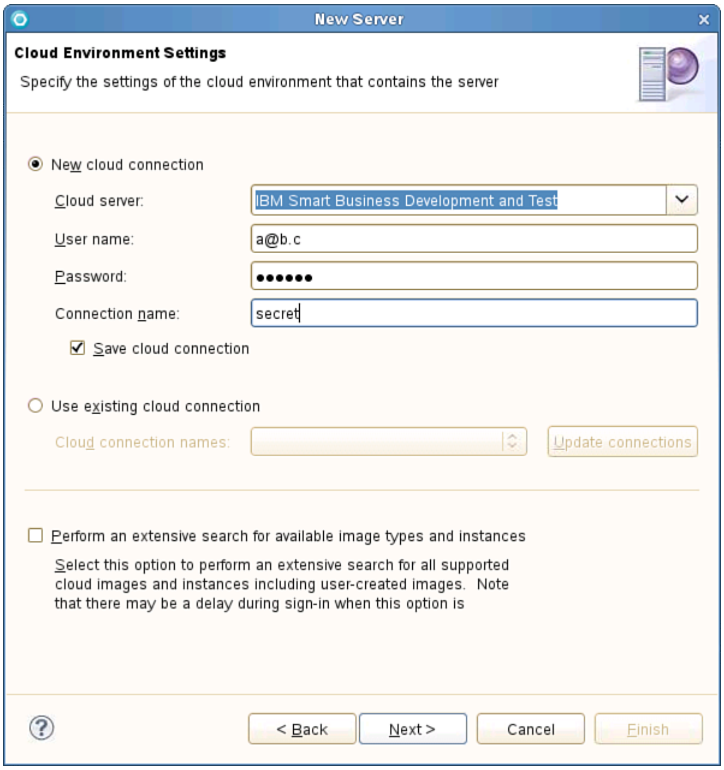
Server runtime environment: WebSphere Application Server v7.0

Figure 2.23 Defining a new cloud server in Rational Application Developer

Click Next. Enter your Cloud user credentials in the next screen, as shown in Figure 2.24. In the final screen, enter the virtual machine instance type characteristics and Click Finish.

Rational Team Concert

In addition to helping with source code management, RTC is a full-featured development collaboration tool, supporting Agile project management, a build server, and a defect management–tracking system. An RTC image is available in the IBM SmartCloud Enterprise catalog. RTC supports browser, Eclipse, and Microsoft Visual Studio clients. RTC allows for different roles, such as project administrator, project lead, and project team member, to support multiple projects and teams with delegated administration. Rational Team Concert images are available on the IBM SmartCloud Enterprise.



Cloud Environment Settings

Specify the settings of the cloud environment that contains the server

(●) New cloud connection

Cloud server: IBM Smart Business Development and Test

User name: a@b.c

Password: ●●●●●●

Connection name: secret

[✓] Save cloud connection

( ) Use existing cloud connection

Cloud connection names:

[ ] Perform an extensive search for available image types and instances

Select this option to perform an extensive search for all supported cloud images and instances including user-created images. Note that there may be a delay during sign-in when this option is

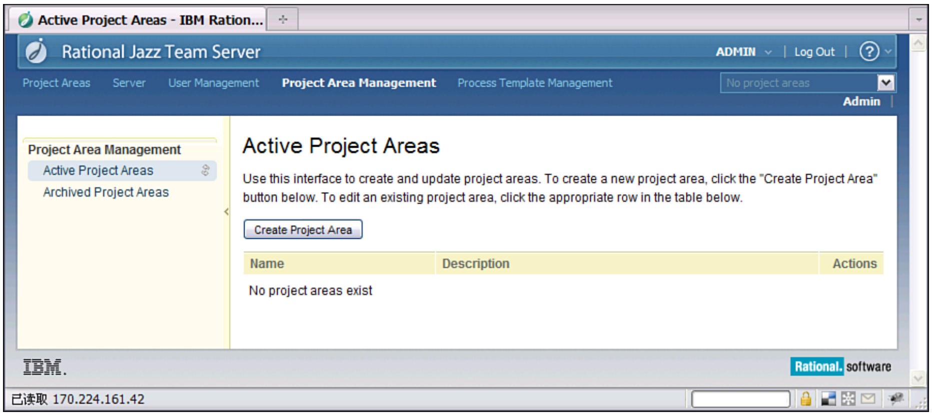
Figure 2.24 Enter cloud credentials in the New Server Wizard

To try Rational Team Concert, search for it in the catalog and then click the Start an Instance of Your Image link. To access the web user interface, use the URL https://hostname/jazz/web/. You will be prompted for the administrative password that you entered when provisioning the virtual machine instance. After you have logged in, you will see a message to perform the Jazz™ Team Server setup. Follow the Fast Path. Two embedded licenses are available for developers and can be assigned during setup process. Select the Tomcat User Database for the user registry.

Navigate to the Project Management area. Initially, there are no projects. Follow the link to the Project Area Management section. You should see something like Figure 2.25.

Click the Create Project Area button. Enter a name, summary, and description for the project. Click the Deploy Predefined Process Templates link.

You can add users to RTC using the User Management section, as shown in Figure 2.26. After you have created users, you can add them to the project.

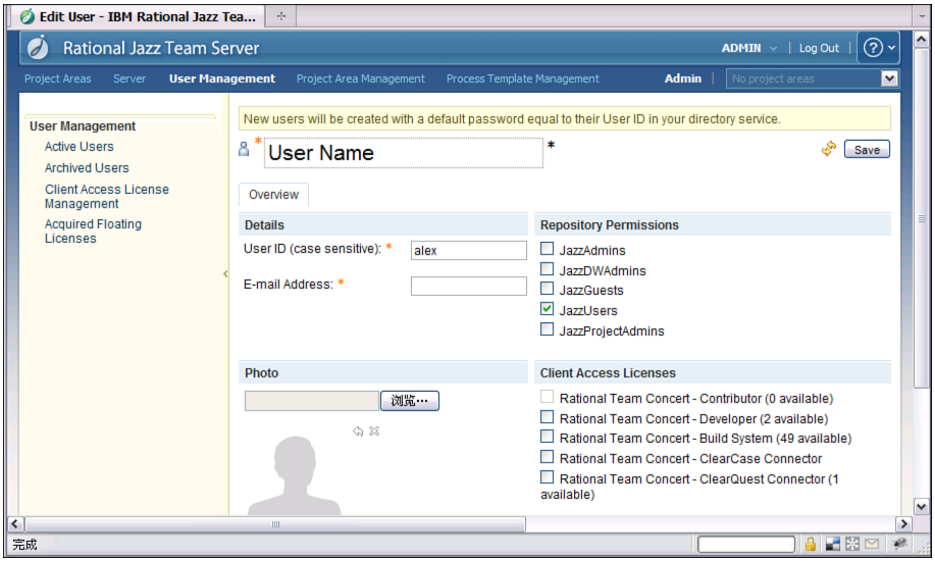


Active Project Areas

Use this interface to create and update project areas. To create a new project area, click the "Create Project Area" button below. To edit an existing project area, click the appropriate row in the table below.

Name	Description	Actions
No project areas exist		

Figure 2.25 Rational Team Concert Project Areas Management section



New users will be created with a default password equal to their User ID in your directory service.

User Name *

Details

User ID (case sensitive): * alex

E-mail Address: *

Repository Permissions

- [ ] JazzAdmins
- [ ] JazzDWAdmins
- [ ] JazzGuests
- [x] JazzUsers
- [ ] JazzProjectAdmins

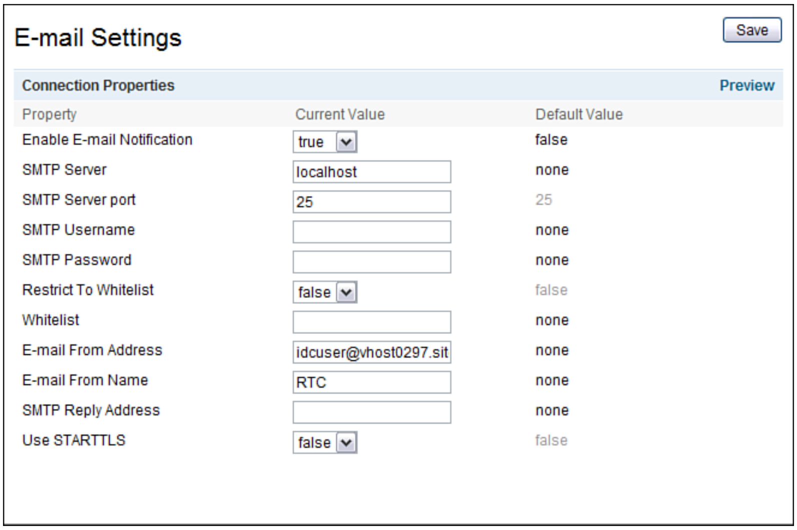
Photo

Client Access Licenses

- [ ] Rational Team Concert - Contributor (0 available)
- [ ] Rational Team Concert - Developer (2 available)
- [ ] Rational Team Concert - Build System (49 available)
- [ ] Rational Team Concert - ClearCase Connector
- [ ] Rational Team Concert - ClearQuest Connector (1 available)

Figure 2.26 User Management section in RTC

You can configure email settings in RTC in the Server area under the Email Settings menu. First, set up the email server, as explained in the “Email” section in Chapter 6. Then navigate to Server, Configuration, Email Settings and enter values for the email server, as shown in Figure 2.27.


E-mail Settings

Connection Properties

Property	Current Value	Default Value
Enable E-mail Notification	true	false
SMTP Server	localhost	none
SMTP Server port	25	25
SMTP Username		none
SMTP Password		none
Restrict To Whitelist	false	false
Whitelist		none
E-mail From Address	idcuser@vhost0297.sit	none
E-mail From Name	RTC	none
SMTP Reply Address		none
Use STARTTLS	false	false

Figure 2.27 RTC Email Settings configuration

Set Enable Email Notification to True. Enter localhost for the SMTP server. Use the fully qualified domain name of the server as the last part of the email address—for example, `idcuser@vhost0297.site1.compute.ihost.com`. Click Save.

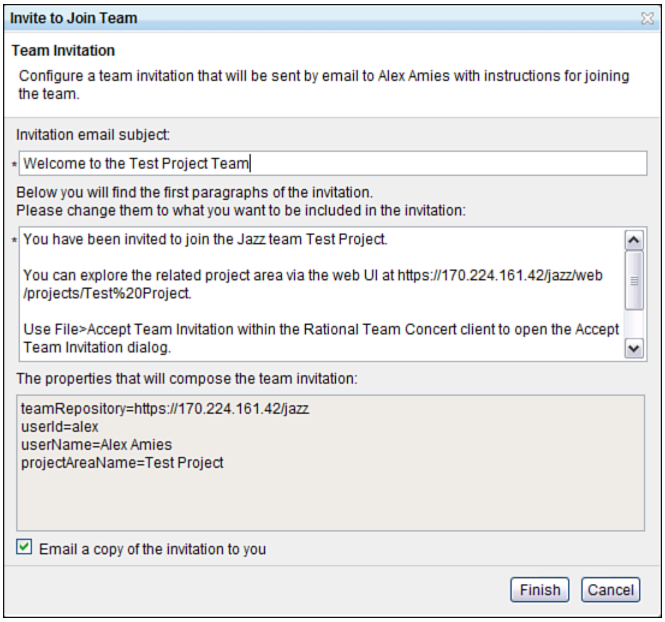
You can add users to projects using the Project Management area. To add a user, navigate to Project Management, Project Name and, under the Members header, add the user to the project. To generate an invitation, hover over the line containing the member and, at the right of the line, click the Invite to Join Team Icon (which is visible only when you hover over it). You should see something like Figure 2.28.

The web user interface is useful for many purposes, but you should use the RTC Eclipse client to administer RTC or as a developer. You can download the client from jazz.net and use the Install Manager to install it.

Build and Deployment Automation

Rational Application Developer and integrated development tools are ideal for code development, including performing local builds and unit testing. However, creating a repeatable build process is critical for application development in a team greater than one person. Many tasks are often included in build and deployment lifecycles, including these:

- Compilation
- Packaging
- Validation
- Unit testing
- Middleware configuration
- Deployment



Configure a team invitation that will be sent by email to Alex Amies with instructions for joining the team.

Invitation email subject:

Welcome to the Test Project Team

Below you will find the first paragraphs of the invitation.
Please change them to what you want to be included in the invitation:

You have been invited to join the Jazz team Test Project.

You can explore the related project area via the web UI at https://170.224.161.42/jazz/web/projects/Test%20Project.

Use File>Accept Team Invitation within the Rational Team Concert client to open the Accept Team Invitation dialog.

The properties that will compose the team invitation:

```
teamRepository=https://170.224.161.42/jazz
userId=alex
userName=Alex Amies
projectAreaName=Test Project
```

☑ Email a copy of the invitation to you

Figure 2.28 RTC team member invitation

In cloud computing, you can take deployment automation a step further by creating virtual machines with preinstalled software.

Both open source and commercial build tools are capable of handling build and deployment automation, including Ant, Maven, and IBM Build Forge®. We focus on the Apache Maven open source tool. Importantly, it has definitions for all the phases of build and deployment that we are interested in and includes an extensible framework. In addition, it has patterns for best practices and plug-ins for useful build tasks.

To set up Maven, download it from the project web site and define the environment variables shown here:

```
set M2_HOME=D:\opt\apache-maven-3.0.3
set JAVA_HOME=D:\Program Files\IBM\SDP\jdk
set PATH=%JAVA_HOME%;%PATH%;%M2_HOME%\bin
```

(Or do the equivalent for Linux.) Verify that Maven is available by executing this command:

```
>mvn --version
Apache Maven 3.0.3 (r1075438; 2011-03-01 01:31:09+0800)
Maven home: D:\opt\apache-maven-3.0.3
Java version: 1.6.0, vendor: IBM Corporation
Java home: D:\Program Files\IBM\SDP\jdk\jre
```

```
Default locale: en_US, platform encoding: GB18030
OS name: "windows xp", version: "5.1 build 2600 service pack 3", arch:
"x86", family: "windows"
```

You should see something like the previous information printed. You can create a Maven repository and project structure with this command:

```
>mvn archetype:generate -DgroupId=com.iotdata -DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

This downloads various plug-ins the first time it executes. Then it generates an application project directory called `my-app` with a source directory tree and a `pom.xml` file, which are part of the Maven standard project structure. The Project Object Model (`pom.xml`) file is the build script and is shown here:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.iotdata</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

This defines the project, including a group ID, the artifact ID `my-app`, JAR packaging, and a dependency on `junit`. The group ID is a unique identifier for the organization. In this case, the package is a JAR, but it can also be a WAR, EAR, or some other type of output artifact. You can substitute your own company web site for the `url` parameter.

To build the application change to the `my-app` directory and execute this command:

```
> mvn package
[INFO] BUILD SUCCESSFUL
...
```

The `package` argument is a phase on the build lifecycle. This command downloads more plug-ins the first time you execute it, but it should end with the success message shown previously. Maven executes all the phases in the build lifecycle, up to `package`. In this case, it includes the compile phase. It generates the JAR file `my-app-1.0-SNAPSHOT.jar` in the target directory. Other common phases in the Maven build lifecycle include these:

- Validate
- Compile
- Test
- Package
- Integration test
- Verify
- Install
- Deploy
- Clean

You also can define your own, such as gathering code quality metrics. To try the package, type the following command:

```
> java -cp target/my-app-1.0-SNAPSHOT.jar com.iotdata.app.App
Hello World!
```

In the generate archetype, Maven also created the source code for class `com.iotdata.app.App` for you, with a "Hello World" application located in the directory `${basedir}/src/main/java`. The compiled classes go in the directory `${basedir}/target/classes`. You can also run unit tests from Maven using this command:

```
>mvn test
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.iotdata.app.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015
sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...
```

Here, the output came from the unit test class that Maven also generated with the `archetype:generate` command.

Maven also has the concept of a repository. The local repository is the directory `~/.m2/repository`. On Windows, this is in a location such as `C:\Documents and Settings\Administrator\.m2\repository`. You can install your application into the local repository with this command:

```
>mvn install
```

This excludes test files that match certain naming conventions.

Now you are confident that you can use Maven, so let's apply it to the projects. In Maven, an archetype is a template for a project, which includes a standard project structure. The previous example is good, but in the examples in previous sections, we already had our own project created in our IDE and we were creating web and enterprise applications. Let's see how to address that and learn a little more about Maven in the process. The Maven standard directory structure is shown here:

```
my-app
|-- pom.xml
 -- src
    |-- main
    |   |-- java
    |    -- resources
    |        -- META-INF
    |            -- application.properties
    `-- test
         -- java
```

This includes an `application.properties` file for configuration settings specific to your project. You saw a dependency on JUnit in the previous example. In Maven, a dependency includes at least a groupId, an artifactId, a version, and a scope. To process dependencies, Maven looks in your local repository. If the dependency does not exist in your local repository, it downloads it from an external repository into your local repository. The default external repository is at http://repo1.maven.org/maven2/. You can set up your own remote repository as well, and the cloud is a good place to do that. Suppose that you have a dependency on a third-party JAR file. You can add that to `pom.xml` with a stanza such as this:

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.12</version>
  <scope>compile</scope>
</dependency>
```

In this example, it is for the log4j library.

We can add our JAR to our own repository with this stanza:

```
<distributionManagement>
  <repository>
    <id>iotdata-repository</id>
    <name>IoT Data Repository</name>
    <url>scp://host/repository/maven2</url>
  </repository>
</distributionManagement>
```

Here, `IoT Data` is the name of the fictitious company, and `host` is the name of the server. We also need to create a settings file in the `~/.m2` directory with the information shown here to connect to the repository:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <servers>
    <server>
      <id>iotdata-repository</id>
      <username>a.user</username>
      <!-- Default value is ~/.ssh/id_dsa -->
      <privateKey>/path/to/identity</privateKey>
      <passphrase>my_key_passphrase</passphrase>
    </server>
  </servers>
  ...
</settings>
```

Using these settings, Maven uses the SCP utility to upload the artifact to the repository. Sometimes third-party libraries are not in the Maven repository. For legal reasons, the J2EE and related libraries are not. In such a case, you need to go to the relevant web site, agree to the terms and conditions, and download the JAR files yourself. Then use this command:

```
> mvn install:install-file -Dfile=/opt/apache-tomcat70/lib/servlet-
api.jar -DgroupId=javax -DartifactId=servlet-api -Dversion=3.0 -
Dpackaging=jar
```

This example focuses on the Servlet APIs, which are needed to compile a J2EE web project.

To create a web application with the standard project structure, type this command:

```
> mvn archetype:generate -
DarchetypeGroupId=org.apache.maven.archetypes -
DarchetypeArtifactId=maven-archetype-webapp -DgroupId=com.iotdata
-DartifactId=my-webapp
```

Maven creates the project tree in the directory `my-webapp`, including the `pom.xml` file. But instead, let's use the project that we already created in the section "WebSphere Application Server." Add the following `pom.xml` file to the top directory of your web application project:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.iotdata</groupId>
  <artifactId>CloudAPIClientWeb</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>IoT Data Cloud API Client Web</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>servlet-api</artifactId>
      <version>3.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>${pom.artifactId}</finalName>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
          <warSourceDirectory>WebContent</warSourceDirectory>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Notice that the package is now a WAR. The `pom.xml` file uses the Maven WAR plug-in, which creates the WAR package. Our directory structure is a little different than the standard structure, so we add the `sourceDirectory` tag, which is a property of the `build` element. Notice that we used the dependency for the Servlet API that we discussed earlier. The scope is set to `provided` so that we do not have to copy the Servlet API jar to our web application.

To build it, change `directories` to `my-webapp` and type this command:

```
> mvn package
```

You should find the output in the file `CloudAPIClientWeb.war` in the target directory. Deploy this to your application server, point your browser at `http://host:8080/my-webapp/`, and you should see a page with the text "Hello World!"

You can manage multiple projects within a single Maven file structure by using multiple `pom.xml` files. You can do this because your web application will be included in an enterprise application. Create a `pom.xml` file like this above the_`my-app` and `my-webapp` directories:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.iotdata</groupId>
  <version>1.0</version>
  <artifactId>app</artifactId>
  <packaging>pom</packaging>
  <modules>
    <module>CloudAPIClientWeb</module>
    <module>CloudAPIClientEAR</module>
  </modules>
</project>
```

This defines modules for each of the_`CloudAPIClientWeb` and `CloudAPIClientEAR` projects. Create a `pom.xml` file for the EAR project, as shown here:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.iotdata</groupId>
    <artifactId>app</artifactId>
    <version>1.0</version>
  </parent>
  <groupId>com.iotdata</groupId>
  <artifactId>CloudAPIClientEAR</artifactId>
  <packaging>ear</packaging>
  <version>1.0</version>
  <name>IoT Data Cloud API Client EAR</name>
  <url>http://maven.apache.org</url>
```

```
    <dependencies>
      <dependency>
        <groupId>com.iotdata</groupId>
        <artifactId>CloudAPIClientWeb</artifactId>
        <version>1.0</version>
        <type>war</type>
      </dependency>
    </dependencies>
    <build>
      <finalName>${pom.artifactId}</finalName>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-ear-plugin</artifactId>
          <version>2.4</version>
          <configuration>
            <earSourceDirectory>
              ${basedir}
            </earSourceDirectory>
            <modules>
              <webModule>
                <groupId>com.iotdata</groupId>
                <artifactId>CloudAPIClientWeb</artifactId>
                <bundleFileName>
                  CloudAPIClientWeb.war
                </bundleFileName>
              </webModule>
            </modules>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </project>
```

This defines a parent that refers to the top-level `pom.xml`. You also need to add this stanza to your web `pom.xml`. This time, the package value is `ear`, and it uses the Maven `ear` plug-in. It defines the web model from the earlier web project.

Now clean and rebuild the entire project with this command:

```
> mvn clean install
```

The `ear` will be built, and it will include the WAR file from the web application. Now you know how to create a reproducible build. That is important because, in test and production environments, you will not allow developers to connect to the application server with their IDEs.

Business Scenario: Application Lifecycle Management Tools

IoT Data is a small-to-medium project with five developers, three testers, an architect, a project manager, and a market manager. It decides to follow an Agile development project, to reduce risk by always having a buildable and working code base and by adding capabilities in increments with each iteration. The company adopts IBM Rational Software Architect (RSA) as its standard development environment. This helps in the development of the J2EE application and includes design capabilities.

The source code will be stored in the IBM Rational Team Concert (RTC) code repository. The RTC server will run in the cloud. RTC includes a client plug-in that installs into RSA, which makes checking in code possible from a single tool.

RTC also acts as the project management tool, allowing the project manager to track the project progress. Defects are also tracked in RTC, but the test team accesses RTC via the web interface. The architect enters functional requirements as Agile stories in RTC, and development work is tracked against these requirements. However, the project manager uses IBM Rational Requirements Composer to track requirements from business stakeholders. He gets many more business requirements than the development team can deliver but they can become critical sometimes when customer deals depend on delivery of specific requirements. Executive commitments to customers need to be tracked especially carefully.

CHAPTER 3

Developing with IBM SmartCloud Enterprise APIs

IBM SmartCloud Enterprise is a public Infrastructure as a Service cloud hosted at www.ibm.com/cloud/enterprise and is suitable for enterprises. IBM's differentiators include a broad range of services and products, self-service and delegated administration models that enable collaboration, enterprise-suitable business support services, and a large catalog of virtual machine images created by the image development center of excellence.

The IBM SmartCloud Enterprise REST API is the fundamental way to programmatically communicate with the IBM SmartCloud Enterprise. It includes most of the capabilities you can perform from the self-service user interface, including managing instances, images, storage, and IP addresses. For convenience, a command-line API and Java API client are also provided; both wrap the REST services. The Java API client and command-line tool require downloads, which are freely available to registered users in the asset catalog. The command line conveniently speeds up operations for power users, but if you need to create an application that interprets the output from the API, you need to write a program. You can use the IBM SmartCloud Enterprise API to add capability or integrate cloud management into your other applications. For Java programmers, the most convenient option is the Java API; developers in other languages can use the REST API. This chapter explains the principles and then gives examples of use of the command line, Java API, and REST interfaces.

Resource Model

Figure 3.1 shows the resource model.

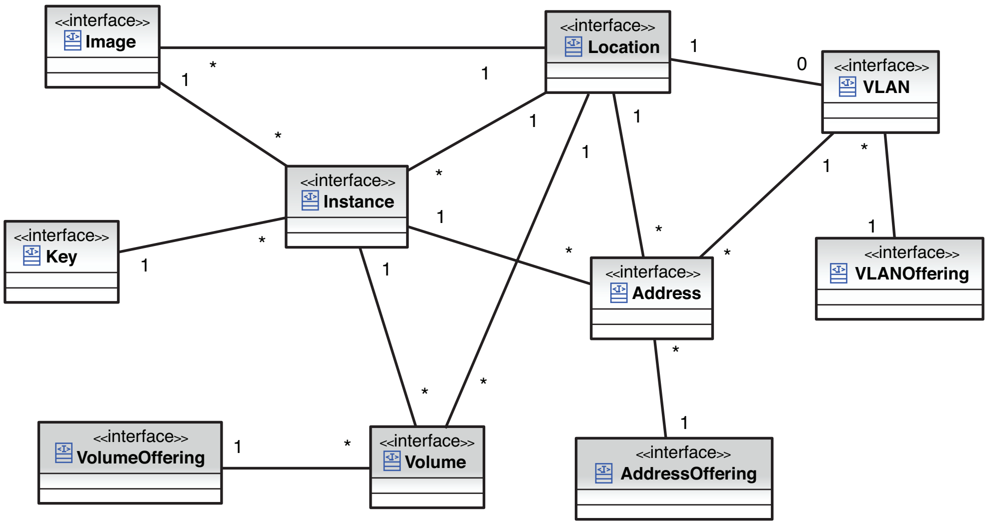


Figure 3.1 Resource model for IBM SmartCloud Enterprise

The fundamental resource of interest is an **Instance** object, which represents a virtual machine. An **Image** object represents a snapshot by capturing or saving a virtual machine, with some additional data, from which other Instance objects can be created. Because there can be many Instances of an Image, this is a one-to-many relationship. An Address is a reserved IP address. An Instance can be assigned zero or more Addresses, so this is also a one-to-many relationship. A Location represents a data center. All the resources, except Key, are associated with a Location. A Key is used to log onto an Instance using SSH. One Key can be used for many Instances.

Each resource has a set of attributes. Some attributes can be set during creation of the virtual machine instance and then changed at will afterward. Some attributes can be set only at creation time and are read-only afterward. Some attributes are generated by the system and only readable by users. The naming convention that we give is the one from the IBM SmartCloud Enterprise native API.

The key entities include instance, image, address, volume, credential, and VLAN:

- **Instance**—An Instance is a virtual machine created from an Image. It is normally in the active state, meaning that it is running, but it can also be in other states, such as provisioning or failed.

- **Image**—An Image is a template that can be used to create an instance of a virtual machine. Think of this as a large file that saves the file systems of an operating system, all the software installed on it, and configuration settings. Some of the configuration settings for runtime use, such as IP address, are not saved.
- **Address**—An Address is an IP address that can be either assigned by the cloud to an instance at provisioning time or reserved by the user as an independent operation.
- **Volume**—A Volume is a virtual block storage device that can be attached to an instance. Only one instance can be attached to a volume at any given time, but multiple volumes can be attached to a single instance.
- **Credential**—A Credential is used to connect to a running virtual machine. On the IBM SmartCloud Enterprise, the credential is an SSH public–private key pair.
- **VLAN**—A VLAN is the virtual equivalent of a subnet.

Configuration and other utilities include these:

- **Location**—Global delivery points are important to reduce network latency and satisfy compliance needs for certain enterprises that store important information. Images, Instance, Address, and Volume resources are all located in a particular location, which this object represents.
- **VM Configuration**—Specifies the compute size of an Instance, including number of CPUs, amount of memory, and amount of local disk storage.
- **Volume Offering**—Specifies the characteristics of a storage volume.

Entity Lifecycles

The different states of the individual resources are listed in the appendixes of the IBM SmartCloud Enterprise REST API Reference. The states are given as numeric codes, for ease of consumption by the APIs. Instance entities have a total of 15 states. Table 3.1 lists the most important states.

Table 3.1 Key States for Instance Entities

Code	State	Description
0	New	The request to provision an instance has just been submitted.
1	Provisioning	The instance is being created.
2	Failed	The request did not succeed.
3	Removed	The instance has been deleted.
5	Active	The instance is running and available to be used.

Table 3.1 Key States for Instance Entities (continued)

Code	State	Description
7	Deprovisioning	The instance is in the process of being deleted.
8	Restarting	The instance is being rebooted.
9	Starting	The instance is booting.

If an instance is in normal operation, it is in the *active* state. When creating a new instance, the instance begins in the *new* state and transitions through to the *provisioning* state and then the *active* state, when it can be used. Figure 3.2 illustrates the instance lifecycle.

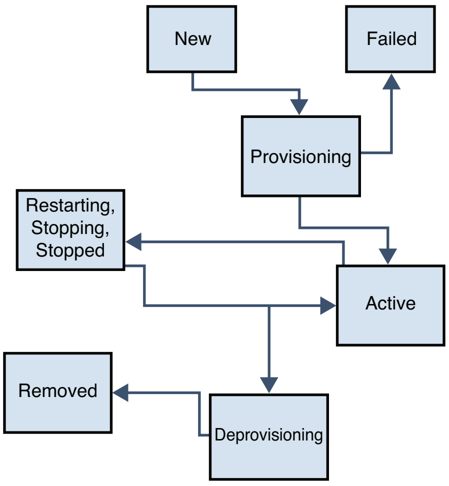


Figure 3.2 Instance partial lifecycle

Table 3.2 lists the states for images.

Table 3.2 States for Image Entities

Code	State	Description
0	New	The request to save an instance has just been submitted.
1	Available	An instance may be created from the image.
2	Unavailable	The image is not available for use.
3	Deleted	The image has been deleted.
4	Capturing	The image is currently being saved from an instance.

The normal state for images is *available*. An image cannot be deleted while an instance for that image exists. The other important interaction point with instances is saving, also known as capturing, an instance to an image. When a request to save image is first submitted, it is in the *new* state. Then it transitions to the *capturing* state and ends in the *new* state. Figure 3.3 shows the image lifecycle.

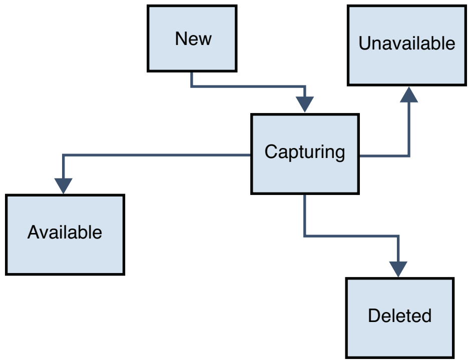


Figure 3.3 Image lifecycle

Table 3.3 shows the IP address states.

Table 3.3 States for IP Address Entities

Code	State	Description
0	New	The request for an IP address has just been submitted.
1	Allocating	The IP address is in the process of being allocated.
2	Free	The IP address is available to be attached to an instance.
3	Attached	The IP address is attached to an instance.
4	Releasing	The IP address is in the process of being released.
5	Released	The IP address has been released.
6	Failed	The request for an IP address failed.
7	Pending	

Dynamically assigned IP addresses do not have independent entity representations. IP addresses are not deleted like instances or images; they are simply returned to a pool for reuse. However, the entity representations of the IP addresses appear to be deleted. All IP addresses currently are IPv4. Figure 3.4 depicts the address lifecycle.

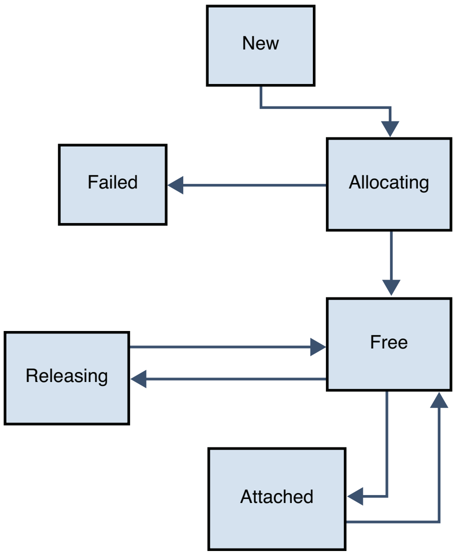


Figure 3.4 Address lifecycle

When an IP address is used by an instance, it determines the MAC address for the virtual network interface. The MAC address thus cannot be determined when reserving an IP address, but it can be determined after the instance is provisioned. If you reuse the same reserved IP with a different instance, you get the same MAC address. If multiple IP addresses exist, there will be multiple MAC addresses, one MAC address per each IP address associated with the instance.

Table 3.4 lists the key states for storage volumes.

Table 3.4 Key States for Storage Volumes

Code	State	Description
0	New	The request for storage volume has just been submitted.
1	Creating	The volume is in the process of being created.
2	Deleting	The volume is in the process of being deleted.
3	Deleted	The volume has been deleted.
4	Attached	The disk has not been attached (previously was mounted).
5	Detached	The disk has been attached (previously was mounted).
6	Failed	The request for a volume failed.
7	Delete pending	
8	Being cloned	The disk is being cloned.
9	Cloning	

The normal state for a volume that is not attached to an instance is *unattached*. A key difference exists between Linux and Windows with attached disks. With Linux, the normal process is to mount the disk at the time of provisioning an instance. With Windows, this is not possible; the disk is attached to Windows instances with a RAW format. Mounting is a concept that is not as transparent in Windows systems compared with Linux systems.

Command Line

Command-line tools can be helpful for power users or in scripting contexts to automate common tasks and configuration. The IBM SmartCloud Enterprise command line tool has nearly the same capabilities as the REST API. This section walks you through setting up the command line tool, querying the catalog, provisioning an instance, managing storage, working with parameters during the provisioning process, managing IP addresses, and saving virtual machine images.

Environment Setup

To start, this section demonstrates how to create an instance using the command line on a Windows client. Command-line utilities are also available for Linux/UNIX clients. Download the command-line tool from the Asset Catalog. You will see a link on the Support tab. Unzip the bundle somewhere on your local disk in a location with no spaces in the folder names. Follow these steps to get your scripting environment set up:

1. Install a version 1.6 JDK if you do not have one already installed.
2. Define `JAVA_HOME` as the location where you installed the JDK.
3. Extract the command-line tool ZIP into a directory on your system without spaces in the path. Define a variable called `DT_HOME` for this.
4. Create a directory for your scripts and define a variable called `SCRIPT_HOME` for the location.

After you have created a directory for your scripts, initialize your client with a password file, which is required and protects your real IBM SmartCloud Enterprise password. These commands are shown here:

```
> set JAVA_HOME=D:\Program Files\IBM\SDP\jdk
> set PATH=%JAVA_HOME%\bin;%PATH%
> set DT_HOME=D:\alex\architecture\articles\cloud_monitoring\
cmd
> set SCRIPT_HOME=D:\alex\architecture\articles\cloud_monitoring\
script
> cd %DT_HOME%
> ic-create-password -u a@bc.com -p aq1sw2 -w unlock -g
%SCRIPT_HOME%/mykey.ext
> cd %SCRIPT_HOME%
```

In the script, substitute your own values for `user ID (a@bc.com)`, `password (aq1sw2)`, and `passphrase (unlock)`.

Querying the Catalog

You first need to retrieve some information from the catalog to determine what image to provision an instance for. To get a list of images, use this command:

```
> ic-describe-images -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext
> images_list.txt
```

Save the output to a file called `images_list.txt` because the output is long. You will see output similar to this:

```
...
ID : 20001119
Name : Red Hat Enterprise Linux 5.4 (64-bit)
Visibility : PUBLIC
State : AVAILABLE
Owner : SYSTEM
Platform : Red Hat Enterprise Linux/5.4
Location : 41
    ~~~~~
    InstanceType ID : BRZ64.2/4096/850
    Label : Bronze 64 bit
...
    InstanceType ID : COP64.2/4096/60
    Label : Copper 64 bit
Price : USD0.4/UHR
...
```

This describes a Red Hat Enterprise Linux 5.1 server with image ID 20001119. The data center ID is 41 (Research Triangle Park, North Carolina). The images can be provisioned with different compute sizes. The size shown is Bronze 64 bit, which has 4086 MB memory and 850 GB storage. The next instance that it can be provisioned on is Copper 64 bit, and so on.

Provisioning an Instance

Instances can be provisioned at different data centers. The data center also must match for the instance, image, IP address, and any storage volumes attached. To find a list of available locations, use the `ic-describe-locations` command:

```
> ic-describe-locations -u a@bc.com -w unlock -g
%SCRIPT_HOME%/mykey.ext
```

This gives output similar to this:

```
...
ID : 101
Location : ca-on-dc1
Name : ca-on-dc1
Description : Data Center 1 at Ontario, Canada
----------------------------------
----------------------------------
ID : 121
Location : ap-jp-dc1
Name : ap-jp-dc1
Description : Data Center 1 at Makuhari, Japan
----------------------------------
```

The first entry is data center 101, in Ontario, Canada; the second is data center 121, in Makuhari, Japan. At the time of writing, data centers are located in Research Triangle Park (RTP), North Carolina (41); Enhigen, Germany (61); Boulder, Colorado (81 and 82); Ontario, Canada (101); and Makuhari, Japan (121). Use the location IDs in your program to identify specific data centers.

You need to define an SSH key pair to log into the instance. You can do this using the user interface or the command line. You can get a list of your key pairs using this command:

```
> ic-describe-keypairs -u a@bc.com -w unlock -g
%SCRIPT_HOME%/mykey.ext
```

You should see output similar to this:

```
Name : MyKey
Default Key : false
Material : AAAAB3NzaC1yc2EAAAADAQABAAABAQCLI7Q/ntSeb7eeZpnA9J5qGsR/
CjnNlACav9O2R
ztowZwnWx9vMwzLue4z3A+cfBrzmGvZgnNTkVlnhKbjJ72dvIMLWgE5oKnYsud+As0+pxJ
cfJ1Pd7
xeOnxayX+dMM7lIjQHQsorFC9/AEKix8uIzTI4G0LjLTtLk56QXQw+PLNcjNx77eG1G1gi
nF2UXV9
Wjvhbqt8MebjDpoZP55URKlH+24IifcURKXVV6IfjfHRcTToy5sL1QrLqKXEMK08TKndMp
WvSEv3j+
q5X7DwwAiC4V7NH1OWIVl7VtDfoCoDsli/
KEMAlMFOYIxlOeLA7MlmAWOb99K2p
LastModifiedTime : 2010-11-24 04:07:29
0 instances related.
...
```

`Name` is the name of the key used to identify it in the IBM SmartCloud Enterprise portal and on the command line when requesting a new instance. `Material` is the public part of the key pair. When requesting the key, you should save the private key, which is not displayed in the command, on your local disk. The IBM SmartCloud Enterprise stores only your public key. If you lose your private key, you cannot recover it from the cloud.

Now you have all the information you need to create the instance. You can do this with the following command:

```
> ic-create-instance -u u@abc.com -w unlock -g %SCRIPT_HOME%/mykey.ext
  -t COP64.2/4096/60 -n MyRedHat -k 20001119 -c MyKey -d
"Base OS"
  -L DATA_CENTER_ID
```

The `-t` (or `type`) parameter is compute size. The `-n` parameter is a user-friendly name for the image. The `-k` parameter is the image ID from the `describe images` command. The `-c` parameter is the name of the key pair from the `describe-keypairs` command. The `-d` (or `description`) parameter is a potentially long description of the image. The `-L` (or `location`) parameter is the data center ID. This should match the output from the image you selected from the `describe-images` command. No IP address is included; the IP address will be dynamically generated, and you can discover it after the virtual machine has been provisioned. You should see output similar to this:

```
Executing action: CreateInstance ...
The request has been submitted successfully.
1 instances!
----------------------------------
ID : 49571
Name : MyRedHat
Hostname :
InstanceType : COP64.2/4096/60
IP :
KeyName : MyKey
Owner : a@bc.com
RequestID : 49871
RequestName : MyRedHat
Status : NEW
Volume IDs :
----------------------------------
Executing CreateInstance finished
```

The request takes a little while to provision. To find the status, use this command:

```
> ic-describe-request -u a@bc.com -e 49871 -w unlock -g
%SCRIPT_HOME%/mykey.ext
```

The `-e` parameter is the request ID from the `create-instance` command. You should see output similar to this:

```
Executing action: DescribeRequest ...
1 instances :
----------------------------------
ID : 49571
Name : MyRedHat
```

```
Hostname : 170.224.160.62
InstanceType : COP64.2/4096/60
IP : 170.224.160.62
KeyName : MyKey
Owner : a@bc.com
RequestID : 49871
RequestName : MyRedHat
----------------------------------
Executing DescribeRequest finished
```

The output indicates that the request has finished. This means that you can use the instance. You can also find out about the instance using the `describe-instances` request, shown here:

```
ic-describe-instances -u aamies@cn.ibm.com -w unlock -g
%SCRIPT_HOME%/mykey.ext
```

No additional parameters are needed in this command. You should see output similar to this:

```
Executing action: DescribeInstances ...
1 instances!
----------------------------------
ID : 49571
Name : MyRedHat
Hostname : 170.224.160.62
InstanceType : COP64.2/4096/60
IP : 170.224.160.62
KeyName : MyKey
Owner : a@bc.com
RequestID : 49871
RequestName : MyRedHat
Status : ACTIVE
Location : 41
Volume IDs :
Disk Size : 60
Root Only : false
----------------------------------
Executing DescribeInstances finished
```

Now you can use your instance. This is the command line for deleting it when you are finished:

```
> ic-delete-instance -u u@abc.com -w unlock -g %MY_HOME%/mykey.ext -l
INSTANCE_ID
```

Provisioning Storage

When you are familiar with how a basic instance works, you can try to attach storage. You can obtain a description of storage volume offerings using this command:

```
ic-describe-volume-offerings -u aamies@cn.ibm.com -w unlock -g
%SCRIPT_HOME%/mykey.ext
```

This gives the following type of output:

```
Executing action: DescribeVolumeOfferings ...
----------------------------------
ID : 20001208
Name : Small
Location : 41
Price : .../UHR
CurrencyCode : USD
CountryCode :
PricePerQuantity : 1
UnitOfMeasure : UHR
Supported Format :
~~~~~~
Format : EXT3
Format : RAW
~~~~~~
...
```

The `ID` is used in commands to create storage volumes. `Name` is also the size. Use EXT3 format on Linux and RAW format on Windows. You can request a new storage volume with this command:

```
ic-create-volume -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext -n
MyVolume -s Small -f EXT3 -L 41 -O 20001208
```

The `-n` parameter is the name of the volume, the `-s` parameter is the size, the `-f` parameter is the format, the `-L` parameter is the location, and the `-O` parameter is the offering ID. You should see output similar to this:

```
Executing action: CreateVolume ...
The request has been submitted successfully.
ID : 7253
Name : MyVolume
Owner : a@b.com
Size : 256
Format : ext3
InstanceID : 0
OfferingID : 20001208
```

The ID of the volume is returned in the output (7253). You need this to attach the volume to an instance. Use the `ic-describe-volumes` command to get a list of volumes, and use `ic-delete-volume` to delete one. To use the volume with an instance, use the `create instance` command with some additional parameters:

```
ic-create-instance -u aamies@cn.ibm.com -w unlock -g
%SCRIPT_HOME%/mykey.ext -t COP64.2/4096/60 -n MyRedHat -k 20001119 -c
alex-prod -d "Base OS" -L 41 -v 7253 -m
{oss.storage.id.0.mnt:/mnt/volume1}
```

The `-v` option specifies the storage volume ID, and the `-m` parameter specifies the mount point of the volume. To check it out, log into the instance with putty using the idcuser ID and no password. Use the Linux `df` command:

```
[idcuser@vhost1374 ~]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda2             60880276   3563828  54804044   7% /
/dev/hda1               101086     11738     84129  13% /boot
tmpfs                  2022056         0   2022056   0% /dev/shm
/dev/hdb1            264218344    191712 250605132   1% /mnt/volume1
```

The `/mnt/volume1` device is the storage volume we provisioned and attached to the instance.

Provisioning an Instance with Parameters

Creating an instance with parameters is similar. First, find out the parameters that are associated with the image using the `describe image` command:

```
> ic-describe-image -u aamies@cn.ibm.com -w unlock -g
%SCRIPT_HOME%/mykey.ext -k 20007420 > %SCRIPT_HOME%/image.txt
```

The `-k` parameter is the image ID 20017833 for a WebSphere image that contains parameters. These parameters are parsed from the `parameters.xml` file and listed in the output:

```
ID : 20007420
Name : IBM WebSphere Application Server V7.0 - DUO
...
there are 4 additional parameters.
@@@@@@@@@@@@@@@@@@@@@@@@
name : WASAdminUser
type : text
description : Specify a user ID for executing and administering
WebSphere processes on the instance. To ensure security, do not
specify 'root' or 'idcuser' as administrator ID.
value : null
@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@
name : WASAdminPassword
```

```
type : password
description : Specify a password for WebSphere administrator ID.
Password must contain at least 1 number, at least 1 lower case letter,
and at least 1 upper case letter.
value : null
@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@
name : WASProfileType
type : radioGroup
description : Choose development profile if you are developing an
application using tools such as IBM Rational Application Developer.
Choose default single server server profile for running the
application in a production-like setting.
value : development
Options :~~~~~
option : development
option : production
Options :~~~~~
@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@
name : WASAugmentList
type : radioGroup
description : Specify feature packs to enable in the profile
value : all
Options :~~~~~
option : cea
option : sca
option : sca_sdo
option : xml
option : all
option : none
Options :~~~~~
```

This tells you that four parameters exist. The first two parameters, `WASAdminUser` and `WASAdminPassword`, have no default values, so you must supply values for them. The second two parameters, `WASProfileType` and `WASAugmentList`, do have default values. If you do not supply values, the defaults are used. You can request the instance with the parameters using a form of the `create instance` command:

```
> ic-create-instance -u aamies@cn.ibm.com -w unlock -g
%SCRIPT_HOME%/mykey.ext -t COP32.1/2048/60 -n WebSphere -k 20007420 -c
alex-prod -d "WebSphere" -L 41 -m
{WASAdminUser:webadmin,WASAdminPassword:Aq1sw2de}
```

Managing IP Addresses

To understand IP address offerings, first use the `ic-describe-address-offerings` command. It gives output similar to this:

```
Executing action: DescribeAddressOfferings ...
----------------------------------
ID : 20001223
Location : 41
Ip Type : PUBLIC
Price : /UHR
CurrencyCode : USD
CountryCode :
PricePerQuantity : 1
UnitOfMeasure : UHR
...
```

This shows a public IP address offering in a location with ID 41 in addition to price data. You will need the address offering ID to provision. To allocate an IP address, use this command:

```
ic-allocate-address -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext
-L 41 -O 20001223
```

The `-O` parameter is the offering ID from the `describe address offerings` command. Use the `ic-describe-addresses` command to list IP addresses, as shown in this command:

```
ic-describe-addresses -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext
```

No additional special parameters are used here. This results in output similar to this:

```
Executing action: DescribeAddresses ...
3 addresses.
----------------------------------
ID : 36550
InstanceId : null
IP : 170.224.161.34
State : FREE
Location : 41
...
```

The address ID is needed if you want to assign this to a virtual machine. The instance ID field tells you which instance, if any, is associated with the IP address. If the state is `FREE`, you can use the IP address in an instance provisioning request. Use the `ic-release-address` command to release an address.

Saving Images

After you have worked with an image for some time, installed software, changed configuration settings, and so on, you will need to save the image. To do this, use the `ic-save-instance` command:

```
ic-save-instance -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext -l
50464 -n MyRedHatImage -d "Snapshot of my Red Hat Image"
```

Here, the `-l` parameter is the instance ID, the `-n` parameter is the name, and the `-d` parameter is a description. You should see output similar to this:

```
Executing action: SaveInstance ...
ID : 20016436
Name : MyRedHatImage
Visibility : PRIVATE
State : NEW
Owner : null
State : NEW
Description : Snapshot of my Red Hat Image
CreatedTime : Wed Feb 23 17:33:17 CST 2011
...
```

At this point, the request has been submitted, but the image has not yet been made. The ID of this image is 20016436. The visibility is private, which means that only you can see it. To find the status of the image, use the command `ic-describe-image`:

```
ic-describe-image -u a@bc.com -w unlock -g %SCRIPT_HOME%/mykey.ext -k
20016436
```

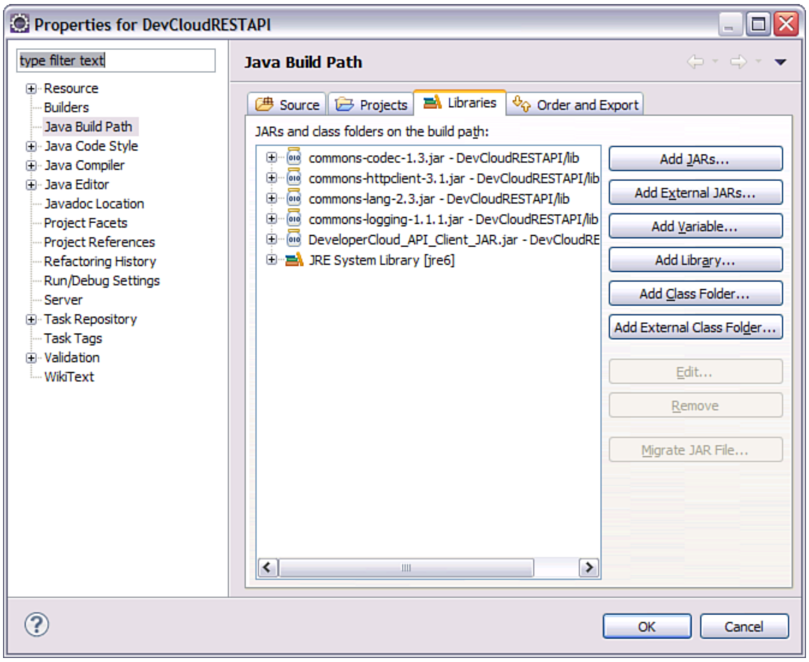
Here, the `-k` parameter is the image ID. You will also receive an email when the image has been saved. Use the `ic-delete-image` to delete the image.

Java API

In general, you can do a greater range with a Java program than with a command line. The examples in this section are not the same as those from the earlier command-line examples. You might want to read the previous section if you have not done so already.

Environment Setup

The IBM SmartCloud Enterprise provides a Java API to make it easy to manage resources via Java. A few prerequisite libraries can be downloaded from the Asset Catalog. These include the Apache Commons Codec 1.3, HttpClient 3.1, lang 2.3, and Logging 1.1.1 libraries, as well as the Developer Cloud API Client JAR. Figure 3.5 shows an Eclipse project setup.



JARs and class folders on the build path:

commons-codec-1.3.jar - DevCloudRESTAPI/lib
commons-httpclient-3.1.jar - DevCloudRESTAPI/lib
commons-lang-2.3.jar - DevCloudRESTAPI/lib
commons-logging-1.1.1.jar - DevCloudRESTAPI/lib
DeveloperCloud_API_Client_JAR.jar - DevCloudRE
JRE System Library [jre6]

Figure 3.5 Java library dependency setup

Querying the Catalog

The basic interface that implements most of the methods needed is `DeveloperCloudClient`. All methods on this interface, except `getLocale` and `setLocale setRemoteCredentials`, result in REST calls to the cloud. This basic program retrieves a list of images and prints the ID and name to the console:

```
import java.io.IOException;
import java.util.HashMap;
import java.util.List;

import com.ibm.cloud.api.rest.client.DeveloperCloud;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.*;
import com.ibm.cloud.api.rest.client.exception.*;

/**
 * Example program to demonstrate the Java client API's.
 */
```

```
public class JavaAPIExamples {

      public static final String USERNAME = "a@bc.com";
      public static final String PASSWORD = "secret";

      private DeveloperCloudClient client =
DeveloperCloud.getClient();

      /**
       * Configures the REST client proxy.
       */
      public JavaAPIExamples() {
            client.setRemoteCredentials(USERNAME, PASSWORD);
      }

      /**
       * Exercises the describeImages() method.
       * @throws UnauthorizedUserException Invalid user credentials
       * @throws UnknownErrorException Some other kind of error
       * @throws IOException Network problems
       */
      public void describeImages() throws UnauthorizedUserException,
UnknownErrorException, IOException {
            List<Image> images = client.describeImages();
            System.out.println("Found " + images.size() +
" image(s).");
            for (Image image: images) {
                  System.out.println("Found image with id " +
image.getID() +                         ", " + image.getName());
            }
      }

      /**
       * Entry point
       * @param argv Not used
       * @throws Exception Authorization, network IO, or other
problems
       */
      public static void main(String[] argv) throws Exception {
            System.out.println("Running Java API examples");
            JavaAPIExamples examples = new JavaAPIExamples();
            examples.describeImages();
      }
}
```

The program is executed from the command line. It first imports all the required packages and classes. It defines two constants for user ID and password; replace these with your own credentials. After this, it defines the client proxy using the `DeveloperCloud` class to create it. In the constructor, the program sets the credentials in the client proxy. In the method `describeImages`, the program calls the API `DeveloperCloudClient.describeImages`, which returns a list of `Image` objects. The program then iterates over the list, printing the ID and name. The output looks similar to this:

```
Running Java API examples
Found 336 image(s).
Found image with id 20009984, IBM DB2 Enterprise Developer Ed SLES
9.7.1 - BYOL
...
Found image with id 20016550, MyRedHatImage
Found image with id 20013501, alex SUSE public 12/1/10 6:33 PM
...
```

The previous program lists all the images in the catalog, in addition to the images owned by the user. To find a list of the images the user owns, use a method like this:

```
public void describeMyImages() throws UnauthorizedUserException,
UnknownErrorException, IOException {
      List<Image> images = client.describeImages();
      for (Image image: images) {
            if (USERNAME.equals(image.getOwner())) {
                  System.out.println("Found image with id " +
                  image.getID() + ", attributes: ");
                  System.out.println("Name: " + image.getName());
                  System.out.println("Description: " +
                   image.getDescription());
                  System.out.println("Visibility: " +
image.getVisibility());
                  System.out.println("Location: " +
image.getLocation());
                  System.out.println("State: " + image.getState());
                  System.out.println("Platform: " +
image.getPlatform());
                  System.out.println("Architecture: " +
                   image.getArchitecture());
                  System.out.println("Supported instance types: ");
                  List<InstanceType> instanceTypes =
                   image.getSupportedInstanceTypes();
                  for (InstanceType instanceType: instanceTypes) {
                        System.out.print(instanceType.getId() + " " +
                                    instanceType.getLabel() + ", ");
                        }
                  }
            }
      }
```

The program filters the list by checking the `owner` attribute of the image. A number of attributes and the list of supported instance types (compute sizes) are printed to standard output. The program output is similar to this:

```
Running Java API examples
Found image with id 20016550, attributes:
Name: MyRedHatImage
Description: Snapshot of my Red Hat Image
Visibility: PRIVATE
Location: 41
State: AVAILABLE
Platform: Red Hat Enterprise Linux/5.4
Architecture: x86_64
Supported instance types:
COP64.2/4096/60 Copper 64 bit, BRZ64.2/4096/60*500*350 Bronze 64 bit,
GLD64.8/16384/60*500*500 Gold 64 bit,
PLT64.16/16384/60*500*500*500*500 Platinum 64 bit,
SLV64.4/8192/60*500*500 Silver 64 bit,
...
```

Working with Virtual Machine Instances

You can use the previous information to provision a virtual machine instance. The attributes needed are the location ID, image ID, instance type, and a key when provisioning an instance. You can get a list of your keys using the `DeveloperCloudClient.describeKeys` method, as demonstrated in the following method:

```
public void describeKeys() throws UnauthorizedUserException,
UnknownErrorException, IOException {
      List<Key> keys = client.describeKeys();
      for (Key key: keys) {
            System.out.println("Found key with id " + key.getName());
      }
}
```

This results in the following output:

```
Running Java API examples
Found key with id mykey
...
```

The following code demonstrates creating an instance with the `DeveloperCloudClient.createInstance` method:

```
public void createInstance() throws UnauthorizedUserException,
UnknownErrorException, IOException,
      InsufficientResourcesException, InvalidConfigurationException,
PaymentRequiredException {
      System.out.println("Provisioning a new instance");
      Address address = null;
      Volume volume = null;
      Address[] secondaryAddresses = new Address[0];
      List<Instance> instances = client.createInstance(
            "Java API Test Instance",       // Name of instance
            "41",                           // Data center ID
            "20016550",                     // Image ID
            "COP64.2/4096/60",              // Instance type
            "mykey",                        // Key
            address,                        // Address
            volume,                         // Volume
            new HashMap<String,Object>(),   // Options
            null,                           // VLAN ID
            secondaryAddresses,             // Secondary IP addresses
            true);                          // Minimum ephemeral
      Instance instance = instances.get(0);
      System.out.println("ID: " + instance.getID());
}
```

The name, data center ID, image ID, instance type, and key were described previously. This method has a lot of strings, so we defined typed variables for address and volume for clarity, even though they are given null values. Also, for illustration, we included a HashMap for the options, even though we have not included any options. If the VLAN ID is set to `null`, a public Internet address is used. Otherwise, include the ID of your enterprise's VLAN. The secondary address is an optional list of additional IP addresses. The minimum ephemeral flag is an option for performance optimization. Because you likely will not need all the ephemeral (local) storage right away, the instance is provisioned with the minimum amount of storage needed; later expansion is allowed as needed. The method produces output similar to this:

```
Running Java API examples
Provisioning a new instance
ID: 50767
```

Notice that the instance is created immediately and has an identifier that you can use. However, immediately after the provisioning request, the virtual machine instance is not in a state that can be used. You must wait until the instance is in an active state before logging on to it. Use any of the methods `DeveloperCloudClient.describeInstances`, `describeInstance`, or `describeRequest` to find the status of the request. The following method demonstrates the use of `DeveloperCloudClient.describeInstances`.

```
public void describeInstances() throws UnauthorizedUserException,
UnknownErrorException, IOException {
      List<Instance> instances = client.describeInstances();
      System.out.println("Found " + instances.size() + "
instance(s).");
      for (Instance instance: instances) {
            System.out.println("ID: " + instance.getID());
            System.out.println("Name: " + instance.getName());
            System.out.println("IP: " + instance.getIP());
            System.out.println("Status: " + instance.getStatus());
      }
}
```

The program produces output similar to this:

```
Found 1 instance(s).
ID: 50767
Name: Java API Test Instance
IP: 170.224.160.133
Status: ACTIVE
```

Some images have additional parameters that must be supplied when creating an instance. You can find these in the `parameters.xml` file, available in the `image.getManifest()` method call. For example, the WebSphere Application Server (WAS) has these parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="parameters.xsd">
...
   <field name="WASAdminUser" type="text" label="WebSphere
administrator ID"
        description="Specify a user ID ..."/>
   <field name="WASAdminPassword" type="password" label="WebSphere
administrator password"
        pattern="^\w*(?=\w*\d)(?=\w*[a-z])(?=\w*[A-Z])\w*$"
        patternErrorMessage="Invalid Password. Must ..."
        description="Specify a password for ..."/>
   <field name="WASProfileType" type="radioGroup" label="Select a ..."
       description="Choose ...">
       <values>
          <value>development</value>
       </values>
```

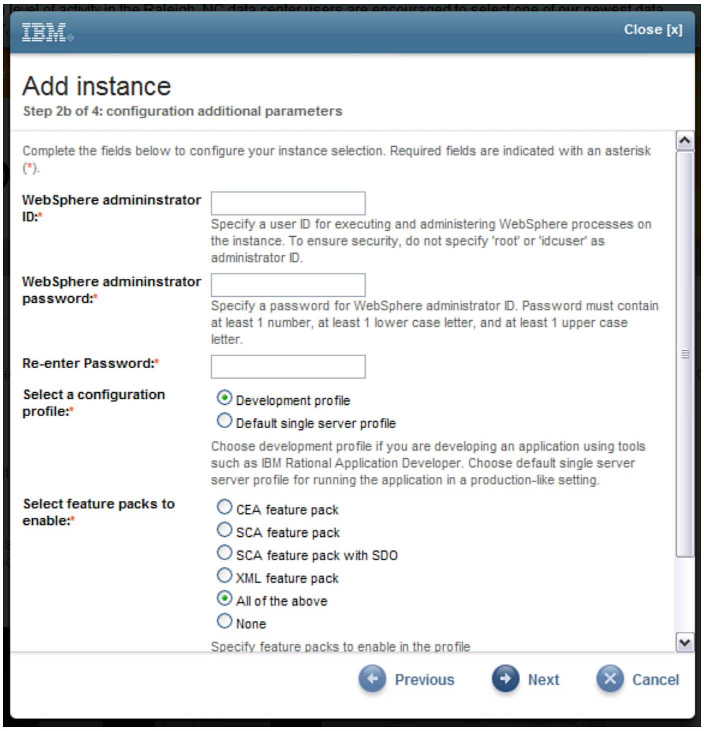
```
          <options>
             <option label="Development profile">development</option>
             <option label="Default single server
profile">production</option>
          </options>
      </field>
      <field name="WASAugmentList" type="radioGroup" label="Select ..."
          description="Specify feature packs to enable in the profile">
          <values>
             <value>all</value>
          </values>
          <options>
             <option label="CEA feature pack">cea</option>
             <option label="SCA feature pack">sca</option>
             <option label="SCA feature pack with SDO">sca_sdo</option>
             <option label="XML feature pack">xml</option>
             <option label="All of the above">all</option>
             <option label="None">none</option>
          </options>
      </field>
   </parameters>
```

The file indicates that two parameters are required: a user name for the WAS administrative user and a password. The other parameters, `WASProfileType` and `WASAugmentList`, include default parameters. If you do not supply a value, the defaults are used. On the IBM SmartCloud Enterprise user interface, you are prompted to enter these parameters with a panel in the Instance Creation Wizard, as shown in Figure 3.6.

These parameters can be supplied using the Java API, as in the following method:

```
public void createInstanceWebSphere() throws
UnauthorizedUserException,      UnknownErrorException, IOException,
InsufficientResourcesException,      InvalidConfigurationException,
PaymentRequiredException {
      Address address = null;
      Volume volume = null;
      Address[] secondaryAddresses = new Address[0];
      Map<String,Object> parameters = new HashMap<String,Object>();
      parameters.put("WASAdminUser", "***");
      parameters.put("WASAdminPassword", "***");
      List<Instance> instances = client.createInstance(
                  "WebSphere Server",            // Name of instance
                  "41",                          // Data center ID
                  "20017599",                    // Image ID
                  "BRZ32.1/2048/60*175",         // Instance type
                  "mykey",                       // Key
```

```
                        address,                            // Address
                        volume,                             // Volume
                        parameters,                         // Options
                        null,                               // VLAN ID
                        secondaryAddresses,                 // Secondary IP
        addresses
                        true);                              // Minimum ephemeral
            Instance instance = instances.get(0);
        }
```



Add instance

Step 2b of 4: configuration additional parameters

Complete the fields below to configure your instance selection. Required fields are indicated with an asterisk (*).

WebSphere admininstrator ID:*

Specify a user ID for executing and administering WebSphere processes on the instance. To ensure security, do not specify 'root' or 'idcuser' as administrator ID.

WebSphere admininstrator password:*

Specify a password for WebSphere administrator ID. Password must contain at least 1 number, at least 1 lower case letter, and at least 1 upper case letter.

Re-enter Password:*

Select a configuration profile:* [✓ Development profile] Default single server profile

Choose development profile if you are developing an application using tools such as IBM Rational Application Developer. Choose default single server server profile for running the application in a production-like setting.

Select feature packs to enable:* CEA feature pack / SCA feature pack / SCA feature pack with SDO / XML feature pack / [✓ All of the above] / None

Specify feature packs to enable in the profile

Figure 3.6 Instance creation parameters for WebSphere Application Server

Locations and Capabilities

Different cloud locations help you minimize network latency while delivering and consuming services at different points around world. In the context of the IBM SmartCloud Enterprise, a location is a data center. Some data centers have different capabilities. You can query the data centers using the `describeLocations` method, shown here, to determine the capabilities.

```
        public void describeLocations() throws UnauthorizedUserException,
        UnknownErrorException, IOException {
            List<Location> locations = client.describeLocations();
            System.out.println("Found " + locations.size() + " 
        locations(s).");
            for (Location l: locations) {
                Map<String, Capability> capabilities = 
        l.getCapabilities();
                Set<String> keys = capabilities.keySet();
                System.out.println("Location " + l.getId() + ", name " + 
        l.getName() + ", capabilities:");
                for (String key: keys) {
                    Capability capability = capabilities.get(key);
                    System.out.println("\t" + key + " : " + 
        capability.getId());
                    Map<String, List<String>> entries = 
        capability.getEntries();
                    Set<String> entryKeys = entries.keySet();
                    for (String entryKey : entryKeys) {
                        List<String> list = entries.get(entryKey);
                        System.out.print("\t\t" + entryKey + " : ");
                        for (String item : list) {
                            System.out.print(item + ", ");
                        }
                        System.out.println();
                    }
                }
            }
        }
```

This program retrieves a list of locations and iterates through it, printing the ID, name, and capabilities. The capabilities are encapsulated in a `Capability` object. This returns output similar to the following:

```
Found 6 locations(s).
Location 41, name RTP, capabilities:
        oss.instance.spec.x86_64 : oss.instance.spec.x86_64
        oss.storage.format : oss.storage.format
                RAW : raw,
                EXT3 : ext3,
        oss.storage.availabilityarea : oss.storage.availabilityarea
        oss.instance.spec.i386 : oss.instance.spec.i386
...
```

Location 41 RTP (Research Triangle Park, North Carolina) has capabilities that include these:

- 64-bit x86 architecture (`oss.instance.spec.x86_64`)
- 32-bit x86 architecture (`oss.instance.spec.i386`)
- Storage formats RAW and ext3 (`oss.storage.format`)

Working with Images

To save an instance, use the method `DeveloperCloudClient.saveInstance()`, as in this example:

```
public void saveInstance() throws UnauthorizedUserException,
UnknownErrorException, IOException,
      InsufficientResourcesException, InvalidConfigurationException,
PaymentRequiredException, UnknownInstanceException {
      System.out.println("Saving an instance");
      Image image = client.saveInstance("50767", "LAMP Server",
            "LAMP with VNC enabled");
      System.out.println("ID: " + image.getID());
}
```

The `DeveloperCloudClient.saveInstance` method takes three parameters: the instance ID, a name for the new image, and a description for the instance. You can think of this method as a request to save an image. It returns an `Image` object, but because the image has not been saved at this point, only basic information is available. However, importantly, the image ID can be extracted from the object, as the previous code shows. The output from the program looks similar to this:

```
Saving an instance
ID: 20016616
```

You can use the previous `describeMyInstances` method to check the status. Wait until the status shows `AVAILABLE` before you try to use the image or delete the instance it was created from. The method `cloneImage` copies the RAM asset of an image without copying the image file itself. This can be useful if you want to customize the metadata, startup scripts, or other files loaded to the image at provisioning time without actually changing the base image file. For example, you could clone a base image and add a software bundle to the clone without starting and saving an instance. It returns with the clone immediately in an available state. The following program demonstrates the `cloneImage()` method:

```
public void cloneImage() throws Exception {
      String cloneId = client.cloneImage("20017142", "LAMP Clone", "A
description");
      System.out.println("Cloned image id: " + cloneId);
      Image image = client.describeImage(cloneId);
      System.out.println("Name: " + image.getName());
      System.out.println("Description: " + image.getDescription());
      System.out.println("State: " + image.getState());
}
```

This gives the following output:

```
Cloning image
Cloned image id: 20018355
```

```
Name: LAMP Clone
Description: A description
State: AVAILABLE
```

You can delete an instance with the method `DeveloperCloudClient.deleteInstance()`, which takes the instance ID as the only parameter.

Uploading Files When Creating a New Instance

After saving an instance, you can customize the metadata associated with the instance. The section "Creating and Customizing Images" in Chapter 6, "Cloud Services and Applications," describes this. You might want to upload a file when creating a new instance. For example, you might want to pass in a license key to an instance containing commercial software. You can edit the `parameters.xml` file stored in Rational Asset Manager to have a parameter of type `file`, as shown in this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="parameters.xsd">
   <firewall>
       <rule>
          <source>0.0.0.0/0</source>
          <minport>1</minport>
          <maxport>65535</maxport>
       </rule>
   </firewall>
   <field name="key" type="file" label="SSH key"
          description="SSH key for Hadoop"/>
</parameters>
```

The `field` element specifies that a file should be supplied and that the name of the parameter containing the file should be `key`. The following method requests a new instance of the image and supplies the file to upload:

```
public void createInstanceFileUpload() throws
            UnauthorizedUserException, UnknownErrorException,
IOException,
            InsufficientResourcesException,
InvalidConfigurationException,
             PaymentRequiredException {
      System.out.println("Provisioning a new instance");
      Address address = null;
      Volume volume = null;
      Address[] secondaryAddresses = new Address[0];
      Map<String,Object> parameters = new HashMap<String,Object>();
      File attachFile = new File("d:/temp/key");
```

```
            parameters.put("key", attachFile);
            List<Instance> instances = client.createInstance(
                  "WebSphere Server",             // Name of instance
                  "41",                           // Data center ID
                  "20017833",                     // Image ID
                  "BRZ32.1/2048/60*175",          // Instance type
                  "alex-prod",                    // Key
                  address,                        // Address
                  volume,                         // Volume
                  parameters,                     // Options
                  null,                           // VLAN ID
                  secondaryAddresses,             // Secondary IP addresses
                  true);                          // Minimum ephemeral
            Instance instance = instances.get(0);
            System.out.println("ID: " + instance.getID());
            System.out.println("Request ID: " + instance.getRequestID());
      }
```

The program uploads the file `d:/temp/key`. After the instance is provisioned, the file is placed in a directory described in the file, placed in the instance at `/etc/cloud/parameters.xml`. For this example, the file looks similar to this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<parameters>
  <field description="SSH key for Hadoop" label="SSH key"
         name="key" type="file">
    <values><value>/home/idcuser/cloud/key</value></values>
  </field>
  <firewall>
    <rule><source>0.0.0.0/0</source>
    <minport>1</minport><maxport>65535</maxport></rule>
  </firewall>
</parameters>
```

The file is located at `/home/idcuser/cloud/key` in the virtual machine instance.

Minimizing REST Calls

Making REST calls across the Internet is relatively expensive, so try to minimize these. All the methods in `DeveloperCloudClient` make REST calls except `getLocale`, `setLocale`, and `setRemoteCredentials`. Most of the other objects, such as `Address`, `Image`, `Instance`, `Location`, and `Volume`, are transfer objects that can be cached in the client application. Consider a scenario in which you need to find the price of an instance for a list of instances. The only way to find out the price is via the `Image` object. If you start with an `Instance` object, you need to make a REST call to find the associated `Image`, run through a loop to find the matching instance size, and finally get the price. Without caching, if you do that for a list of *n* instances, it would take *n* REST calls, one for each lookup of the `Image` object. The following code example avoids that by using caching of the `Image` objects in a prior call:

```
// Find all images and put them in a map
List<Image> images = client.describeImages();  // REST call
System.out.println("Found " + images.size() + " image(s)");
Map<String, Image> imageMap = new HashMap<String, Image>();
      for (Image image: images) {
            imageMap.put(image.getID(), image);
      }

// Get a list of instances
List<Instance> instances = client.describeInstances();  // REST call
System.out.println("Found " + instances.size() + " instance(s).");
for (Instance instance: instances) {
      String instanceTypeID = instance.getInstanceType();
      Image image = imageMap.get(instance.getImageID());
      List<InstanceType> supportedTypes =
image.getSupportedInstanceTypes();
      for (InstanceType instanceType: supportedTypes) {
            if
(instanceType.getId().trim().equals(instanceTypeID.trim())) {
                  String label = instanceType.getLabel();
                  PriceDetails priceDetails = instanceType.getPrice();
                  double price = priceDetails.getRate();
                  String name = instance.getName();
                  System.out.println("Price for instance " +
                  name + " supported type <" +
                  label + "> with price " + price);
            }
      }
}
```

The code first gets the whole image catalog and caches it in the map `imageMap`, based on image ID. Then it gets all the instances that belong to the user and finds the image ID for each. Using the image ID, the code looks up the object in the map `Image` and finds the price information. The output of the program is shown here:

```
Found 617 image(s)
Found 1 instance(s).
Price for instance RSA 8.0 Preview supported type <Bronze 64 bit> with
price 0.45
```

This example has only one instance.

Example: Developing a Maven Cloud Plug-In

The section “Build and Deployment Automation” in Chapter 2, “Developing on the Cloud,” described how to work with Maven for building J2EE applications. Maven provides an extensible framework using a plug-in mechanism. In this section, you create a Maven plug-in that looks up or creates a virtual machine on the cloud using the IBM SmartCloud API. The plug-in first tries to look up the virtual machine instance. If it succeeds, it can use this instance for provisioning the J2EE application. Otherwise, it creates a new virtual machine.

To get started, generate a sample Maven plug-in project using the command line:

```
>mvn archetype:generate -DarchetypeArtifactId=maven-archetype-mojo
-DgroupId=com.ibm.cloud.enterprise.example -DartifactId=sce-maven-
plugin -DinteractiveMode=false
```

This creates a plug-in project directory tree with the MOJO (Maven Old Java Object) type. Import the project into your IDE and add the maven core and plug-in libraries to your IDE build path. Maven plug-ins implement the `Mojo` interface, usually by extending the `AbstractMojo` class. Add the source code for the `SCEMojo` class, as shown here:

```
package com.ibm.cloud.enterprise.example;

import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
import org.apache.maven.plugin.logging.Log;

import com.ibm.cloud.api.rest.client.DeveloperCloud;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.Address;
import com.ibm.cloud.api.rest.client.bean.Instance;
import com.ibm.cloud.api.rest.client.bean.Volume;
import com.ibm.cloud.api.rest.client.exception.*;

/**
 * @goal create_instance
 */
public class SCEMojo extends AbstractMojo {

      private DeveloperCloudClient client =
DeveloperCloud.getClient();
      private Log log = getLog();
      /**
     * The user name for the cloud account
     * @parameter expression="${create_instance.user_name}"
     */
    private String user_name;

      /**
     * The password for the cloud account
     * @parameter expression="${create_instance.password}"
     */
    private String password;

      /**
     * The name of the server to lookup or create
     * @parameter expression="${create_instance.name}" default-
value="app_server"
     */
    private String name;

      /**
     * The data center to create the server in
     * @parameter expression="${create_instance.data_center}"
     */
    private String data_center;

    /**
     * The image ID to create the server with
     * @parameter expression="${create_instance.image_id}"
     */
    private String image_id;

    /**
     * The name of the SSH key to create the server with
     * @parameter expression="${create_instance.key_name}"
     */
    private String key_name;

    /**
     * The name of the WebSphere administrative user
     * @parameter expression="${create_instance.was_admin_user}"
     */
    private String was_admin_user;

      /**
```

```
          * The name of the WebSphere administrative user password
          * @parameter expression="${create_instance.was_admin_password}"
          */
         private String was_admin_password;

     }
```

The code already has the imports for the methods that you will add shortly. The class-level JavaDoc has a `@goal` tag. This is required and indicates the build goal, which is `create_instance`. The private fields `user_name`, `password`, `name`, `data_center`, `image_id`, `key_name`, `was_admin_user`, and `was_admin_password` are parameters that are passed in by users of the plug-in. Table 3.5 explains these parameters.

Table 3.5 Description of Parameters Used in Maven Plug-In

Name	Description
user_name	The user name for the cloud account
password	The password for the cloud account
name	The name of the server to look up or create (default value = `app_server`)
data_center	The data center to create the server in
image_id	The image ID to create the server with
key_name	The name of the SSH key to create the server with
was_admin_user	The name of the WebSphere administrative user
was_admin_password	The name of the WebSphere administrative user password

Classes implementing the `Mojo` interface must implement the `execute` method. Add the `execute` method as shown here:

```
public void execute() throws MojoExecutionException {
      try {
            log.info("Logging onto cloud with user name " +
user_name);
            client.setRemoteCredentials(user_name, password);
            log.info("Looking for a server with name " + name);
            List<Instance> instances = client.describeInstances();
            log.info("Found " + instances.size() + " instances");
            boolean found = false;
            for (Instance instance: instances) {
                   if ((instance.getStatus() ==
Instance.Status.ACTIVE) &&
                        instance.getName().equals(name)) {
                        log.info("Found a server with name " + name);
                        found = true;
                    }
                }
                if (!found) {
                    log.info("No server with name " + name + "
found");
                    createInstance();
                }
        } catch (Exception e) {
                log.warn(e);
            throw new MojoExecutionException(e.getMessage());
        }
}
```

The method first sets the user's credentials and then retrieves a list of virtual machines that the user owns. It iterates over the list of virtual machines looking for an active instance with the same name as the parameter provided by the user. If it finds a matching virtual machine, you are done. Otherwise, you create a virtual machine with the `createInstance()` method, shown next. Add this method to the earlier `SCEMojo` class.

```
private void createInstance() throws InsufficientResourcesException,
                InvalidConfigurationException,
PaymentRequiredException,
                UnauthorizedUserException, UnknownErrorException,
IOException {
      Address address = null;
      Volume volume = null;
      Address[] secondaryAddresses = new Address[0];
      Map parameters = new HashMap<String,Object>();
      parameters.put("WASAdminUser", was_admin_user);
      parameters.put("WASAdminPassword", was_admin_password);
      List<Instance> newInstances = client.createInstance(
                  name, // Name of instance
                  data_center, // Data center ID
                  image_id, // Image ID
                  "COP32.1/2048/60", // Instance type
                  key_name, // Key
                  address, // Address
                  volume, // Volume
                  parameters, // Options
                  null, // VLAN ID
                  secondaryAddresses, // Secondary IP addresses
                  true); // Minimum ephemeral
      Instance instance = newInstances.get(0);
      log.info("ID: " + instance.getID());
}
```

The method creates the virtual machine with the parameters the user provided and writes the instance ID to the log.

Create a `pom.xml` file for the plug-in, as shown:

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ibm.cloud.enterprise</groupId>
  <artifactId>sce-maven-plugin</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>maven-plugin</packaging>
  <name>sce-maven-plugin Maven Mojo</name>
  <url>http://ibm.com/cloud/enterprise</url>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-plugin-api</artifactId>
      <version>2.0</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>commons-httpclient</groupId>
      <artifactId>commons-httpclient</artifactId>
      <version>3.1</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>1.1.1</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>commons-codec</groupId>
      <artifactId>commons-codec</artifactId>
      <version>1.3</version>
```

```
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.3</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>com.ibm.cloud.enterprise</groupId>
            <artifactId>DeveloperCloud_API_Client</artifactId>
            <version>1.4.1</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
</project>
```

The `pom.xml` file uses a `maven-plugin` package type. It defines a `maven-plugin-api` dependency and dependencies on the IBM SmartCloud Enterprise Java API and its dependent libraries, Apache HTTP Client, Commons, Logging, Codec, and Lang. Maven can automatically download the Apache libraries from the Maven central repository. Download and install the SmartCloud API library into your local repository with this command:

```
mvn install:install-file -
Dfile=/opt/sce/DeveloperCloud_API_Client_1.4.1.jar -
DgroupId=com.ibm.cloud.enterprise
-DartifactId=DeveloperCloud_API_Client -Dversion=1.4.1 -Dpackaging=jar
```

Be sure to adjust the file parameter to be consistent with your local environment.

Build and install the plug-in into your local repository with these Maven commands:

```
>mvn package
>mvn install
```

Now you can use the plug-in in a project. Generate the project cloud-app with the Maven `generate` command:

```
mvn archetype:generate -DgroupId=com.ibm.cloud.enterprise -
DartifactId=cloud-app -DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

Import a new project into your IDE. Modify the `pom.xml` generated to add a dependency and plug-in configuration, as shown here:

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.ibm.cloud.enterprise</groupId>
<artifactId>cloud-app</artifactId>
<version>1.0-SNAPSHOT</version>
<name>cloud-app</name>
<url>http://maven.apache.org</url>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.ibm.cloud.enterprise</groupId>
    <artifactId>sce-maven-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>com.ibm.cloud.enterprise</groupId>
      <artifactId>sce-maven-plugin</artifactId>
      <version>1.0-SNAPSHOT</version>
      <configuration>
        <user_name>a.user@example.com</user_name>
        <password>****</password>
        <name>was</name>
        <data_center>101</data_center>
        <image_id>20015399</image_id>
        <key_name>july26</key_name>
        <was_admin_user>wasadmin</was_admin_user>
        <was_admin_password>***</was_admin_password>
      </configuration>
      <executions>
        <execution>
          <phase>compile</phase>
          <goals>
            <goal>create_instance</goal>
          </goals>
        </execution>
      </executions>
```

```
          </plugin>
        </plugins>
      </build>
    </project>
```

Notice the use of the parameters in the configuration section.

Now you are ready to try the plug-in. First check with a new instance and existing instance. Invoke the plug-in with the Maven goal shown here:

```
>mvn com.ibm.cloud.enterprise:sce-maven-plugin:1.0-
SNAPSHOT:create_instance
```

If you do not already have a virtual machine with the given name, you should see output similar to this:

```
[info] Logging onto cloud with user name a.user@example.com
[info] Looking for a server with name was
[info] Found 4 instances
[info] No server with name was found
[info] ID: 112332
[INFO] ----------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
...
```

If you already have a virtual machine with the given name, you should see output similar to the following. Remember that if you just executed the previous command to create a server, it takes some time to create. We discuss this challenge shortly.

```
...
[INFO] --- sce-maven-plugin:1.0-SNAPSHOT:create_instance (default-cli)
@ cloud-a
pp ---
[info] Logging onto cloud with user name aamies@cn.ibm.com
[info] Looking for a server with name was
[info] Found 5 instances
[info] Found a server with name was
[INFO] ----------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------
...
```

It was good to see how a cloud can fit into the development and deployment lifecycle, but you have to complete a few more steps to use this in a truly useful way. You saw earlier that it takes some time to create a server; you need a polling mechanism to wait for it to complete. The additional challenge here is to wait for the application server itself to come up before continuing with other actions that depend on it. You can use the WebSphere wasadmin APIs for this. When you have an active server with the application server running, you can deploy your J2EE application to it. When you are finished with your WebSphere server, you then can delete it. You can add that into the clean phase of the lifecycle or somewhere else.

Further steps that you can take include these:

- Creating a private Maven repository on the cloud
- Creating a Cloud API project template

You can do this by creating a Maven archetype, which would automatically add dependent JARs.

REST API

REST APIs have become popular due to their simple programming model, especially for lightweight application programming languages like Python, PHP, and Ruby. This section provides background on the IBM SmartCloud Enterprise REST API and using it to create a simple web application that manages cloud resources using PHP. It also gives some simple examples of Java programs that invoke the REST APIs.

Background

The IBM SmartCloud Enterprise REST API enables you to manage cloud resources with any language that supports HTTP. Many tools invoke REST APIs. In Java, one possibility is the Apache HttpClient, which you saw was the library need for the IBM SmartCloud Enterprise command-line tool and Java client. Another option in Java is the JAX-RS API. PHP provides the cURL function, based on `libcurl`, which is a standard part of PHP 5. Another option in PHP is the `HTTP` class, which is a PECL extension for PHP 5. We use PHP for these examples. Java programmers can more conveniently use the Java client library provided by the IBM SmartCloud Enterprise.

You can view simple `GET` requests using a web browser. However, the IBM Cloud REST API implementation refuses requests from user agents from well-known browsers, to protect users from cross-site request forgery (CSRF) attacks that could be executed from JavaScript in web pages without their knowledge. To get around this for the purpose of development, install the Firefox User Agent Switcher plug-in. Add a new User Agent Profile using the Edit User Agents dialog shown in Figure 3.7.

Click the New button to add a new user agent, as shown in Figure 3.8.

Enter the value `cloudapi` and a name of the User Agent Profile, and click OK. You can then make REST requests using Firefox.

The REST API has a base URL of this:

www.ibm.com/computecloud/enterprise/api/rest/20100331

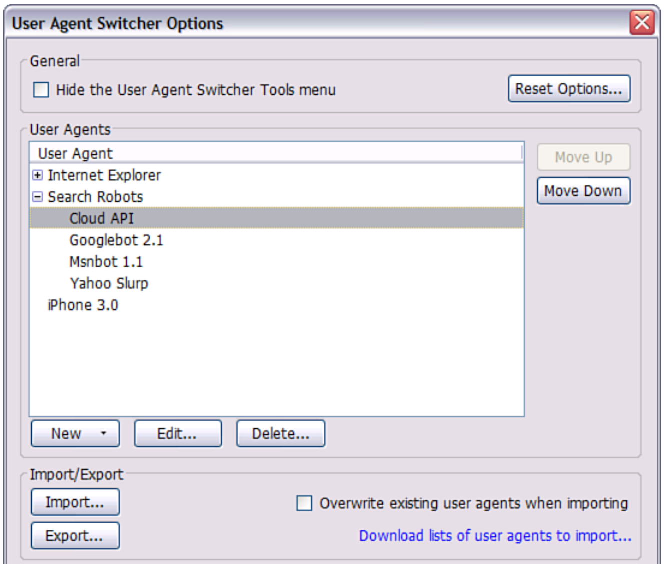
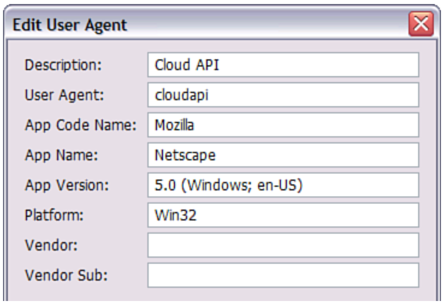


Figure 3.7 Firefox User Agent Switcher dialog box



Edit User Agent

Description:	Cloud API
User Agent:	cloudapi
App Code Name:	Mozilla
App Name:	Netscape
App Version:	5.0 (Windows; en-US)
Platform:	Win32
Vendor:	
Vendor Sub:	

Figure 3.8 New/Edit User Agent dialog box

At the time of writing, a redirect to another server is `www-147.ibm.com`. The forwarding takes some time, so it can make sense to find out the actual host and use that directly in the HTTP requests. Following the base URL, you can access the individual resource types by adding `/instances`, `/images`, and so on, as described in the REST API reference. For example, to get a list of images, enter this URL:

```
www-147.ibm.com/computecloud/enterprise/api/rest/20100331/instances
```

This returns a result similar to the following:

```
<ns2:DescribeInstancesResponse>
-
      <Instance>
            <ID>50767</ID>
            <Location>41</Location>
            <RequestID name="Java API Test Instance">51067</RequestID>
            <Name>Java API Test Instance</Name>
            <Owner>a@bc.com</Owner>
            <ImageID>20016550</ImageID>
            <InstanceType>COP64.2/4096/60</InstanceType>
            <KeyName>alex-prod</KeyName>
            <Hostname>170.224.160.133 </Hostname>
            <IP>170.224.160.133 </IP>
            ...
            <DiskSize>76</DiskSize>
      <MiniEphemeral>false</MiniEphemeral>
</Instance>
</ns2:DescribeInstancesResponse>
```

In theory, REST APIs can be executed from a web browser using asynchronous JavaScript and XML (AJAX). However, most modern web browsers, including Firefox and IE, the supported web browsers for IBM SmartCloud, allow only AJAX calls back to the server where the HTML document originated, which is the so-called "server of origin" browser security policy. That makes it difficult for you to create a web site with AJAX calls back to the IBM SmartCloud Enterprise. To work around this, you can use a proxy that serves web pages making AJAX calls back to it and relay that to calls to the SmartCloud REST APIs from the server.

cURL is a common Linux/UNIX command-line utility that is useful for making and processing HTTP requests. The Red Hat images on the IBM Cloud have cURL installed by default. For other images, you can download it from the Haxx cURL web site.

If you use cURL with a single argument representing a URL, it executes a `GET` request. For example, to get the same result as in the previous request in the browser, you can use the cURL command as follows:

```
$ curl -u a@bc.com:password --location-trusted https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/instances
```

HTTP Basic authentication is required by the IBM cloud REST API, which cURL supports via the `-u user:password` option. The `--location-trusted` option is needed to send the password to URLs you are redirected to. This is required because whatever URL you request on the IBM cloud, you are redirected to a central authentication point.

Using PHP to Invoke the IBM SmartCloud Enterprise REST APIs

To demonstrate the REST API further, we use some PHP scripts. If you have not already done so, set up the LAMP environment described in the previous chapter. PHP has a wrapper for cURL. The following PHP script calls the REST API for `list` instances.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>IBM Cloud Get Instances</title>
    </head>
    <body>
<?php
      // Include the functions for parsing the response
      include("response_parser.php");

      // Get data from cloud
      $url = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/instances";
      $ch = curl_init();
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
      curl_setopt($ch, CURLOPT_URL, $url);
      curl_setopt($ch, CURLOPT_USERPWD, "a@bc.com:password");
      curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH);
      $result = curl_exec($ch);
      curl_close($ch);

      // Parse XML response
      parseResponse($data);
?>
    </body>
</html>
PHP script to call describe instances (response_parser.php)
```

This is an HTML page with an embedded PHP script. The `curl_init()` function is called to initialize the cURL handle `$ch`. The option `CURLOPT_RETURNTRANSFER` returns a string. The `CURLOPT_USERPWD` option sets the user ID and password. Basic authentication is the default. The `CURLOPT_UNRESTRICTED_AUTH` option allows credentials to be sent to redirected pages. The `curl_exec` function executes the HTTP request. The function is shown in the following `include`.

The REST API returns either XML or JSON. You can parse XML in PHP in several ways, including with the XML Parser and SimpleXML. The following parsing utility uses the XML Parser.

```
<?php
function startElement($parser, $name, $attrs) {
    echo "$name : ";
}

function endElement($parser, $name) {
    echo "<br/>";}

function characterData($parser, $data) {
    echo $data;
}

function parseResponse($data) {
      echo "Parsing XML document<br/>";
      $xml_parser = xml_parser_create();
      xml_set_element_handler($xml_parser, "startElement",
"endElement");
      xml_set_character_data_handler($xml_parser, "characterData");
      if (!xml_parse($xml_parser, $data)) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
      }
      xml_parser_free($xml_parser);
      echo "Done<br/>";
}
?>
Parsing utility (response_parser.php)
```

The parser calls the `startElement`, `endElement`, and `characterData` functions when it encounters the start of an element, the end of an element, or character data. The `parseResponse` function is the entry point for the XML parsing. This is not intended to be a proper XML parser for the requests; it just demonstrates use of the REST API from PHP. It is an example of a SAX (Simple API for XML)-style parser, which is convenient when looking for specific tags or printing all the tags, as the previous program does. SimpleXML is an easier option if you are looking to fully parse the XML responses. SimpleXML is an example of a DOM (Document Object Model)-style parser that enables you to traverse the entire document. You can invoke the script with a browser. If you previously set up the LAMP environment on the IBM cloud, copy the `response_parser.php` and `response_parser.php` files to the directory `/var/www/html` using WinSCP. Enter the URL `http://<>/response_parser.php` into your browser. You should the output similar to this:

```
Parsing XML document
NS2:DESCRIBEINSTANCESRESPONSE : INSTANCE : ID : 50767
```

```
LOCATION : 41
REQUESTID : 51067
NAME : Java API Test Instance
OWNER : a@bc.com
IMAGEID : 20016550
INSTANCETYPE : COP64.2/4096/60
KEYNAME : alex-prod
IP : 170.224.160.133
```

The REST API can also return JSON. The advantage of this is that it might not even need to be parsed; it can just be sent to the browser as part of an HTML document and loaded by the browser's JavaScript interpreter. To specify JSON as the return format, set the HTTP Accept header as shown here:

```
Accept: application/json
```

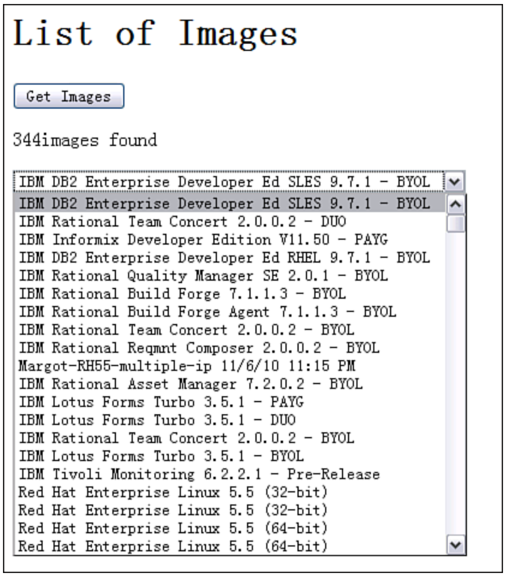
For `describe images`, this leads to output similar to the following:

```
{"images":[
  {"name":"IBM DB2 Enterprise Developer Ed SLES 9.7.1 - BYOL",
   "manifest":"https://.../parameters.xml",
   "state":1,
   "visibility":"PUBLIC",
   "owner":"SYSTEM",
   "architecture":"i386",
   "platform":"SUSE Linux Enterprise Server/11",
   "createdTime":1282466774126,
   "location":"61",
   "supportedInstanceTypes":[
     {"label":"Silver 32 bit",
      "price":{...},
      "id":"SLV32.2/4096/60*350"
    },
    ...
   ],
   "productCodes":["ehnmbYowzwCR4@zx2oBRcn"],
   "documentation":"https://.../GettingStarted.html",
   "id":"20009984",
   "description":"IBM DB2 Enterprise 9.7.0.1 ..."
   },
   ...
]};
```

You can do this in PHP with the `curl_setopt` method using the `CURLOPT_HTTPHEADER` constant. Next is an example of a PHP web page that uses JSON to build a list of images in an HTML select (drop-down) form object:

```
<!-- HTML page to get a list of images -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>IBM Cloud Get Images</title>
        <script type="text/javascript">
          function getImages() {
<?php
      // Get image data from cloud
      $url = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/offerings/image"
;
      $ch = curl_init();
      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
      curl_setopt($ch, CURLOPT_URL, $url);
      curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH, TRUE);
      curl_setopt($ch, CURLOPT_USERPWD, "a@bc:secret");
      curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept:
application/json"));
      $data = curl_exec($ch);
      curl_close($ch);

      // Put JSON response into a JavaScript variable
      echo "var jsonString = " . $data . ";\n";
      echo "var data = eval(jsonString)\n";
?>
            var div = document.getElementById('images_select');
              var images = data['images'];
              var imageString = '<p>' + images.length + 'images found
</p>';
              if (images.length > 0) {
                    imageString += '<select>';
                  for (i=0; i<images.length; i++) {
                          var image = images[i];
                          imageString += "<option id='" + image['id']
+ "'>" +                                     image['name'] +
"</option>";
                  }
                  imageString += '</select>';
              } else {
                    imageString = 'Error: ' + jsonString;
              }
              div.innerHTML = imageString;
          } // end getImages
      </script>
    </head>
    <body>
      <h1>List of Images</h1>
      <button onclick='getImages();'>Get Images</button>
      <div id='images_select'></div>
    </body>
</html>
```

All the programmatic code in this program is in the JavaScript section of the HTML `<head>` tag. It sends a REST call to the cloud using the PHP cURL API. It puts the result into a string and writes this to the JavaScript variable `jsonString`, and then uses the top-level `eval()` JavaScript function that evaluates the JSON string as an expression. Some JavaScript processing follows this to put the data returned into an HTML select widget. The generated HTML is put into the `div` tag dynamically. When the user first brings up the page, the select widget is not visible. After the user clicks on the Get Images button, the select widget displays, as in Figure 3.9.


List of Images

```
344images found

IBM DB2 Enterprise Developer Ed SLES 9.7.1 - BYOL
IBM DB2 Enterprise Developer Ed SLES 9.7.1 - BYOL
IBM Rational Team Concert 2.0.0.2 - DUO
IBM Informix Developer Edition V11.50 - PAYG
IBM DB2 Enterprise Developer Ed RHEL 9.7.1 - BYOL
IBM Rational Quality Manager SE 2.0.1 - BYOL
IBM Rational Build Forge 7.1.1.3 - BYOL
IBM Rational Build Forge Agent 7.1.1.3 - BYOL
IBM Rational Team Concert 2.0.0.2 - BYOL
IBM Rational Reqmnt Composer 2.0.0.2 - BYOL
Margot-RH55-multiple-ip 11/6/10 11:15 PM
IBM Rational Asset Manager 7.2.0.2 - BYOL
IBM Lotus Forms Turbo 3.5.1 - PAYG
IBM Lotus Forms Turbo 3.5.1 - DUO
IBM Rational Team Concert 2.0.0.2 - BYOL
IBM Lotus Forms Turbo 3.5.1 - BYOL
IBM Tivoli Monitoring 6.2.2.1 - Pre-Release
Red Hat Enterprise Linux 5.5 (32-bit)
Red Hat Enterprise Linux 5.5 (32-bit)
Red Hat Enterprise Linux 5.5 (64-bit)
Red Hat Enterprise Linux 5.5 (64-bit)
```

Figure 3.9 PHP web page to retrieve list of images

Clearly, in this example, it would be better to populate the drop-down list as soon as the page loads. Separating PHP, JavaScript, and HTML better would also improve the code. We work on this in the next example.

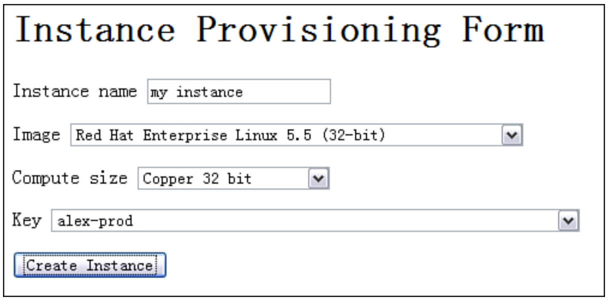
Example: create instance Form

The `create instance` REST API is a `POST` request to the `/instances` relative URL that takes a names for the instance, the image ID, the compute size, the location, and the name of a key. The following HTML page collects the information for a `create instance` request.

```
<!-- create_instance_wizard.php HTML page to provision instances -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>IBM Cloud Instance Provisioning Wizard</title>
<?php
      include("get_imagesjs.php");
      include("get_keysjs.php");
?>
          <script type="text/javascript"
src="create_instance_wizard.js"></script>
          <script type="text/javascript"
src="create_instance_image.js"></script>
          <script type="text/javascript"
src="create_instance_compute.js"></script>
          <script type="text/javascript"
src="create_instance_key.js"></script>
    </head>
    <body>
      <h1>Instance Provisioning Form</h1>
      <form action='create_instance_submit.php' method='POST'>
        <p>
          <label for='name'>Instance name</label>
          <input  id='name' name='name' type='text'/>
        </p>
        <p>
          <label for='images_select'>Image</label>
          <select  id='images_select' name='imageID'></select>
        </p>
        <p>
          <label for='images_select'>Compute size</label>
          <select id='compute_select'
name='instanceType'><option>Select an image first</option></select>
        </p>
        <p>
          <label for='key_select'>Key</label>
          <select id='key_select' name='publicKey'><option>Select a
key</option></select>
        </p>
```

```
            <p>
              <input id='compute_select_button' type='submit'
value='Create Instance'/>
              <input id='location_hidden' name='location' type='hidden'/>
            </p>
          </form>
        </body>
    </html>
```

The page has two PHP `include` statements for REST calls to query for the images and keys available. They are followed by four JavaScript imports to initialize the page and to contain functions to respond to events on the page. The HTML form contains a text field to enter the image name, a select element to select the image type, a select element for the compute size, a select element for the key, a button to submit the form, and a hidden field to contain the instance location. Figure 3.10 shows the page.



Instance Provisioning Form

Instance name: my instance

Image: Red Hat Enterprise Linux 5.5 (32-bit)

Compute size: Copper 32 bit

Key: alex-prod

Figure 3.10 Instance provisioning form

The user should fill in a name for the instance, select an image, select a compute size, select a key, and click the Create Instance button. The image must be selected before the compute size can be populated because you need to show the list of supported computed sizes for the particular image. You determine the data center from the image. You get an error if you try to submit a `create instance` request for a data center that does not match the image. The PHP `include` makes the REST call to get an array of images.

```
<!-- get_imagesjs.php JavaScript function to get an array of images --
>
<script type="text/javascript">
      // Returns an array of images
      function getImages() {
<?php
            // Get image data from cloud
            $url = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/offerings/image"
;
            $ch = curl_init();
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
            curl_setopt($ch, CURLOPT_URL, $url);
            curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH, TRUE);
            curl_setopt($ch, CURLOPT_USERPWD, "a@bc.com:secret");
            curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept:
application/json"));
          $data = curl_exec($ch);
            curl_close($ch);

            // Put JSON response into a JavaScript variable
            echo "var jsonString = " . $data . ";\n";
            echo "var data = eval(jsonString)\n";
?>
            return data['images'];
      } // end getImages
</script>
```

It is similar to the use of the `describe images` function discussed previously. However, it uses the `HTTPHEADER` cURL option to specify that JSON should be returned. It writes the JSON data to a variable and returns it from the JavaScript function. The `get_keysjs.php include` makes the REST call to get the list of keys:

```
<!-- get_keysjs.php JavaScript function to get an array of keys using
a REST call -->
<script type="text/javascript">
      // Returns an array of keys
      function getKeys() {
<?php
            // Get image data from cloud
            $url = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/keys";
            $ch = curl_init();
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
            curl_setopt($ch, CURLOPT_URL, $url);
            curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH, TRUE);
            curl_setopt($ch, CURLOPT_USERPWD, "a@bc.com:secret");
            curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept:
application/json"));
          $data = curl_exec($ch);
            curl_close($ch);
            // Put JSON response into a JavaScript variable
            echo "var keyJsonString = " . $data . ";\n";
            echo "var keysData = eval(keyJsonString)\n";
?>
            return keysData['keys'];
      } // end getImages
</script>
```

This function works in a similar way to `getImages`. However, the URL is `/keys`. The file `create_instance_wizard.js` initializes the JavaScript for the page:

```
//create_instance_wizard.js

// global variables
var imagesSelect = null;
var imageArray = null;
var selectedImage = null;
var keyArray = null;

// Initializes the page
window.onload = function() {

      // Populate the images drop down
      imagesSelect = document.getElementById('images_select');
      imageArray = getImages();
      populateImageDropdown(imagesSelect, imageArray);

      // Wire the processing function for the image select
      imagesSelect.onchange = populateComputeDropdown;

      // Wire the processing function for the compute size select
      var computeSelectButton =
document.getElementById('compute_select_button');

      // Populate the keys drop down
      keyArray = getKeys();
      populateKeySelect(keyArray);

}
```

The script defines several global variables: `imagesSelect` contains the DOM element for the select element listing the image options. This is needed several times. The variable `imageArray` contains the array of images returned from the REST call. The `keyArray` variable contains the keys returned from the REST call to get the list of keys. Next, the script defines the `window.onload` function listing the code to be executed after the page loads. It calls the `getImages` function to parse the list of images and the `getKeys` function to get the list of keys.

The create_instance_image.js script populates the image select with the array of images from the catalog:

```
//create_instance_image.js
// select The select element to populate
// images The array of images
function populateImageDropdown(select, images) {
      var imageString = "<option>Select an image</option>";
      if (images.length > 0) {
            var i = 0;
            for (i=0; i<images.length; i++) {
                  var image = images[i];
                  imageString += "<option value='" + image['id']
+ "'>" + image['name'] + "</option>";
            }
      } else {
            alert('No images found');
      }
      select.innerHTML = imageString;
} // end populateImageDropdown
```

The script iterates through the image array returned from the REST call and creates HTML option elements, which it inserts into the image select element. The value of each option is the image ID, and the text displayed is the image name. Other data here might be interesting to the user, such as description and data center, but we have omitted it, to keep the example simple. The create_instance_compute.js script populates the compute size select widget (drop-down) with data. It is called whenever the user changes the selected option in the image select element.

```
// create_instance_compute.js
// Populate the select with the compute size options
function populateComputeDropdown() {
      selectedImage = imageArray[imagesSelect.selectedIndex-1];
      var supportedInstanceTypes =
selectedImage['supportedInstanceTypes'];
      var computeSelect = document.getElementById('compute_select');
      var computeString = "<option>Select compute size</option>";
      if (supportedInstanceTypes.length > 0) {
            var j =0;
            for (j=0; j<supportedInstanceTypes.length; j++) {
                  var size = supportedInstanceTypes[j];
                  computeString += "<option value='" + size['id'] +
"'>" + size['label'] + "</option>";
            }
      } else {
            alert('No compute sizes found');
      }
      computeSelect.innerHTML = computeString;

      // Set the location field
      var locationHidden = document.getElementById('location_hidden');
      locationHidden.value = selectedImage['location'];
} // end populateComputeDropdown
```

The script determines the image selected by the user and extracts the supported instance types (compute sizes) for that image from the `describe images` JSON data. Then the function creates an HTML option for each compute size. Finally, the function finds the location associated with the selected image and sets it to a hidden field so that it can be used in the `create instance` request. The script `create_instance_key.js` populates the key selection element.

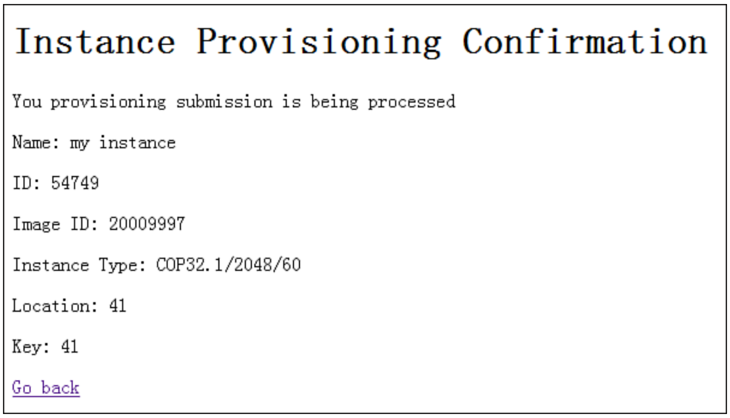
```
      //create_instance_key.js
      // Create the key drop down
      // keyArray An array of key objects
      function populateKeySelect(keyArray) {
            var keySelect = document.getElementById('key_select');
            var keyString = "<option>Select key</option>";
            if (keyArray.length > 0) {
                  var k =0;
                  for (k=0; k<keyArray.length; k++) {
                        var key = keyArray[k];
                        keyString += "<option id='" + key['keyName'] + "'>"
      + key['keyName'] + "</option>";
                  }
            } else {
                  alert('No keys found');
            }
            keySelect.innerHTML = keyString;
      } // end populateKeySelect
```

The JavaScript function `populateKeySelect` iterates over the key objects in the array `keyArray` and creates an HTML option element for each one. Finally, it inserts the option elements into the `key select` element. When the user clicks the Create Instance button, the script is submitted to the `create_instance_submit.php` script.

```
      <!-- create_instance_submit.php Processes a create instance submission
      -->
      <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
      <html>
          <head>
              <title>IBM Cloud Instance Provisioning Wizard</title>
              <script type='text/javascript'>
```

```
<?php
      include('create_instancejs.php');
?>
            </script>
            <script type='text/javascript'
src='create_instance_submit.js'></script>
    </head>
    <body>
      <h1>Instance Provisioning Confirmation</h1>
      <p>You provisioning submission is being processed</p>
        <p>Name: <span id='name_span'/></p>
        <p>ID: <span id='instance_id_span'/></p>
        <p>Image ID: <span id='image_id_span'/></p>
        <p>Instance Type: <span id='instance_type_span'/></p>
        <p>Location: <span id='location_span'/></p>
        <p>Key: <span id='key_span'/></p>
      <p><a href='create_instance_wizard.php'>Go back</a></p>
    </body>
</html>
```

This page includes the PHP script `create_instancejs.php` to submit the `create instance` request using the REST API. Then it imports the `create_instance_submit.js` JavaScript file. The body of the page contains placeholders for the return data from the `create instance` request. These are filled in using JavaScript after the page loads. The page generated by these scripts looks like Figure 3.11.


Instance Provisioning Confirmation

You provisioning submission is being processed

Name: my instance

ID: 54749

Image ID: 20009997

Instance Type: COP32.1/2048/60

Location: 41

Key: 41

Go back

Figure 3.11 Create instance submit confirmation page

The `create_instancejs.php` script collects the form values submitted to the page:

```
<?php
// create_instancejs.php
// Collect the form variables
$name = $_POST['name'];
$imageID = $_POST['imageID'];
$instanceType = $_POST['instanceType'];
$location = $_POST['location'];
$publicKey      = $_POST['publicKey'];

// Call the REST API
$url = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/instances";
$ch = curl_init();
$request_body = 'name=' . urlencode($name) . '&imageID=' .
       urlencode($imageID) . '&' . 'instanceType=' .
      urlencode($instanceType) . '&location=' . urlencode($location) .
      '&publicKey=' . urlencode($publicKey);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH, TRUE);
curl_setopt($ch, CURLOPT_USERPWD, "aamies@cn.ibm.com:hsing1yun");
curl_setopt($ch, CURLOPT_POST, TRUE);
curl_setopt($ch, CURLOPT_POSTFIELDS, $request_body);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept:
application/json"));
$data = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);

// Check HTTP response and put JSON response into a JavaScript
variable
echo "var httpCode = $httpCode;\n";
if ($httpCode != 200) {
      error_log("HTTP response: $httpCode, error message: " . $data);
      echo "var responseData = '" . trim($data) . "';\n"; // error
message
} else {
      echo "var responseData = $data;\n";
      echo "var instances = responseData['instances'];\n";
      echo "var instance = instances[0];\n";
}
?>
```

First, the script uses the PHP `$_POST` environment variable to collect the submitted data. The array names match the HTML form element names. The REST call is similar to others you have seen, except that the URL is different and the HTTP `POST` method is used. The URL is set to `/instances`. The name, image ID, instance type (compute size), location, and key variables are added to the request body in preparation for calling the REST API. The standard PHP function `urlencode` is to escape special character values in the data. The cURL `CURLOPT_POST` option specifies that the HTTP method is `POST`. Finally, the result is written to the JavaScript variable `responseData`, and the HTTP response code is written to the JavaScript variable `httpCode`. The JSON returned from the `create instance` REST call looks similar to this:

```
{"instances":[
  {"launchTime":1299536879688,
   ...
   "requestId":"55032",
   "keyName":"mykey",
   "name":"my instance",
   "instanceType":"COP32.1/2048/60",
   "status":0,
   "owner":"a@bc.com",
   "location":"41",
   "imageId":"20009997",
   "volumes":[],
   "requestName":"my instance",
   "id":"54732",
   "secondaryIP":[],
   "expirationTime":1362608888650}
]};
```

You can use this data to give the user immediate feedback. In addition, you can use the `requestId` or the instance `id` to check the status of the request or the instance later to see if it is ready for use. The response data is processed with the `create_instance_submit.js` script:

```
// create_instance_submit.js
// Initializes the page
window.onload = function() {
      if (httpCode != 200) {
            alert("Response Bad: " + responseData);
      } else {
            var nameSpan = document.getElementById('name_span');
            nameSpan.innerHTML = instance['name'];
            var instanceIdSpan =
document.getElementById('instance_id_span');
            instanceIdSpan.innerHTML = instance['id'];
            var imageIdSpan =
document.getElementById('image_id_span');
```

```
                    imageIdSpan.innerHTML = instance['imageId'];
                    var instanceTypeSpan =
                          document.getElementById('instance_type_span');
                    instanceTypeSpan.innerHTML = instance['instanceType'];
                    var locationSpan =
        document.getElementById('location_span');
                    locationSpan.innerHTML = instance['location'];
                    var keySpan = document.getElementById('key_span');
                    keySpan.innerHTML = instance['location'];
              }
        }
```

The script delays execution until after the page loads by defining the script to be executed in the `window.onload` function. The script first checks the HTTP response code. If it is `OK` (`200`), it extracts the JSON response data for the different fields and sets them into the placeholders of the HTML page.

When you test, make sure you choose an image type with no additional input parameters necessary. Otherwise, you will get an error message similar to this:

```
Error 412: Invalid db2inst1 password. Must contain at least 1 number,
at least 1 lower case letter, and at least 1 upper case letter.
```

To handle this case, you need to parse and examine the `parameters.xml` file for the image. See [Vernier, 2011, "Convert IBM Cloud Image Parameters into Java Using JAXB"] for details on this.

That concludes this lengthy example. Several points are worth noting. First, a lot of work was needed to handle the basics of the page behavior using JavaScript. Open source libraries can help with this; one of the most useful basic libraries is the Prototype JavaScript framework. Second, the pages look very plain; you could write your own CSS stylesheets to tidy them up. Taking a further step, the HTML widgets and page can be made to look more professional and act more nicely using one of the open source JavaScript widget sets, such as the Dojo Toolkit. Third, the code is not very robust. It omits a lot of error handling and validation, to keep the example as simple as possible. Fourth, the user interface displays many alphanumeric IDs, which is hardly user friendly; you should replace these with names by looking up the appropriate API. Fifth, the example makes the REST calls in series. The `create instance` form page has two REST calls, one for getting a list of images and one for getting a list of keys. Not until the `describe images` call returns do you call `describe keys`. If you added storage volumes and IP addresses to the form, you would have to make four REST calls before showing the page to the user.

Example: Page to Show a List of Instances

This example describes an example PHP page that shows a list of instances. The example also addresses some of the issues with the last example. We make requests for data from the browser using Asynchronous JavaScript and XML (AJAX); however, we do not actually use XML. We use JSON because it is easier to handle in the browser JavaScript code.

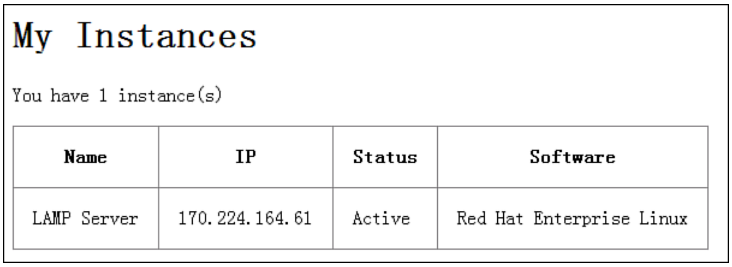
The Prototype JavaScript framework enables JavaScript developers to code in an object-oriented way. The framework includes an AJAX class to handle the low-level HTTP request and response handling, and to isolate the developer from the differences between browsers. To use Prototype, download it from the project web site and copy the file `prototype.js` to your web server. Add a line like this in the header section of your page:

```
<script type='text/javascript' src='prototype.js'></script>
```

The page `list_instances.html`, shown next, shows a list of virtual machines instances:

```
<!-- list_instances.html
     HTML page to get a list of instances -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
        <title>My Instances</title>
            <script type='text/javascript'
src='prototype.js'></script>
            <script type='text/javascript'
src='list_instances.js'></script>
            <script type='text/javascript'
src='list_rest.js'></script>
            <script type='text/javascript'
src='list_populate.js'></script>
            <link rel="stylesheet" type="text/css"
href="examples.css"/>
    </head>
    <body>
      <h1>My Instances</h1>
      <p id='instance_count_p'></p>
      <table>
        <tbody id='instance_list_tbody'></tbody>
      </table>
    </body>
</html>
```

Most of the work is done by the `list_instances.js`, `list_rest.js`, and `list_populate.js` scripts, referenced in the head section. The body contains a placeholder for the count of the number of instances and a table listing the instances. The page looks as shown in Figure 3.12.


My Instances

You have 1 instance(s)

Name	IP	Status	Software
LAMP Server	170.224.164.61	Active	Red Hat Enterprise Linux

Figure 3.12 List instances HTML page

The `list_instances.js` script initializes the page.

```
// list_instances.js

// Initializes the page
window.onload = function() {
      callREST('instances', populateTable);
}
```

When the page has loaded, it calls the `callREST` function with the arguments `instances` and `populateTable`. The `callREST` function is in the `list_rest.js` script.

```
// list_rest.js

// JavaScript function to initiate the REST API call
// resource The kind of resource to list (instances, locations, or
images)
// functionToCall The function to call after receiving the data
function callREST(resource, functionToCall) {
      new Ajax.Request('/list_rest.php', {
            method:'get',
            parameters: {resource: resource},
            onSuccess: function(transport){
                var response = transport.responseText || 'no response
text';
                var data = transport.responseText.evalJSON();
                functionToCall(data);
            },
            onFailure: function(transport){
                  alert(transport.responseText || 'no response text');
            }
      });
}
```

The `callREST` function makes an AJAX call to the `list_rest.php` PHP script, which retrieves a list of resources—these can be instances, images, or any other kind of entity that the REST API supports. You also can reuse it for other purposes. The `resource` parameter indicates the type of entity to retrieve. The script uses the Prototype AJAX class, supplying functions to call on success and on failure. The second parameter to the `callREST` function is the name of a function to call with the JSON data returned. This is invoked with the data returned from the AJAX call.

When sending JSON content from your server to a browser, use the HTTP header shown here:

```
Content-type: application/json
```

The data returned in the AJAX calls looks similar to this:

```
{'instances':[
  {'launchTime':1299239330335,
   'software':[{
     'version':'5.4',
     'type':'OS',
     'name':'Red Hat Enterprise Linux'}],
   'primaryIP':{'hostname':'myhost','ip':'170.224.164.61','type':0},
   'requestId':'53865',
   'keyName':'mykey',
   'name':'LAMP Server',
   'instanceType':'COP64.2/4096/60',
   'status':5,
   'owner':'a@bc.com',
   'location':'41',
   'imageId':'20016733',
   'root-only':'false',
   'productCodes':[],
   'volumes':[],
   'requestName':'LAMP Server',
   'id':'53565',
   'secondaryIP':[],
   'expirationTime':1362311339164,
   'diskSize':'120'}
]}
```

We are most interested in the `name`, `software`, `ip`, and `status` fields. The `list_rest.php` PHP script, shown next, makes the REST call.

```
<?php
// list_rest.php
// PHP script to make a rest call to describe the resource passed in
the GET parameter,
// including instances, locations, images, etc
$resource = $_GET['resource'];
error_log("resource $resource");
$baseUrl = "https://www-
147.ibm.com/computecloud/enterprise/api/rest/20100331/";
$url = $baseUrl . $resource;
$ch = curl_init();
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_USERPWD, "aamies@cn.ibm.com:hsing1yun");
curl_setopt($ch, CURLOPT_UNRESTRICTED_AUTH, TRUE);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept:
application/json"));
$data = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);

// Check HTTP response and put JSON response into a JavaScript
variable
if ($httpCode != 200) {
      error_log("HTTP response: $httpCode, error message: " . $data);
      header("HTTP/" . $httpCode . ' ' .  trim($data)); // error
message
} else {
      header('Content-type: application/json');
      echo trim($data);
}
?>
```

The script takes one parameter, `resource`, which has the same purpose as described earlier. In this case, the value is `instances`. The script adds this to the base URL to determine which REST API to use. It works only for APIs that use HTTP GET and do not take any parameters. The cURL use is similar to the earlier `create instance` example. Finally, the JSON response is relayed to the browser. In the case of an error, the HTTP response is relayed as well.

The `list_populate.js` script populates an HTML table with the data.

```
// list_populate.js

// JavaScript function to process the result returned from the REST
call
// Populates the table listing instances
function populateTable(instancesObject) {
      var instances = instancesObject['instances'];
      var summaryStr = '';
      if (instances.length == 0) {
```

```
                summaryStr = 'You do not have any instances';
                $('instance_count_p').innerHTML = summaryStr;
        } else {
                summaryStr = 'You have ' + instances.length + '
instance(s)';
                $('instance_count_p').innerHTML = summaryStr;
                var tableStr =
'<tr><th>Name</th><th>IP</th><th>Status</th><th>Software</th></tr>';
                for (var i=0; i<instances.length; i++) {
                        var instance = instances[i];
                        var primaryIP = instance['primaryIP'];
                        var status = instance['status'];
                        var softwareList = instance['software'];
                        if (softwareList.length > 0) {
                                software = softwareList[0];
                        }
                        tableStr += '<tr><td>' + instance['name'] + '</td>'
+
                                        '<td>' + primaryIP['ip'] + '</td>' +
                                        '<td>' + statusLabel[status] + '</td>' +
                                        '<td>' + software['name'] +
'</td></tr>';
                }
                $('instance_list_tbody').innerHTML = tableStr;
        }
}

// Instance Status labels
var statusLabel = ['New','Provisioning','Failed','Removed',
'Rejected', 'Active', 'Unknown', 'Deprovisioning', 'Restarting',
'Starting', 'Stopping', 'Stopped', 'Deprovisioning Pending',
        'Restart Pending'];
```

The `populateTable` function takes one argument, which is the instantiated JSON data from the AJAX call. It uses the Prototype `$('element_id')` function to look up an HTML element based on the element ID. This is a compact way to retrieve the placeholder HTML elements. The status values are returned from the REST API as integer values. The REST API reference guide lists the meaning of the difference values; these are listed in the `statusLabel` array and used as labels in the table using a lookup from the array.

Using Java to Invoke the IBM SmartCloud Enterprise REST APIs

Even though there is a Java API for the IBM SmartCloud Enterprise, you might have an occasion to use the REST API with Java (for example, when creating a proxy for the API). This section gives several simple examples using Java; they make use of the Apache HttpClient 3.1 library. You can find a list of instances with the following code:

```
public int getInstances() throws HttpException, IOException {
      System.out.println("Getting instances");
      HttpClient httpclient = getHttpClient();
      GetMethod method = getGetMethod(baseUrl + "instances/");
      try {
            int statusCode = httpclient.executeMethod(method);
            printBody(method);
            System.out.println("Status: " + statusCode);
            return statusCode;
      } finally {
            method.releaseConnection();
      }
}
```

The output of this method is similar to the output of the earlier PHP examples. The method is a utility method shown here:

```
private HttpClient getHttpClient() {
      HttpClient httpclient = new HttpClient();
      HttpState state = new HttpState();
      UsernamePasswordCredentials creds = new
UsernamePasswordCredentials(userName, password);
      state.setCredentials(AuthScope.ANY, creds);
      httpclient.setState(state);
      httpclient.getParams().setAuthenticationPreemptive(true);
      return httpclient;
}
```

The utility method `getGetMethod` is shown next:

```
private GetMethod getGetMethod(String url) {
      GetMethod method = new GetMethod(url);
      if (!xml) {
            method.setRequestHeader("Accept", "application/json");
      } else {
            method.setRequestHeader("Accept", "gzip,deflate");
      }
      return method;
}
```

You can retrieve a list of locations with this method:

```
public int getLocations() throws HttpException, IOException {
      System.out.println("Getting instances");
      HttpClient httpclient = getHttpClient();
```

```
                GetMethod method = getGetMethod(baseUrl + LOCATIONS);
                try {
                        int statusCode = httpclient.executeMethod(method);
                        printBody(method);
                        System.out.println("Status: " + statusCode);
                        return statusCode;
                } finally {
                        method.releaseConnection();
                }
        }
```

Again, the output is similar to the output in the earlier PHP examples.

Rational Asset Manager

Rational Asset Manager (RAM) is an application that manages images, software, documents, and other assets. In the context of the IBM SmartCloud Enterprise, RAM provides a framework to enable asset providers to publish assets and support a community of users of those assets. The IBM SmartCloud Enterprise also uses RAM to manage images, software, and document metadata. When image providers prepare an image, they can use RAM to store information in a manner similar to a "Getting Started" guide and to host a support forum for the users of the image; they can also take advantage of Web 2.0 capabilities, such as user ratings and tags. This is valuable to users, who can select well-used and well-supported images. RAM also enables users to browse and query assets and to personalize their user account. In addition to the RAM server hosted by IBM SmartCloud Enterprise, the catalog has a RAM image that you can use for your own projects.

You can programmatically download and manage assets in RAM using either the RAM client API or the RAM Eclipse client. Download the required libraries from the asset catalog. In the image catalog, click Help, Extensions, as in Figure 3.13.

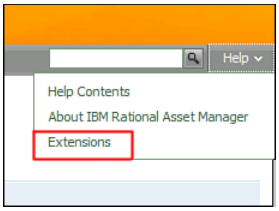
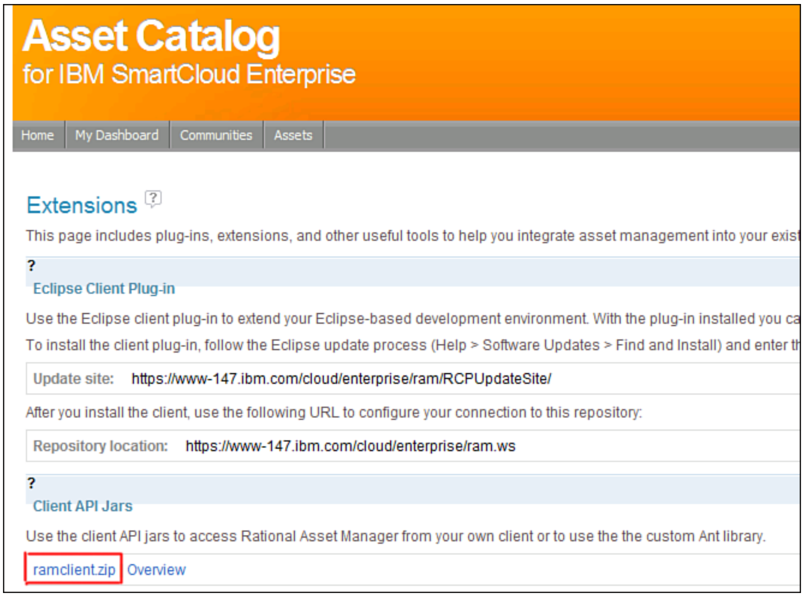


Figure 3.13 Location for downloading RAM client

Under Client API Jars, download the `ramclient.zip`, as shown in Figure 3.14. This includes all the JARs you need to support a remote client to the asset catalog.



Extensions

This page includes plug-ins, extensions, and other useful tools to help you integrate asset management into your exist

?
Eclipse Client Plug-in

Use the Eclipse client plug-in to extend your Eclipse-based development environment. With the plug-in installed you ca

To install the client plug-in, follow the Eclipse update process (Help > Software Updates > Find and Install) and enter th

Update site: https://www-147.ibm.com/cloud/enterprise/ram/RCPUpdateSite/

After you install the client, use the following URL to configure your connection to this repository:

Repository location: https://www-147.ibm.com/cloud/enterprise/ram.ws

?
Client API Jars

Use the client API jars to access Rational Asset Manager from your own client or to use the the custom Ant library.

ramclient.zip Overview

Figure 3.14 RAM extensions page

The Eclipse Client plug-in and Web Services are options as well.

Following is a simple program that uses the RAM API to retrieve an image asset based on the GUID. Notice that the image ID used in the SCE APIs is a parameter that you can access in the asset catalog.

```
package com.ibm.cloud.example.ram;

import java.util.logging.Level;
import java.util.logging.Logger;

import com.ibm.ram.client.RAMAsset;
import com.ibm.ram.client.RAMSession;
import com.ibm.ram.common.data.AssetAttribute;
import com.ibm.ram.common.data.AssetIdentification;

public class RAMClient {

      private static final String USER = "a@b.com";
      private static final String PASSWORD = "***";
      private static final String URL = "https://
```

```
www-147.ibm.com/cloud/enterprise/ram.ws";
      private RAMSession session;
      private static Logger logger =
Logger.getLogger("com.ibm.cloud.example.ram");
      private static final String LS =
System.getProperty("line.separator");

      public void connect() {
            try {
                  session = new RAMSession(URL, USER, PASSWORD);
            } catch(Exception e) {
                  logger.log(Level.SEVERE, "Error connecting", e);
            }
      }

      /**
       * Prints out information about an image asset
       * @param guid The RAM Asset globally unique identifier
       */
      public void getAsset(String guid) {
            try {
                  AssetIdentification assetID = new
AssetIdentification();
                  assetID.setGUID(guid);
                  RAMAsset asset = session.getAsset(assetID);
                  logger.log(Level.INFO, "Got asset " +
asset.getName() +
" with ID " + asset.getAssetAttribute("Image Id"));
                  AssetAttribute[] assetAttributes =
asset.getAssetAttributes();
                  StringBuilder sb = new StringBuilder();
                  for (int i=0; i<assetAttributes.length; i++) {
                        StringBuilder valueString = new
StringBuilder();
                        String[] values =
assetAttributes[i].getValues();
                        if (values != null) {
                              for (int j=0; j<values.length; j++) {
                                    valueString.append(values[j] + "
");
                              }
                        }
                        sb.append(assetAttributes[i].getName() + " : "
+ valueString + LS);
                  }
```

```
                        logger.log(Level.INFO, "Attributes: " + sb);
                  } catch(Exception e) {
                        logger.log(Level.SEVERE, "Error getting asset", e);
                  }
            }

            /**
             * Example usage: java com.ibm.cloud.example.ram.RAMClient Guid
             */
            public static void main(String[] args) {
                  RAMClient client = new RAMClient();
                  client.connect();
                  String guid = args[0];
                  client.getAsset(guid);
            }
      }
```

You can invoke the program by passing in an example GUID:

```
Java -cp $CP com.ibm.cloud.example.ram.RAMClient {6D6FF958-0EBA-7A7E-
CEC8-BE737E9ACA94}
```

Here, `$CP` is a system variable containing the classpath of the RAM and dependent libraries downloaded earlier, and the program argument is the GUID for an image in the SmartCloud Enterprise catalog. When invoked, the program gives output similar to this:

```
INFO: Got asset Microsoft Windows Server 2008 R2 (64-bit) with ID
Image Id = 20012053
INFO: Attributes: Primary Artifact :
Request Image URL : <a href="https://www-
147.ibm.com/cloud/enterprise/user/control.jsp?autocreate={id:'20012053
'}" target="_blank">Start an instance of this image</a>
Terms and Conditions : <a href="https://www-
147.ibm.com/cloud/enterprise/ram/artifact/%7B6D6FF958-0EBA-7A7E-CEC8-
BE737E9ACA94%7D/1.0/Terms.pdf" target="_blank">Link to Terms.pdf</a>
Image Id : 20012053
Operating System : Windows Server 2008
Product Name :
Support Information :
Installed Software :
Offering Deployment Automation Model :
Offering Deployment Model :
Offering Documentation :
Operating System Details : /win2k8dc-r2-
x64.topology#capability.windowsOperatingSystem[displayName='win2k8dc-
r2-x64' and name='WindowsOperatingSystem_3369' and linkType='any' and
kernelWidth='64-bit' and os.type='Windows' and os.version='2008 R2']
```

We have deleted some of the log trace information here. The image ID `20012053` is the important link to the SmartCloud Enterprise REST API.

You can extend the program with the following method to perform a generic search:

```
public void search(String queryText) {
      SearchQuery query = session.createAssetQuery(queryText);
      SearchResult searchResult = session.getAssets(query);
      RAMAssetSearchResult[] results =

(RAMAssetSearchResult[])searchResult.getAssetSearchResults();
      logger.log(Level.INFO, results.length + " results found.");
      for (int i=0; i<results.length; i++) {
            RAMAsset asset = (RAMAsset)results[i].getAsset();
             String guid = assetInfo.getIdentification().getGUID();
             String name = assetInfo.getName();
            AssetAttribute imageID = asset.getAssetAttribute("Image
Id");
            logger.log(Level.INFO, "Asset " + name + " " +
            imageID + " " + guid);
      }
}
```

The method creates a `SearchQuery` object that allows a web-style search to be executed. You invoke this method with the query text `Windows`. The output is shown here:

```
INFO: 93 results found.
INFO: Asset Microsoft Windows Server 2008 R2 (64-bit) (EHN) Image Id =
20012054 {16AD9067-E16E-6C83-A2E3-80512F3AD815}
INFO: Asset Microsoft Windows Server 2008 R2 (64-bit) Image Id =
20012053 {6D6FF958-0EBA-7A7E-CEC8-BE737E9ACA94}
INFO: Asset Microsoft Windows Server 2008 R1 (32-bit) (EHN) Image Id =
20012835 {2E49860A-930F-6052-88EF-B2212B9DB95A}
INFO: Asset Microsoft Windows Server 2003 R2 (32-bit) (EHN) Image Id =
{015B829A-5586-0775-85DB-5AAB432A5587}
...
```

These examples return all the data in the asset catalog that the user whose credentials you use has the authority to browse. This is slightly different from the `GET /offerings/image` call in the SmartCloud Enterprise REST API, listing the images the user is entitled to, considering the enterprise's onboarding contract. That is a subset of the full list of public images in the catalog.

Three levels of visibility in RAM—public, enterprise, and private—relate to three respective communities. Publicly visible assets are visible to all the users on the cloud. Shared visibility assets are visible by the enterprise communities, including all the users in each respective enterprise. A privately visible asset is visible only to the user who created it. The previous examples returned images from all levels of visibility. The example method that follows shows how to find the name of the community associated with an asset.

```
public void getAssetCommunityInfo(String guid) {
      try {
            AssetIdentification assetID = new AssetIdentification();
            assetID.setGUID(guid);
            RAMAsset asset = session.getAsset(assetID);
            logger.log(Level.INFO, "Got asset " + asset.getName() +
            " with ID " + asset.getAssetAttribute("Image Id"));
            CommunityInformation community = asset.getCommunity();
            String communityName = community.getName();
            int communityID = community.getId();
            logger.log(Level.INFO, "Attributes: " + communityName +
            ", community id: " + communityID);
      } catch(Exception e) {
            logger.log(Level.SEVERE, "Error getting asset", e);
      }
}
```

This results in output similar to the following:

```
INFO: Getting asset with guid {6D6FF958-0EBA-7A7E-CEC8-BE737E9ACA94}
INFO: Got asset Microsoft Windows Server 2008 R2 (64-bit) with ID
Image Id = 20012053
INFO: Attributes: Cloud Computing Core Community, community id: 1000
```

The asset belongs to the `Cloud Computing Core Community`—that is, it is publicly visible. The ID of this community is 1000.

With the RAM search APIs, you can find all assets matching a generic search term. The RAM API also helps you be more precise in matching specific attributes and either OR'ing or AND'ing different terms together, paging results, and so on. In addition, you can filter searches based on communities. Consider an example:

```
public void search(String queryText, String communityName) {
      RAMAssetQueryBuilder queryBuilder = new
RAMAssetQueryBuilder(session);
      CommunityInformation community = new CommunityInformation();
      community.setName(communityName);
      queryBuilder.addSearchFilter(community);
      queryBuilder.addQueryTextField("*" + queryText + "*");
      SearchResult searchResult = session.getAssets(queryBuilder);
      RAMAssetSearchResult[] results =
       (RAMAssetSearchResult[])searchResult.getAssetSearchResults();
      logger.log(Level.INFO, results.length + " results found.");
      for (int i=0; i<results.length; i++) {
              RAMAsset asset = (RAMAsset)results[i].getAsset();
              String guid = asset.getIdentification().getGUID();
              String name = asset.getName();
```

```
                AssetAttribute imageID = asset.getAssetAttribute("Image
Id");
            logger.log(Level.INFO, "Asset " + name + " " +
            imageID + " " + guid);
        }
}
```

The example uses a `RAMAssetQueryBuilder` object to construct a query with two conditions: a query with the input text and a filter based on the community. The output based on invoking the method with the query text of `Windows` and a community name of `Cloud Computing Core Community` is shown here:

```
INFO: Getting asset with search text Windows, community: Cloud
Computing Core Community
INFO: 79 results found.
INFO: Asset Microsoft Windows Server 2008 R2 (64-bit) (EHN) Image Id =
20012054 {16AD9067-E16E-6C83-A2E3-80512F3AD815}
INFO: Asset Microsoft Windows Server 2008 R2 (64-bit) Image Id =
20012053 {6D6FF958-0EBA-7A7E-CEC8-BE737E9ACA94}
...
```

Notice that the number of results returned is now 79, which is a subset of the total before: 93 total.

Business Scenario: Using Elastic Cloud Services to Scale

Let's continue developing our application from the business scenario in Chapter 2. The main advantages of an IaaS cloud for IoT Data are twofold: (1) The cloud hosts the application to save hosting costs and (2) the cloud APIs add servers and storage volumes to elastically increase storage capacity.

Fundamentally, the IoT Data application needs to allow customers to register new devices. You can add devices using the JPA code described earlier. However, because devices might have limited network bandwidth availability, we prefer to have file storage locations geographically close to the locations of the devices to optimize networking performance. We need to query the IBM SmartCloud Enterprise to find these locations. Let's build an HTML form for a customer to register a new device. This is the `doGet` method of the `NewDeviceServlet` class:

```
protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
      PrintWriter writer = response.getWriter();
      String customerIdStr = request.getParameter("customerid");
      long customerId = Long.parseLong(customerIdStr);
      writeTitle(writer, customerId, "Add Device");
      writer.println("<form method='POST' action='#'>");
```

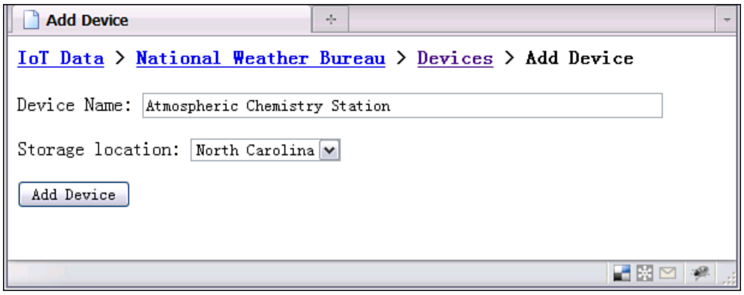
```
        writer.println("<p>Device Name: <input type='text'
name='device_name' size='60'/></p>");
        writer.println("<p>Storage location: <select name='location'>");
        Collection<Location> locations =
LocationService.getInstance().getLocations();
        for (Location location : locations) {
              writer.println("<option value='" + location.getId() +
                    "'>" + location.getName() + "</option>");
        }
        writer.println("</select></p>");
        writer.println("<p><input type='hidden' name='customerid'
value=" + customerId + "/></p>");
        writer.println("<p><input type='submit' value='Add
Device'/></p>");
        writer.println("</form>");
        writeFooter(writer);
}
```

The servlet expects the `customerid` parameter to contain the customer ID. It uses the `LocationService` to discover the different locations that the cloud supports. This populates the select element in the HTML form. The `LocationService` class is shown here:

```
package com.ibm.cloud.examples.iotdata.iaas;

import java.util.*;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.Location;

public class LocationService {

      private static LocationService instance = new LocationService();
      private Map<String, Location> locationMap =
            new HashMap<String, Location>();

      private LocationService() {
            load();
      }

      public static LocationService getInstance() {
            return instance;
      }

      public void load() {
            DeveloperCloudClient client =
                  ClientHolder.getInstance().getClient();
```

```
                try {
                        List<Location> locations =
        client.describeLocations();
                        for (Location location : locations) {

        LocationLabels.getInstance().replaceLabel(location);
                                locationMap.put(location.getId(), location);
                        }
                } catch (Exception e) {
                        System.err.println("Could not load locations: " +
                         e.getMessage());
                        e.printStackTrace();
                }
        }

        public Location getLocation(String id) {
                return locationMap.get(id);
        }

        public Collection<Location> getLocations() {
                return locationMap.values();
        }
}
```

This is a service to find and cache location objects from the cloud REST API. The locations are not expected to change frequently, so the service caches them for better performance. Also, because the names of the locations are not user friendly, the service adds user-friendly labels using the `LocationLabels` utility. The servlet writes out an HTML form that looks like Figure 3.15.



Device Name: Atmospheric Chemistry Station

Storage location: North Carolina

Figure 3.15 Business scenario screen to add a new device (NewDeviceServlet)

The locations are shown in the `select` element in the figure. You need to provide code to process the user's request to added the device when he or she clicks the Add Device button. That is also in the `NewDeviceServlet` class. It is in the `doPost` method, shown here:

```
protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
      String customerIdStr = request.getParameter("customerid");
      long customerId = Long.parseLong(customerIdStr);
      Customer customer = new Customer();
      customer.setId(customerId);
      String deviceName = request.getParameter("device_name");
      String location = request.getParameter("location");
      Device device = new Device();
      device.setName(deviceName);
      device.setLocation(location);
      device.setCustomer(customer);
      DeviceService deviceService = new DeviceService();
      PrintWriter writer = response.getWriter();
      writeTitle(writer, customerId, "Add Device");
      try {
            deviceService.addDevice(device);
            writer.println("<p>The device " + deviceName +
            " was successfully added</p>");
            writer.println("<p><a href='NewDeviceServlet?customerid="
+
            customer.getId() + "'>Add Another Device</a></p>");
      } catch(Throwable t) {
            writer.println("<p>The device could not be added.  Error:
" +
                        t.getMessage() + "</p>");
            writer.println("<p>Device Name: " + device.getName() +
"</p>");
            writer.println("<p>Location: " + device.getLocation() +
"</p>");
            writer.println("<p>Customer ID: " +
            device.getCustomer().getId() + "</p>");
            t.printStackTrace();
      }
      writeFooter(writer);
}
```

The code collects the customer ID, device name, and location from the servlet request. It uses these to add the device using the `DeviceService` class, which uses JPA to add the data to DB2. A message is displayed that depends on the success of the record insert operation.

Chapter 4

Standards

The goal of this chapter is to outline the standards that relate to IaaS cloud management and comment on how these can be leveraged in creating cloud applications.

Data Exchange

This chapter looks at several data exchange languages and tools that are important in cloud computing, particularly XML and JSON. Most applications built on top of IaaS clouds need both XML and JSON. This chapter helps users gain a basic background in the topic without having to go to external sources to understand the content in the book.

Extensible Markup Language (XML)

XML has been around for a long time, but a few points about it are worth discussing in relation to clouds. The first problem a programmer encounters with XML is how to parse it. The Java API for XML Binding (JAXB) API makes parsing XML documents simple. It binds Java objects to XML elements in a preprocessing step, an approach that saves a lot of development effort compared to a hand-crafted parser using the Document Object Model (DOM) or Simple API for XML (SAX). Unfortunately, not all programming languages have a utility like this.

The IBM SmartCloud Enterprise uses XML files called `parameters.xml` as a template to indicate which parameters the user should be prompted to enter when creating an image. The file is located in the RAM catalog entry for the image it is associated with, and the `GET` offerings/image REST API gives a URL for it. The equivalent Java API is `Image.getManifest()`. The following example represents the template of an image for an install of IBM DB2.

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="platform:/resource/com.ibm.ccl.devcloud
.client/schema/parameters.xsd">
      <field name="db2_user_password" type="password" label="Instance
owner (db2inst1) password"
            pattern="^\w*(?=\w*\d)(?=\w*[a-z])(?=\w*[A-Z])\w*$"
    patternErrorMessage="Invalid db2inst1 password. Must contain least
..."
            description="db2inst1 password. Must contain ...">
    <values>
          <value>db2cloud</value>
    </values>
</field>
...
</parameters>
```

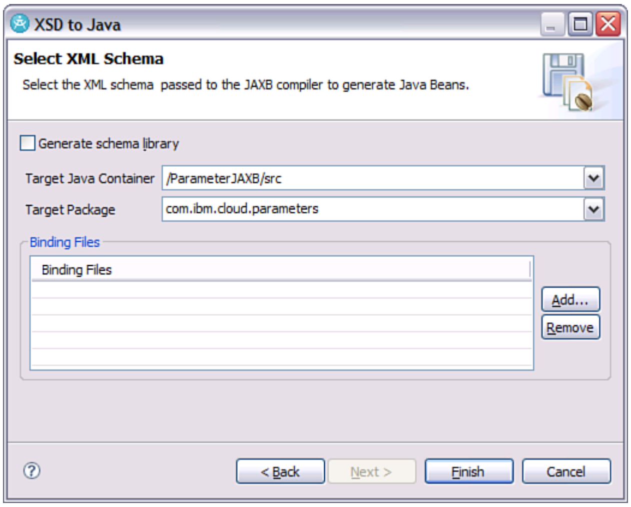
You might need to parse the `parameters.xml` file in two scenarios: when provisioning an instance in a cloud-management application and when customizing the behavior of an instance from inside the instance itself. In the first case, the `parameters.xml` file tells you how to prompt the user for parameters that are passed in at the time the provisioning request is submitted (for example, for values of the different DB2 passwords listed earlier). This was discussed in the section on the IBM SmartCloud Enterprise REST API in Chapter 3, "Developing with IBM SmartCloud Enterprise APIs." Your program must discover the name, description, type, and validation rules of the field to properly prompt the user to enter the data.

Now consider how DB2 will find the values of the passwords to set. The IBM SmartCloud Enterprise platform populates the `parameters.xml` file with the values submitted by the user and places them in the instance in the `/etc` directory. In this case, your program must discover the values of the fields.

The JAXB xjc tool binds XML elements to Java objects using the W3C XML schema language. The command takes a package name and a schema file as input, as shown here:

```
xjc -p <packageName> parameters.xsd
```

Alternatively, you can use the IBM Rational Software Architect (RSA) Wizard for this. Add the schema file (`parameters.xsd`) to your project and right-click. Select Generate, Java. A wizard displays, as in Figure 4.1.


XSD to Java

Select XML Schema

Select the XML schema passed to the JAXB compiler to generate Java Beans.

☐ Generate schema library

Target Java Container: /ParameterJAXB/src

Target Package: com.ibm.cloud.parameters

Figure 4.1 Rational Software Architect XSD to Java Wizard

Enter the package name and click the Finish button. This simple program makes use of JAXB to parse the DB2 `parameters.xml` file:

```
package com.ibm.cloud.api.parser;

import java.net.URL;
import java.util.List;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;

import com.ibm.cloud.api.parser.parameters.Field;
import com.ibm.cloud.api.parser.parameters.Parameters;

public class ParameterParser {

      public static void main(String[] args) throws Exception {
            URL url =
ClassLoader.getSystemResource("data/parameters.xml");
            JAXBContext jc =
JAXBContext.newInstance("com.ibm.cloud.api.parser.parameters");
            Unmarshaller u = jc.createUnmarshaller();
            Parameters parameters = (Parameters) u.unmarshal(url);
```

```
                //Display the parameters
                for (Field field : parameters.getField()) {
                     System.out.println("Field: " + field.getName());
                     System.out.println("\tLabel: " + field.getLabel());
                     System.out.println("\tDescription: " +
field.getDescription());
                     System.out.println("\tType: " +
field.getType().name());
                     System.out.println("\tNo. values: " +
field.getValues().getValue().size());
                     if (field.getValues().getValue().size() > 0) {
                           List<String> values =
field.getValues().getValue();
                           for (String value: values) {
                                 System.out.println("\t\t" + value);
                           }
                     }
                }
          }
}
```

The program assumes that the `parameters.xml` file is stored in the location /`data` relative to the classpath of the class `ParameterParser`. It creates a `JAXBContext`, which it uses to unmarshal the file. Then the program iterates over the fields, printing the name, label, description, type, and values. The output of the program appears next:

```
Field: db2_user_password
      Label: Instance owner (db2inst1) password
      Description: db2inst1 password. Must contain at least 1 number,
at least 1 lower case letter, and at least 1 upper case letter.
      Type: PASSWORD
      No. values: 1
            db2cloud
Field: db2_fenc_password
      Label: Fenced user (db2fenc1) password
...
```

See [Vernier, 2011, “Convert IBM Cloud Image Parameters into Java Using JAXB”] for more details on downloading and parsing `parameters.xml`.

JavaScript Object Notation (JSON)

The bodies of REST requests and responses might be in different formats, depending on the choices of the service implementers. In IBM SmartCloud Enterprise, the responses can be returned in either XML or JSON format. JSON might be new to some users, so we cover it briefly. JSON can be simpler and more lightweight than XML, but it lacks some of the tight definition of types that XML has. Another advantage of JSON is that it can be instantiated immediately by JavaScript in a browser. Although not a formal standard, JSON is becoming a de facto standard for REST services. JSON is described by IETF Request for Comments 4627, “The application/json Media Type for JavaScript Object Notation (JSON).” However, this document is informational, not a formal standard.

Several packages exist for streaming objects to and parsing JSON, including the IBM JSON4J library packaged with WebSphere Application Server 7.0, the json.org library, and the Google API. Following is an example of JSON from the IBM SmartCloud Enterprise:

```
{"images":[
      {      "name":"IBM DB2 Enterprise Developer Ed SLES 9.7.1 -
BYOL",
            "id":"20009984"
            "state":1,
            "visibility":"PUBLIC"
      }]
}
```

This example describes an image in the catalog. We have deleted some of the attributes to keep it short and simple. An example program that uses the json.org library to traverse this JSON data is shown next:

```
package com.ibm.cloud.examples.rest;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * Class to demonstrate parsing of JSON data.
 */
public class JSONTraverser {

      private static String DATA = "{'images':[" +
                "{'name':'IBM DB2 Enterprise Developer Ed SLES 9.7.1 -
BYOL'," +
              "'id':'20009984'," +
              "'state':1," +
              "'visibility':'PUBLIC'" +
            "}]}";

      public void parse() throws JSONException {
            JSONObject object = new JSONObject(DATA);
            JSONArray images = object.getJSONArray("images");
            for (int i=0; i<images.length(); i++) {
                  System.out.println("JSON array");
                  JSONObject image = (JSONObject)images.get(i);
                  String name = image.getString("name");
                  String id = image.getString("id");
                  String state = image.getString("state");
                  String visibility = image.getString("visibility");
                  System.out.println("Name: " + name);
                  System.out.println("ID: " + id);
                  System.out.println("State: " + state);
                  System.out.println("Visibility: " + visibility);
            }
      }

      /**
       * Entry point
       * @param args Not used
       */
      public static void main(String[] args) throws Exception {
            System.out.println("Parsing JSON string");
            JSONTraverser parser = new JSONTraverser();
            parser.parse();
      }
}
```

The program can be invoked from the command line. It creates a `JSONObject` from the data, from which it extracts a `JSONArray`. The program then iterates over the array, instantiating a `JSONObject` representing an image for each element in the array. The output of the program appears here:

```
Parsing JSON string
JSON array
Name: IBM DB2 Enterprise Developer Ed SLES 9.7.1 - BYOL
ID: 20009984
State: 1
Visibility: PUBLIC
```

This code is fine if you have a very simple task to perform. However, if you need to do something more substantial, you might need a more generic library, such as Google gson.

REST

REST APIs are provided as an important part of cloud computing platforms, both IaaS clouds and other types of clouds. However, REST is not specific to cloud computing, and you can make use of cloud computing without it. This section is provided to give readers basic background material on the topic so that they do not have to consult external sources to understand the examples in the book. Most of the material in this section is not specific to IBM SmartCloud Enterprise. The text also highlights areas that are particularly relevant to cloud computing.

Background

REpresentational State Transfer (REST) is a programming style modeled on the principles of the World Wide Web. Ron Fielding introduced it with his PhD thesis “Architectural Styles and the Design of Network Based Software Architectures,” with the goal of replicating the scalability and success of the web for application programming. REST architecture is defined as having the following characteristics:

- Addressable resources. The key entity is a resource, and it must be addressable with a uniform resource identifier (URI).
- A set of well-defined interfaces for managing the resources.
- Representation-oriented nature (say, with XML, JSON, or HTML).
- Stateless communication capability.

REST is a simpler paradigm than other kinds of distributed programming models that did not enjoy widespread adoption, such as SOAP and COBRA. The simplicity of the REST programming model is the main reason for its widespread popularity today.

HyperText Transfer Protocol

We mostly refer to REST as implemented using HTTP, although that is not necessarily the case. Other web services protocols, such as SOAP, use HTTP only as a transport, but REST uses the methods of HTTP, including `GET`, `POST`, `PUT`, and `DELETE`. We could look at flow of text to an HTTP server with a program like this one that connects with a TCP socket:

```
import java.io.*;
import java.net.Socket;

public class ReadHTTP {

      public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 80);
        PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out.print("GET /index.html\r\n\r\n");
            String line;
            while ((line = in.readLine()) != null) {
                  System.out.println(line);
            }
            out.close();
            in.close();
            socket.close();
      }

}
```

The program is simplified to demonstrate the principle, but it ignores some best practices of catching exceptions and closing streams in `finally` blocks. The program connects to the server `localhost` at HTTP port 80 and writes the text `GET /index.html` over the socket. This is a `GET` request. The request ends with `\r\n\r\n` so that the server gets the blank line that it needs to end the HTTP request. Then it reads the response back from the server and prints it to standard output. Executing the program returns something like this:

```
HTTP/1.0 200 OK
Date: Fri, 11 Feb 2011 00:18:23 GMT
Expires: -1
Cache-Control: max-age=0
Content-Type: text/html; charset=ISO-8859-1

<html><head>...
```

The HTTP response code is 200, which indicates that the response succeeded. A series of HTTP headers exist: `Date`, `Expires`, `Cache-control`, and `Content-type`. After the HTTP headers are printed comes a blank line, and then the HTML body is written out. The HTTP protocol is described in RFC 2616 [Fielding and Gettys, 1999], which includes the different request types, the HTTP headers, and the response codes.

REST Architecture

The idea that every resource must be addressable via a URI is a big help in simplifying programming models. Other programming models have often forced client programs to get objects for objects for objects before finally getting to the object that the client programmer is really interested in.

The REST principle of a constrained interface is difficult for object-oriented programmers to get used to. Only a handful of methods are available for all the different operations you need to provide. The methods include the following:

- `GET`—A read-only operation that requests specific information.
- `PUT`—Stores the body of the message to the server. This is usually an insert or update. The operation is idempotent, which means that, no matter how many times you `PUT` an object, the result is the same. Just as when clicking the Save button of a desktop application, the file will be saved the same no matter how many times you click the button.
- `DELETE`—Removes a resource.
- `POST`—Modifies the resources in some way. This is the only HTTP operation that is not idempotent. `POST` can also be used to create resources. It is the most open and extensible of the HTTP methods.
- `OPTIONS`—Enables querying of the server for capabilities.

HTTP uses the `Content-Type` header to tell the server or client what type of data is encoded in the body. This is a MIME type. Here is the format:

```
Content-Type: type/subtype; name=value; name=value ...
```

Consider these examples:

```
text/plain
application/xml;charset=utf-8
```

Using the `Accept` header, clients can tell the server what type of format they need: XML, JSON, or other. These headers can also be used for versioning.

The principle of statelessness helps applications scale and simplifies server implementations as well. It is also natural. For example, a client makes a call, uses the returned data to make another call, and so on to compose an application.

Hypermedia as the engine of application state (HATEOAS) is an additional REST principle promoted by some proponents. The interpretation is that we should use full URLs when referring to resources within responses, not just an identifier. The HATEOAS engine of application state principle means that it is much easier to navigate from resource to resource using links than it is to know exactly how to construct the addresses for each resource in advance. For example, suppose that a search result returned too many results to manage in one request. The server could return the first batch with a link to the next batch. This works just like searching with a browser at popular search portals.

Implementing and Consuming REST Services

Many tools help developers implement REST in their applications. At least one of these is a standard: JAX-RS is a set of Java APIs for development of REST services. It is defined by Java Specification Request 311, “Java API for RESTful Web Services.” Earlier examples used PHP with the cURL library to invoke REST web services. Let’s look at some other ways of producing and consuming REST services.

Business Scenario: File Uploads

One of the most fundamental capabilities of IoT Data’s cloud service is enabling file uploads from devices. This section looks at how to do that. The Internet Engineering Task Force RFC 1867 Form-based File Upload in HTML defines a `file` value for the `type` attribute of the HTML input element. It also defines a `multipart/form-data` value for the MIME media type. Example HTML code is shown here:

```
<form enctype='multipart/form-data' action='DeviceServlet'
    method='post'>
  <fieldset>
    <label for='devicefile'>File</label>
    <input id='devicefile' 'name='devicefile' type='file'/>
```

```
        <input type='submit' value='Upload File'/>
      </fieldset>
    </form>
```

This HTML fragment generates a file upload widget, which looks like Figure 4.2.

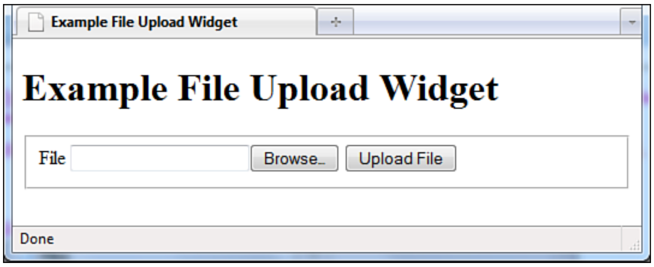


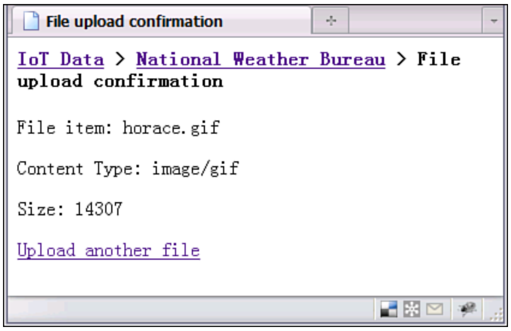
Figure 4.2 Business scenario: file upload

The Apache FileUpload project provides a utility for processing file uploads. The project depends on the Apache Commons IO library. Make sure that you have both JAR files in your classpath (`WebContent/WEB-INF/lib`). You can use this within the `doPost` method of the same servlet, as shown here:

```
protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
      PrintWriter writer = response.getWriter();
      Device device = getDevice(request);
      Customer customer = device.getCustomer();
      writeTitle(writer, customer, "File upload confirmation");
      DiskFileItemFactory factory = new DiskFileItemFactory();
      ServletFileUpload upload = new ServletFileUpload(factory);
      factory.setSizeThreshold(2097152);  // 2 MB
      try {
            List items = upload.parseRequest(request);
            for (Object item : items) {
                  FileItem fileItem = (FileItem)item;
                  writer.println("<p>File item: " + fileItem.getName()
+
                  "</p>");
                  writer.println("<p>Content Type: " +
                  fileItem.getContentType() + "</p>");
                  writer.println("<p>Size: " + fileItem.getSize() +
"</p>");
```

```
                  File uploadedFile = new
File("/opt/IBM/WebSphere/files/" +
                  fileItem.getName());
                  fileItem.write(uploadedFile);
                  }
      } catch (Exception e) {
            writer.println("Could not upload file: " +
e.getMessage());
            e.printStackTrace();
      }
      writer.println("<p><a href='DeviceServlet?deviceid=" +
            device.getId() + "'>Upload another file</a></p>");
      writeFooter(writer);
}
```

The method first creates a `DiskFileItemFactory` object, which is used to create a `ServletFileUpload` object. A limit of 2,097,152 bytes is set for the uploaded files. The `ServletFileUpload` object parses the file uploads and puts references (handles) to them in the `FileItem` objects. These are written out to the local file system under `/opt/IBM/WebSphere/files/`. Figure 4.3 shows the confirmation.



IoT Data > National Weather Bureau > File upload confirmation

File item: horace.gif

Content Type: image/gif

Size: 14307

Upload another file

Figure 4.3 File upload confirmation

Writing an equivalent REST service is similar. One change to make, however, is that the URL should use a REST-style pattern line:

```
http://host_name/devices/234
```

To use a style like this, you can add a mapping in the `web.xml` file as shown:

```
<servlet-mapping>
      <servlet-name>RESTServlet</servlet-name>
      <url-pattern>/devices</url-pattern>
      <url-pattern>/devices/*</url-pattern>
</servlet-mapping>
```

The `HttpServletRequest.getPathInfo()` method returns something like `/234`, from which you can find the device record. Besides this difference, the server implementation is exactly the same as for a browser client, although you can ignore the HTML returned. You can invoke the REST API with a client, as shown in the `uploadFile` method:

```
package com.ibm.cloud.examples.iotdata.rest.client;

import java.io.File;

import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.multipart.*;

public class FileUploadClient {

      private static final String BASE_URL =
             "http://host:9080/CloudAPIClientWeb/devices/";

      public void uploadFile(int deviceId, String fileName) {
            System.out.println("Uploading file: " + fileName);
            HttpClient httpclient = new HttpClient();
            HttpState state = new HttpState();
            httpclient.setState(state);
            PostMethod post = new PostMethod(BASE_URL + deviceId);
            File file = new File(fileName);
            try {
                  Part[] parts = new Part[1];
                  parts[0] = new FilePart("devicefile", file);
                  MultipartRequestEntity requestEntity = new
                   MultipartRequestEntity(parts, post.getParams());
                  post.setRequestEntity(requestEntity);
                  int statusCode = httpclient.executeMethod(post);
                  System.out.println("Response code: " + statusCode);
            } catch (Exception e) {
                  System.err.println("Error executing HTTP request: "
+
                         e.getMessage());
                  e.printStackTrace();
            } finally {
                  post.releaseConnection();
            }
      }
```

```
        public static void main(String[] args) {
              FileUploadClient client = new FileUploadClient();
              client.uploadFile(1,
"d:/temp/air_quality_sanfernando.gif");
        }
}
```

This program uses the Apache HTTPClient 3.1 library. The critical part of the program is the use of the `MultipartRequestEntity` class to add a `FilePart` to the request body. If everything goes well, you should see output similar to this:

```
Uploading file: d:/temp/air_quality_sanfernando.gif
Response code: 200
```

Response code `200` indicates that the request was successful.

Example: Uploading Files When Creating Instances with REST

In the IBM SmartCloud Enterprise "Java API" section in Chapter 3, you saw how to upload files using the Java API. Now that you understand more about the HTTP protocol and Apache HttpClient library, you can use the same tools to upload files when creating instances using the REST API. The following code demonstrates how to do this.

```
public int createInstance() throws HttpException, IOException {
      System.out.println("Creating instance");
      HttpClient httpclient = getHttpClient();
      PostMethod method = getPostMethod(baseUrl + INSTANCES);
      try {
            ArrayList<Part> parts = new ArrayList<Part>();
            parts.add(new StringPart("name", "My Instance"));
            parts.add(new StringPart("imageID", "20017833"));
            parts.add(new StringPart("instanceType",
"BRZ32.1/2048/60*175"));
            parts.add(new StringPart("publicKey", "mykey"));
            parts.add(new StringPart("location", "41"));
            parts.add(new StringPart("key", "license.key"));
            File file = new File("d:/temp/license.key");
            parts.add(new FilePart("key", file));
            Part[] array = (Part[]) parts.toArray(new
Part[parts.size()]);
            MultipartRequestEntity requestEntity = new
                   MultipartRequestEntity(array, method.getParams());
            method.setRequestEntity(requestEntity);
            int statusCode = httpclient.executeMethod(method);
            printBody(method);
```

```
                System.out.println("Status: " + statusCode);
                return statusCode;
            } finally {
                method.releaseConnection();
            }
        }
```

The method uploads the file `license.key`, which acts as the parameter for the same `parameters.xml` in the earlier Java example. The utility method `getHttpClient` is used to create an `HttpClient` object with the authentication parameters set. This is shown here:

```
        private HttpClient getHttpClient() {
            HttpClient httpclient = new HttpClient();
            HttpState state = new HttpState();
            UsernamePasswordCredentials creds = new
                UsernamePasswordCredentials(userName, password);
            state.setCredentials(AuthScope.ANY, creds);
            httpclient.setState(state);
            httpclient.getParams().setAuthenticationPreemptive(true);
            return httpclient;
        }
```

Other programs can reuse this method. You also use the utility method, shown here:

```
        private PostMethod getPostMethod(String url) {
            PostMethod method = new PostMethod(url);
            if (!xml) {
                method.setRequestHeader("Accept", "application/json");
            } else {
                method.setRequestHeader("Accept", "gzip,deflate");
            }
            return method;
        }
```

This method sets the return type using the `Accept` HTTP header with a global flag. The `printBody` method opens an input stream to read and print the response.

```
        private void printBody(HttpMethod method) throws IOException {
            BufferedReader br = null;
            try{
                br = new BufferedReader(new
                    InputStreamReader(method.
        getResponseBodyAsStream()));
                String readLine;
                while(((readLine = br.readLine()) != null)) {
                    System.out.println(readLine);
                }
            } finally {
                if (br != null) {
                    try { br.close(); } catch (Exception fe) {}
                }
            }
        }
```

As in the Java example, the file is uploaded to a location specified in `/etc/cloud/parameters.xml` in the instance, which is at `/home/idcuser/cloud/license.key`.

JAX-RS

JAX-RS is a set of Java APIs for the development of REST services. You might be interested in it for creating your own services built on an IaaS cloud and also for creating a client that communicates with a cloud IaaS-management API. It is defined by Java Specification Request 311, "Java API for RESTful Web Services." The reference implementation of the specification is provided by the Jersey open source project, which is a part of Glassfish. The dependencies for Jersey include the Java 6 SE JDK. JAX-RS enables you to create REST services based on Plain Old Java Objects (POJOs). JAX-RS can be used either to implement or to consume REST services.

Apache Wink is an open source implementation of JAX-RS. In addition to the open source download, it is bundled in WebSphere to enable easy development of REST web services in WebSphere. Several components are available in Wink:

- Service implementation building blocks
- Client building blocks
- Wink runtime

The service implementation building blocks include the following:

- Resource (enables retrieval and manipulation of data; a resource is bound to a URI using the `@Path` annotation).
- Provider (implements one or more interfaces in the JAX-RS specification, annotated with `@Provider`). These varieties of Provider exist:
  - Entity provider (serves data in a specific format or media type)
  - Context provider (provides information on how to serialize or deserialize with JAXB)
  - Exception mapping provider (handles exceptions that may occur during request processing)
- URI dispatching.
- Assets (contain business logic implemented by a developer).
- Annotations.
- URL handling.
- HTTP methods.
- Query parameters.

JAX-RS is a server API. Apache Wink can also help you create Java REST clients. The Apache Wink client transforms REST calls and results into Java classes that be used by a program consuming a REST service. It uses the `java.net.HttpURLConnection` class to connect to the REST service, and it provides the capability to serialize and deserialize REST requests and responses. It also provides Java object models for object models for Atom, JSON, XML, and other types. It supports HTTP proxies and SSL/TLS as well. The client building blocks include the following:

- RestClient (the central access point for the Apache Wink REST client)
- Resource (represents a single web resource)
- ClientRequest (represents a request invocation)
- ClientResponse (represents a response from a REST service)
- ClientConfig
- ClientHandler
- InputStreamAdapter
- OutputStreamAdapter
- EntityType

The Apache Wink runtime is deployed in a J2EE application server. It includes a servlet that accepts HTTP requests. It passes the requests to the Request Processor, which, in turn, passes control to the application code.

- Request Processor
- Deployment Configuration
- Handler Chains

The Apache Wink distribution includes three main JAR files:

```
wink-common-<version>.jar

wink-common-<version>.jar

wink-common-<version>.jar
```

A bundle also includes all the class files together. You need the JAX-RS JAR file `jsr311-api-1.1.1.jar` in the `/lib` directory, along with several other JAR files to support Wink (listed shortly).

A JAX-RS application is packaged as a web application in a WAR file. Packaging and deploying a JAX-RS application depends on whether the application server is JAX-RS aware. IBM WebSphere is JAX-RS aware with the installation of the Mobile/Web 2.0 Feature Pack. However, we assume that your application server is not JAX-RS aware. Follow these steps when deploying a JAX-RS application with Wink:

1. Create a root resource. Create a Java class annotated with the `@Path` tag and add Java methods for each of the HTTP methods that you need to support. Annotate these methods with the `@GET`, `@POST`, `@PUT`, and `@DELETE` tags.
2. Create a class deriving from `javax.ws.rs.core.Application`. This registers your resource in step 1 with the Wink framework. (You do not need this step in a JAX-RS-aware application server.)
3. Create a `web.xml` file to register `org.apache.wink.server.internal.servlet.RestServlet`, which is the entry point for Wink.
4. Package the files into a web container, including the JAR files for `commons-lang-2.3.jar`, `jaxb-api-2.1.jar`, `jaxb-impl-2.1.4.jar`, `stax-api-1.0-2.jar`, `jsr311-api-1.1.jar`, `slf4j-api-1.5.11.jar`, and `slf4j-jdk-1.5.11.jar`.

To create a REST client with Apache Wink, follow these basic steps:

1. Instantiate a `RestClient` object.
2. Instantiate a `Resource` object for the URL of the REST service, matching the resource.
3. Invoke the HTTP method.

The `ClientConfig` configuration class enables you to set security options, such as basic authentication with the `BasicAuthSecurityHandler` class.

Business Scenario: File Listing and Retrieval REST Service

Using what you have learned about JAX-RS and Apache Wink, you can create a file listing and retrieval REST service. The goal is to enable customers to browse and download files that they have uploaded to the IoTData cloud service. We show how to take the first steps in this direction by creating a simple REST service to browse and retrieve files on the local file system.

In RSA, create a Dynamic Web Project, named `iotdata` with context root `iotdata`. Also add a new EAR application project, named `iotdataEAR`. Add the Apache Wink JARs listed previously to the directory `WebContent\WEB-INF\lib`. Create a new class called `IotFile` to represent file resources with the code shown here:

```
package com.iotdata.rest;

import java.io.File;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

@Path("/iotfile")
public class IotFile {
    private static final String BASE_DIRECTORY = "e:/temp";
```

```
        @GET
        @Produces("text/xml")
        public Response get() {
              File directory = new File(BASE_DIRECTORY);
              StringBuilder sb = new StringBuilder("<files>");
              String[] filesNames = directory.list();
              for (int i=0; i<filesNames.length; i++) {
                    sb.append("<file>" + filesNames[i] + "</file>");
              }
              sb.append("</files>");
              return Response.ok(sb.toString()).type("text/xml").
  build();
        }

  }
```

The class defines an `IotFile` resource with the basic path `/iotfile`. Using the JAX-RS `@Path` annotation, you define a Java method to return the result of an HTTP `GET` using the `@GET` annotation. The output is returned in `text/xml` format, as specified with the `@Produces` tag. In this method, you return the names of all the files in the base directory using the standard `java.io` library. Of course, in a real application, you would have to determine the identity of the customer and check the user's authorization; you might not use the `java.io` library in such a straightforward way.

Because you are working with a JAX-RS-unaware application server, you add a class to register your resource with the JAX-RS framework. Create the class `IoTApplication` with the following code:

```
  package com.iotdata.rest;

  import java.util.HashSet;
  import java.util.Set;

  import javax.ws.rs.core.Application;

  public class IoTApplication extends Application {

        @Override
        public Set<Class<?>> getClasses() {
              Set<Class<?>> classes = new HashSet<Class<?>>();
              classes.add(IotFile.class);
              return classes;
        }
  }
```

This code registers the `IotFile` class. Next, edit the `web.xml` file, adding the servlet definition, as shown.

```
<servlet>
  <servlet-name>IoTApp</servlet-name>
  <servlet-
class>org.apache.wink.server.internal.servlet.RestServlet</servlet-
class>
  <init-param>
    <param-name>javax.ws.rs.Application</param-name>
    <param-value>com.iotdata.rest.IoTApplication</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>IoTApp</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Deploy this to the application server. When you enter the URL

```
http://host:8080/iotdata/rest/iotfile
```

you should see something like the XML shown here:

```
<files>
  <file>timezone.png</file>
  <file>timezone2.png</file>
</files>
```

Of course, you will not see the same files we have in our temp directory. You can allow users to retrieve files with an additional Java method supporting HTTP `GET`, shown here:

```
@GET
@Path("{fileName}.{extension}")
public Response getFile(@PathParam("fileName") String fileName,
            @PathParam("extension") String extension) {
      File file = new File(BASE_DIRECTORY + '/' + fileName + '.' +
extension);
      if (!file.exists()) {
            return Response.status(Response.Status.NOT_FOUND).build();
      }
      return Response.ok(file).type(getMediaType(extension)).build();
}
```

The `@Path` annotation is now used at the method level. It is additive to the `@Path` annotation, used at the class level. If an HTTP method matching the pattern

```
iotfile/{fileName}.{extension}")
```

is received, the method `getFile` is invoked. The method checks to see whether the file exists. If it does not, a `NOT_FOUND` HTTP status is returned. Otherwise, the file is returned. You have a little work to do to determine the media type. Otherwise, the REST client might not know how to process the file returned. Add a private method to sniff the media type based on the file extension, as shown:

```
private MediaType getMediaType(String extension) {
      if ("png".equalsIgnoreCase(extension)) {
            return new MediaType("image", "png");
      } else if ("gif".equalsIgnoreCase(extension)) {
            return new MediaType("image", "gif");
      } // ... more media types
      return new MediaType();
}
```

This is a rudimentary implementation that enables you to experiment with the concept. The media type determines the value of the `Content-Type` HTTP header. The official list is the IANA MIME Media Types [IANA, 2001]. Now if you enter this URL

```
http://host:8080/iotdata/rest/iotfile/timezone.png
```

you should receive the file `timezone.png`. Substitute the name of a file on your own file system for this. You can try it in a browser.

We can demonstrate a simple client for this REST service using the Apache Wink client library. First create a new Java project. Add all the JAR files as for the server project but substitute the Wink client JAR for the server JAR used earlier. The following program consumes the REST service to list the files.

```
package com.iotdata.rest.client;

import org.apache.wink.client.Resource;
import org.apache.wink.client.RestClient;

/**
 * Demonstrates building a simple client with Apache Wink
 */
public class IoTDataRESTClient {

      public void listFiles() {
            RestClient client = new RestClient();
            Resource resource =
client.resource("http://host:8080/iotdata/rest/iotfile");
            String xml =
resource.accept("text/xml").get(String.class);
            System.out.println(xml);
      }
```

```
        public static void main(String[] args) {
              IoTDataRESTClient client = new IoTDataRESTClient();
              client.listFiles();
        }

  }
```

The class first instantiates the `RestClient` object. Then it instantiates a `Resource` object for the URL matching a file listing. At this point, the `Resource` object exists only on the client. To invoke the `GET` method, the Java `get` method is called with a class type of Java String. The output looks similar to this:

```
<files><file>timezone.png</file><file>timezone2.png</file></files>
```

There are several options for processing JSON formatted data. We demonstrate the `wink-json-provider` provider, which is based on the open source JSON.org library. A client can indicate that it needs to receive JSON format using the HTTP `Accept` header. The HTTP headers can be retrieved using the annotation `@Context` and by defining a private field class `IotFile`, as shown.

```
@Context
private HttpHeaders headers;
```

If the value is `APPLICATION_JSON`, you return `JSON`. Otherwise, you return `XML`. The following method replaces method `get` in class `IotFile`.

```
@GET
public Response getXmlOrJson() {
      // Read the files from the local directory
      File directory = new File(BASE_DIRECTORY);
      String[] filesNames = directory.list();
      // If JSON is supported by the client return JSON
      List<MediaType> acceptHeaders =
headers.getAcceptableMediaTypes();
      if (acceptHeaders != null) {
            for (MediaType mt : acceptHeaders) {
                  if
(MediaType.APPLICATION_JSON_TYPE.isCompatible(mt)) {
                        JSONArray files = new JSONArray();
                        for (int i=0; i<filesNames.length; i++) {
                              JSONObject file = new JSONObject();
                              try {
                                    file.put("name", filesNames[i]);
                              } catch (JSONException e) {
                                    e.printStackTrace();
                                    return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).build();
```

```
                                    }
                                    files.put(file);
                              }
                              return
          Response.ok(files.toString()).type("application/json").build();
                        }
                  }
            }
            // otherwise, return XML
            StringBuilder sb = new StringBuilder("<files>");
            for (int i=0; i<filesNames.length; i++) {
                  sb.append("<file>" + filesNames[i] + "</file>");
            }
            sb.append("</files>");
            return Response.ok(sb.toString()).type("text/xml").build();
      }
```

The method uses the JSON.org `JSONArray` and `JSONObject` classes to build the response object. Add the `wink-json-provider-1.1.3-incubating.jar` and `jsonxxx.jar` files to the web application classpath and refresh the web application. To try it, you can use the following client code:

```
      public void listFilesJSON() {
            RestClient client = new RestClient();
            Resource resource =
      client.resource("http://host:8080/iotdata/rest/iotfile");
            String xml =
      resource.accept("application/json").get(String.class);
            System.out.println(xml);
      }
```

This client accepts only an `application/json` MIME type. The result is shown here:

```
[{"name":"timezone.png"},{"name":"timezone2.png"}]
```

This indicates that two files are available, with the file names `timezone.png` and `timezone2.png`.

Virtualization

Virtualization standardization is important for the portability of virtual machine images between clouds. The Open Virtualization Format (OVF) standard is the most widely supported standard in this area and the only standard discussed in this book. It is used in the IBM SmartCloud Enterprise import and copy capabilities. These functions enable you to import a virtual machine image created in your own local virtual environment to the cloud, to copy an image from the cloud to your local environment, and to copy an image from one data center to another.

Open Virtualization Format

OVF is a standard from the Distributed Management Task Force for packaging virtual machines for storage and transport, enabling cross-platform portability. The current version is 1.1 (2010), and the standards group is actively developing the standard further. OVF supports both single virtual machine and multiple virtual machine configurations. It is neutral enough to support multiple operating systems and hypervisor platforms.

An OVF package includes these files:

- An OVF descriptor file with a `.ovf` extension
- An OVF manifest with a `.mf` extension (optional)
- An OVF certificate with a `.cert` extension (optional)
- Disk image files (optional)
- Additional resource files, such as ISO images (optional)

The OVF package can be tarred into a single package and named with an `.ova` extension. The system environment is specified in an XML descriptor containing deployment parameters, such as IP address, host name, subnet, and gateways. The OVF descriptor contains the following sections:

- References to external files
- List of disks
- Network information
- Virtual system information (vendor and application information)
- Virtual hardware, such as CPUs and memory
- Operating system information

The goal of OVF is to be a format for storage and distribution of virtual machines rather than execution. Hypervisor implementations mostly use different formats for execution of virtual machines. The Virtual Disk Format (VMDK) from VMWare, the Virtual Hard Disk (VHD) format from Microsoft, and the open source QCOW format are runtime formats. They are often frequently used for storing and distributing virtual machines, but they are not portable. Even though OVF is not a runtime format, it includes information for startup and shutdown.

Cloud Computing

A number of cloud standard initiatives presently exist. We briefly discuss the IBM Cloud Computing Reference Architecture submission to the Open Group, the Distributed Management Task Force Cloud Standard Incubator Group, and the Cloud Data Management Interface. Other emerging standards include the Open Cloud Computing Interface (OCCI) from the Open Grid Forum, but we do not have space to include them.

Cloud Computing Reference Architecture

The Cloud Computing Reference Architecture was developed within IBM. Its architecture overview was the basis for a submission from IBM to the Open Group outlining a blueprint and best practices for developing different models of cloud computing driven by functional and non-functional requirements. The Cloud Computing Reference Architecture is important to cloud providers and consumers because it proposes a common terminology, roles, an understanding of the basic structure of clouds, and design principles. This is critical for providing consistency and quality across cloud offerings. It outlines the different components in a cloud service to support various use cases for Infrastructure as a Service, Platform as a Service, Software as a Service, and Business Process as a Service, as well as for public, private, community, and hybrid cloud models.

The Cloud Computing Reference Architecture places special focus on several key architectural elements:

- Operational and business support, to enable automated delivery, management, and handling of all financial and contractual concerns
- Service layer, providing Infrastructure as a Service, Platform as a Service, Software as a Service, and other services
- Security, performance, resiliency, consumability, and governance as cross-cutting concerns relevant across all elements of a cloud
- Creation and consumption of cloud services

For more on the Cloud Computing Reference Architecture, see the document *Introduction and Architecture Overview: IBM Cloud Computing Reference Architecture 2.0* [Behrendt, 2011].

Distributed Management Task Force Open Cloud Standards Incubator

The Open Cloud Standards incubator group in the Distributed Management Task Force (DMTF) is active in formulating standards for clouds. The group includes representatives from many leading companies in the industry. The white paper "Architecture for Managing Clouds" describes the collective high-level view of cloud computing. The white paper "Use Cases and Interactions for Managing Clouds" [DMTF, 2010] describes the scope of the group. Use cases include the following:

- Establishing a relationship (a potential consumer of services establishing the organizational or individual identity for future use of cloud resources)
- Administering a relationship (for example, adding and removing users of the service)
- Establishing a service contract
- Updating a service contract
- Contract reporting
- Contract billing

- Terminating a service contract
- Provisioning resources
- Deploying a service template
- Changing resource capacity
- Monitoring service resources
- Creating a service template (this is creating an image, in the terminology used in this book)
- Creating a service offering (the service developer creates a new offering)
- Notification (for a variety of different events, such as security events)

Many of these use cases relate to business support systems discussed in Chapter 9, “Operations and Maintenance on the Cloud,” rather than the operational aspects of managing cloud resources.

In 2010, many organizations, including IBM, made submissions to the DMTF for a standard application programming interface for managing cloud resources (IaaS APIs). A draft standard was released in 2012, and we hope that it will be finalized and adopted by cloud providers to enable developers to port cloud applications between different clouds.

Cloud Data Management Interface

The Cloud Data Management Interface (CDMI) is a cloud storage standard from the Storage Network Industry Association. The standard refers to this type of service as Data Storage as a Service. The standards define how applications can create and manage storage elements in the cloud, in addition to discovering the storage capabilities of the cloud provider and managing associated administrative resources (including containers, accounts, security access, and monitoring and billing information). The standard is designed to be generic enough to support Internet file systems, cloud bucket type storage systems, block-based storage, and other types of storage systems. It enables you to build portable applications that rely on cloud storage services.

One of the challenges of a cloud storage API is being able to handle streaming and events when uploading and downloading large files over the Internet. CDMI handles this with queues.

Business Scenario: IoT Data Use of Standards

The Internet of Things (IoT) is a rapidly evolving area that undoubtedly will include many standards. Right now, however, the main question for IoT Data is, what standards should the enterprise expose to IoT Data customers, to allow them to most easily make use of the service? Many of the devices on the IoT are simple, so REST is a good choice: It is simple and neutral toward programming language and is also a natural extension of the IoT itself. With the addressability principle, each device can have its own web site:

```
http://example.com/devices/id
```

Devices can be browsed using this URL:

```
http://example.com/devices
```

Organizations themselves are entities and can be browsed with the following URL:

```
http://example.com/organizations/{org-id}
```

Organizations can elect to be listed and make their data available publicly. In any case, the URLs will be the same, and access control rules determine visibility.

The data itself can up uploaded, downloaded, and searched with the CDMI REST standard. IoT Data can create an implementation of this standing using JAX-RS. Beyond being a standard that makes use of a well-understood and interoperable programming model, it includes metadata representation for access controls, which is an important aspect of cloud storage.

CHAPTER 5

Open Source Projects

The goal of this chapter is to provide an outline of open source projects that are related to IaaS cloud management and comment on how these can be either used in creating cloud applications or extended further.

Virtualization Projects

We start with a small selection of virtualization projects because virtualization is the foundation of cloud computing. Table 5.1 summarizes the different virtualization projects.

Table 5.1 Summary of Open Source Virtualization Projects

Project	Description
KVM	Virtualization platform built into the Linux kernel
libvirt	Virtualization-management API written in C
Xen	Virtualization platform

The following sections discuss each project in more detail.

Kernel-Based Virtual Machine (KVM)

KVM was introduced in the "Virtualization" section in Chapter 1, "Infrastructure as a Service Cloud Concepts." It is an open source virtualization solution for x86 hardware implementing Intel VT or AMD-V virtualization extensions. KVM uses a kernel-based virtualization model and is now included in the Linux kernel. It also relies on some processor-specific extensions and leverages the QEMU open sources project. KVM can run Linux and Windows virtual machines. KVM is included by default in the Red Hat Enterprise Linux (RHEL) images on the IBM SmartCloud Enterprise.

You can tell whether your hardware supports the required extensions by looking in the `/proc/cpuinfo` file. Entries for either vmx (Intel) or svm (AMD) should appear in the file. An example for an Intel Core Duo chip is shown in bold:

```
processor     : 0
vendor_id     : GenuineIntel
...
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl
aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm
sse4_1 xsave lahf_lm ida dts
```

You can compare this to the CPU information from a virtual machine on IBM SmartCloud Enterprise, shown here:

```
processor        : 0
vendor_id        : GenuineIntel
cpu family       : 6
model            : 6
model name       : QEMU Virtual CPU version 0.9.1
...
flags            : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx lm up pni
```

Notice that no `vmx` flag exists for the virtual machine, so you cannot run a KVM hypervisor on it. Similarly, there is often no `vmx` CPU flag for virtual machines running on Xen.

KVM introduced a new mode of execution to the Linux kernel, in addition to kernel and user modes. The new mode, called **guest mode**, has its own user and kernel modes. Guest mode is not used unless the code executes I/O instructions, when it falls back to user mode. The KVM module added a new character device driver to expose virtual hardware to guest operating systems.

The kernel-based virtualization model allows more complete support of the Linux OS. KVM makes the hypervisor a part of the kernel, which reuses a huge body of Linux code for this purpose. This is a different approach than in Xen and VMWare, which have independent modules to do this.

Virtio is the main I/O virtualization platform for KVM, used to support block storage and network devices.

You can install KVM and the virtio libraries with the following commands:

```
# yum install kvm
# yum install virt-manager libvirt libvirt-python python-virtinst
```

You can create a virtual machine image using the QEMU commands, described next. The `kvm` command starts new virtual machines.

The Open Virtualization Alliance is a group of virtualization, data center, and cloud solution providers that promotes the awareness, understanding, and use of KVM.

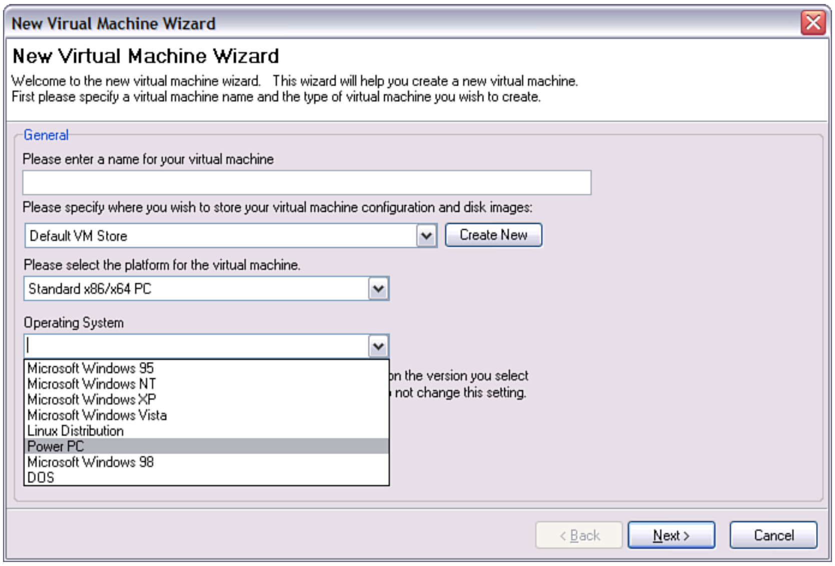
QEMU

QEMU is an open source machine emulator and virtualization tool. QEMU runs on Windows and Linux on x86 systems. It can emulate a number of CPU architectures, including MIPS, PowerPC, and SPARC. Xen and KVM rely on QEMU libraries when running in certain modes. QEMU by itself performs in emulation mode, which can be slow. For better performance, you can install kqemu, which allows it to run directly on the host processor. QEMU Manager is a graphical user interface front end to QEMU for Windows. The current version of QEMU is 0.15.

You can install QEMU on Red Hat or Fedora with this command:

```
# yum install qemu-launcher
```

Figure 5.1 shows the first page of the QEMU Manager New Virtual Machine Wizard.


New Virual Machine Wizard

New Virtual Machine Wizard

Welcome to the new virtual machine wizard. This wizard will help you create a new virtual machine.
First please specify a virtual machine name and the type of virtual machine you wish to create.

General

Please enter a name for your virtual machine

Please specify where you wish to store your virtual machine configuration and disk images: Default VM Store

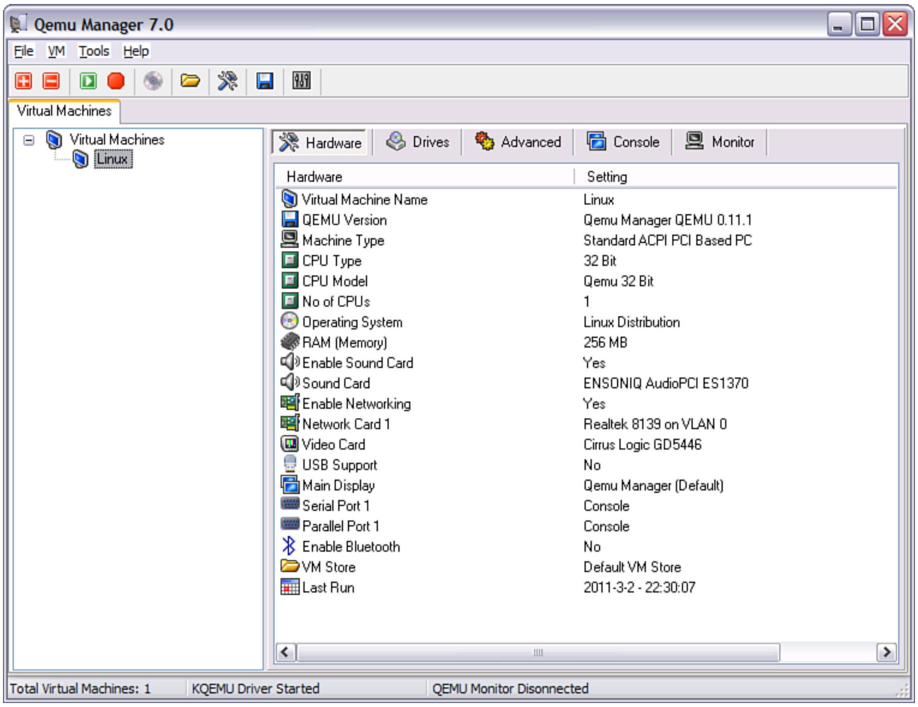
Please select the platform for the virtual machine. Standard x86/x64 PC

Operating System

on the version you select
not change this setting.

Figure 5.1 QEMU Manager New Virtual Machine Wizard

In this step, you select the VMStore, architecture, and operating system. In the next step, you set memory, virtual disk size, and image format. The supported formats are qcow (native QEMU format), raw, and VMDK (VMWare image format). After you create a new virtual machine, you can see it in the console, as shown in Figure 5.2.



Hardware	Setting
Virtual Machine Name	Linux
QEMU Version	Qemu Manager QEMU 0.11.1
Machine Type	Standard ACPI PCI Based PC
CPU Type	32 Bit
CPU Model	Qemu 32 Bit
No of CPUs	1
Operating System	Linux Distribution
RAM (Memory)	256 MB
Enable Sound Card	Yes
Sound Card	ENSONIQ AudioPCI ES1370
Enable Networking	Yes
Network Card 1	Realtek 8139 on VLAN 0
Video Card	Cirrus Logic GD5446
USB Support	No
Main Display	Qemu Manager (Default)
Serial Port 1	Console
Parallel Port 1	Console
Enable Bluetooth	No
VM Store	Default VM Store
Last Run	2011-3-2 - 22:30:07

Total Virtual Machines: 1 KQEMU Driver Started QEMU Monitor Disconnected

Figure 5.2 QEMU Manager console

You can download some QEMU images from the QEMU project web site to try it out. On Linux, you create an image file with this command:

```
$ qemu-img create -f qcow myimage.img 10G
```

This creates a 10 G file for an image with the QEMU qcow format. You can start a virtual machine with this command:

```
$ qemu-system-x86_64 -boot d -cdrom /images/myimage.iso -hda
myimage.img
```

This starts a virtual machine session that installs an operating system based on an ISO image mounted as a CD-ROM using the image file `myimage.img`. After the operating system is installed, you can run it with this command:

```
$ qemu-system-x86_64 myimage.img
```

libvirt

libvirt is a C library to invoke virtualization capabilities in Linux. It was originally designed as a management API for Xen, but it now supports KVM, QEMU, Xen, VMWare, Microsoft

Hyper-V, and other hypervisors. libvirt supports remote management of virtual machines, virtual networks, and storage using secure protocols. libvirt has bindings in other languages than C, including Python, Perl, Ruby, Java, and PHP. It also includes a command-line tool called the virtualization shell, or virsh. Graphical user interfaces, such as Virtual Machine Manager, use libvirt to manage virtualized resources. The current version of libvirt is 0.9.4-1.

libvirt can be used either locally or remotely. It uses a daemon called `libvirtd` to support remote communication.

The libvirt resource model centers on the following entities:

- **Node**—A single physical machine
- **Hypervisor**—A process that allows virtual machines to be run on the node
- **Domain**—An instance of an operating system running on a virtual machine
- **Snapshots**—An image of a virtual machine

There is an XML description of the different resources. The XML schema for domain includes the follow elements:

- BIOS bootloader
- Host bootloader
- System information
- CPU model and topology
- CPU allocation—virtual CPUs
- CPU tuning
- Memory allocation
- NUMA mode tuning
- Block I/O tuning
- Lifecycle control
- Hypervisor features
- Devices
- Filesystems
- Network interfaces
- Graphic framebuffers
- Video devices
- Consoles
- Security

The libvirt CPU concept is that a number of virtual CPUs (VCPUs) can be allocated to a virtual machine, and the VCPUs can optionally be pinned to one or more physical CPUs.

You can use the `virsh` command to launch virtual machines after creating a `domain.xml` file. This is the basic form of the `create` command:

```
$ virsh create domain_file.xml
```

To list active domains, use this command:

```
$ virsh list
```

A number of other commands enable you to save, restore, reboot, suspend, and restore virtual machines.

Xen

Xen is a open source virtualization solution for x86, IA_64, ARM, and other hardware. The project includes a hypervisor and related tools. Xen can run Linux, Windows, and other operating systems.

Xen originated at the University of Cambridge Computer Laboratory in 2001 as part of the XenoServer project. It was first described in an academic paper in 2003 at an Association of Computing Machinery (ACM) symposium. The company XenSource was founded in 2004 to promote commercial adoption. Citrix acquired XenSource in 2007.

With Xen, a special privileged domain called Domain0, or Dom0, is used to control the guest virtual machines, or guest domains, and to manage the hardware. The hypervisor gives each guest domain some of the physical system resources. It exports simplified devices to the guest domains. For example, if the physical network device is a Linksys or 3Com Ethernet card, the guest domain sees a simplified network device. Storage block devices are also exported to the guests as generic block devices.

To install Xen on SUSE 11, start YaST and choose Virtualization, Install Hypervisor and Tools. On the next screen, select Xen. Then select Networked Bridge Configuration and click Install. To make the version of the OS with Xen installed the default upon booting, in YaST, select System, Boot Loader and then reboot. You might experience issues with running a Xen-enabled guest on a cloud that is based on KVM, such as IBM SmartCloud, because of a conflict between the two virtualization systems.

In x86 are four protection rings that act as privilege levels. Xen makes use of these to isolate and manage different guest operating systems. Ring 0 is the most privileged level; ring 3 is the least privileged. The Xen hypervisor runs in ring 0, the guest operating system in ring 1, and the guest user-level applications in ring 3. The Intel and AMD processor virtualization extensions provide alternate ways to achieve the same purpose. Guest domains are called DomU, or unprivileged domains. With the Intel VT extensions present, the Xen hypervisor operates in a mode called Virtual Machine Extensions (VMX) root operation mode, and the guest operating system operates in VMX nonroot mode. Xen uses the Hardware Virtual Machine (HVM) interface to abstract the differences between Intel and AMD processors.

Cloud Projects

This section gives a brief overview of several open source cloud projects. Eucalyptus and OpenStack could be utilized to build private clouds. LibCloud and DeltaCloud could be utilized to build management utilities for cloud resources.

Eucalyptus

Eucalyptus is an open source cloud-management platform sponsored by Eucalyptus Systems, Inc. It provides Infrastructure as a Service (IaaS) based on Linux. Eucalyptus is hypervisor neutral and can currently run (as of version 2.0) on KVM or Xen hypervisors. Eucalyptus also interoperates with public clouds, presently supporting Amazon's EC2. This capability enables customers to support a private-public hybrid cloud model with a private cloud running Eucalyptus on their own premises and using public cloud resources on EC2. Eucalyptus even uses AWS-compatible APIs for access to its own resources and includes an S3-compatible bucket storage manager called Walrus. Using this service, objects are stored as data into buckets with user-defined keys.

You can try Eucalyptus online via a test drive hosted system.

Apache Libcloud

Apache Libcloud is a cloud-management API written in Python. It provides a uniform API that connects to multiple cloud providers, including the IBM SmartCloud Enterprise. Its goal is to provide a basic yet functional cloud library. Libcloud includes modules for managing virtual machines, storage, and load balancers. The current version is 0.5.2. It depends on Python 2 and will not run on Python 3.

To install Libcloud, download and unzip the ZIP bundle, change to the module directory, and type the following command:

```
$ python setup.py install
```

The `setup.py` script is included in the bundle.

In Libcloud terminology, a **node** is a virtual machine instance. The following script gives a list of your virtual machine instances.

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver

SCE_USERNAME= 'a.user@example.com'
SCE_PASSWORD = '***'

Driver = get_driver(Provider.IBM)
conn = Driver(SCE_USERNAME, SCE_PASSWORD)
nodes = conn.list_nodes()
print('No. of virtual machines found:', len(nodes))
```

In this example, the values for user name and password `SCE_USERNAME` and `SCE_PASSWORD` are intended to be stored in environment variables. The function `get_driver()` gets a driver that is appropriate for the cloud provider. The `Driver` class is initialized with a constructor that creates a connection to the cloud with the identity of the user. Finally, the method `list_nodes()` returns a list of virtual machine instances that are printed to the console. Classes for different cloud providers exist. When executing the script, you get a warning about the certificate not being checked; the site has instructions for fixing this. The output from the script is shown here:

```
('No. of virtual machines found:', 1)
```

The following method gives a list of images:

```
...
images = conn.list_images()
```

The first part of the script is identical to the first example.

Delta Cloud

Delta Cloud is a cloud-management API with a similar goal to that of Apache libCloud: to provide a standard interface to access multiple clouds. It is also an Apache incubator project. Delta Cloud provides a REST API that acts as a proxy to multiple cloud providers, including the IBM SmartCloud Enterprise. The project was initiated by Red Hat and is aligned with its REST API for enterprise virtualization. In addition to the REST API is a Ruby client and C library. The current version is 0.3.0.

The model for compute entities in the REST API includes the following resources:

- Realms
- Hardware profiles
- Images
- Instances
- Keys
- Firewalls
- Addresses
- Load balancers

A realm is a container for compute resources, like a data center or a pool of resources within a data center.

The model for storage entities includes the following resources:

- Storage volumes
- Storage snapshots
- Blob storage

OpenStack

OpenStack is an open source project that includes a full Infrastructure as a Service cloud platform for public and private clouds. Sponsored by Rackspace and NASA, it was launched in 2010. OpenStack leverages both the Rackspace and the NASA Nebula cloud computing platforms. It is released with an Apache license and was developed in Python. The OpenStack platform includes compute, storage, and image repository services.

The compute service includes infrastructure for managing a network of virtual machines and networks, including a control panel and API. Access is managed using role-based access control through users and projects. The compute module is hardware and hypervisor agnostic.

The storage service, called the OpenStack Object Store, or Swift, is a scalable redundant storage system. It is an object store, not a file system.

The image service provides discovery, registration, and loading of virtual machine images. It integrates with the OpenStack Object Store for storing the images. It supports a number of image formats, including Raw, AMI, Hyper-V, VDI, qcow2, VMDK, and OVF.

Cloud Foundry

Cloud Foundry is an open source project sponsored by VMWare that focuses on Platform as a Service. It supports multiple cloud providers and application services. The app execution engine runs applications in Rails, Java, and other languages. A request router accepts all HTTP requests to the application and routes them. Cloud Foundry includes an autoscale feature that distributes requests using a request router to multiple application execution engines. A cloud controller loads and unloads applications. A set of services provides functions such as data persistence that applications can leverage. A health monitor checks the health of the application and restarts it if necessary.

Hadoop

A distributed file system is one that can span many nodes. The Hadoop Distributed File System (HDFS) is an example of such a distribute file system used by many large web sites with huge storage requirements. Apache Hadoop is the open source project that supports HDFS. Hadoop also includes a wider platform that supports a MapReduce distributed programming framework. It consists of four parts:

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- MapReduce, a framework for distributed processing of large data sets
- ZooKeeper, coordination service

The prerequisites for Hadoop are Secure Shell (SSH) and the Secure Shell server (SSHD).

HDFS is a fault-tolerant, distributed file system designed to run on low-cost hardware. It allows streaming for large data sets, which typically are gigabytes or terabytes in size. HDFS is designed for batch processing with a write-once, read-many use pattern rather than interactive use. This fits applications such as web crawlers or map-reduce applications.

Hadoop supports a style of programming called map/reduce. In this style, a large task is broken down into small pieces, the pieces are computed individually, and the pieces finally are combined into a whole. A central application of this is combining data from the HDFS file system that is spread over many nodes in the cluster. Hadoop provides a fault-tolerant system, which is important because, with many nodes in a cluster, the chance of any one failing goes higher.

ZooKeeper is a service for maintaining configuration and naming information and for providing distributed synchronization.

Many large-scale business applications use Hadoop. For example, Yahoo! Search uses a 10,000-machine Hadoop cluster. According to the Hadoop web site, Rackspace, Amazon, and Facebook also use Hadoop.

Setting up Hadoop

The prerequisites for Hadoop are Java 6 SE and SSH. We run Hadoop 0.20.2 on Linux.

Hadoop can run in three modes:

- **Stand-alone**—Everything runs as one process, with no daemons
- **Pseudo-distributed**—Simulates a cluster with only one node
- **Distributed**—Fully clustered

Stand-alone and pseudo-distributed modes are suitable for development and testing. Distributed mode is necessary for production use. In stand-alone mode, the only configuration step is to set your `JAVA_HOME` environment variable. Let's set up Hadoop in pseudo-distributed mode. Download Hadoop and execute these commands as root:

```
# mkdir /etc/hadoop
# mv ~idcuser/Desktop/hadoop-0.20.2.tar.gz /etc/hadoop
# chown -R webadmin:users /etc/hadoop
```

This sets up `/etc/hadoop` as the install directory for Hadoop. Add the following lines to `~webadmin/.bashrc` as webadmin:

```
export HADOOP_INSTALL=/opt/hadoop/hadoop-0.20.2
export PATH=$PATH:$HADOOP_INSTALL/bin
```

Execute the following commands as webadmin:

```
> cd
> . ./.bashrc
```

Add the webadmin user to `/etc/ssh/sshd_config`:

```
AllowUsers idcuser webadmin
```

Restart the SSH service. Execute the following commands as webadmin to enable SSH login without a password:

```
> ssh-keygen -t rsa -P '' -f ~/.ssh/mykey
> cat ~/.ssh/mykey >> ~/.ssh/authorized_keys
```

Try it using the command `ssh local`. You should not have to enter a password. See the "Secure Shell (SSH)" section in Chapter 7, "Security," for more on SSH.

Edit the `core-site.xml` file in the directory `$HADOOP_INSTALL/conf`, as shown:

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://127.0.0.1/</value>
  </property>
</configuration>
```

Edit the `hdfs-site.xml` file as shown:

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Edit the `mapred-site.xml` file:

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>127.0.0.1:8021</value>
  </property>
</configuration>
```

Format the Hadoop file system using this command:

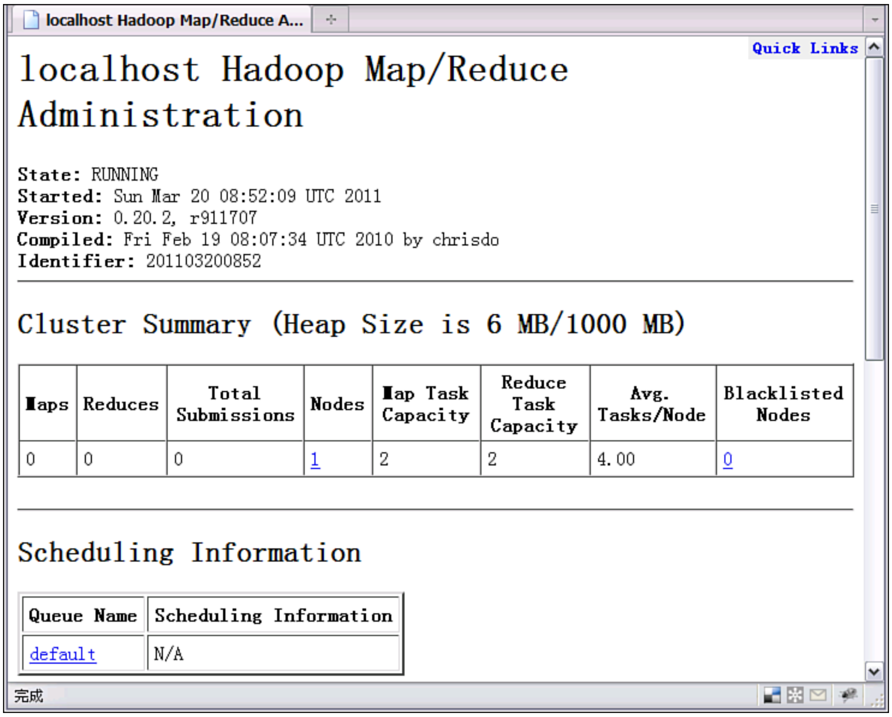
```
> hadoop namenode -format
```

Start the HDFS and MapReduce daemons using these commands:

```
> start-dfs.sh
> start-mapred.sh
```

Add a firewall rule for port 50030, as explained in the section “Firewalls” in Chapter 7. Bring up the Hadoop MapReduce Administration page, at http://host:50030, as shown in Figure 5.3.

Add another firewall rule for port 50070 and then bring up the NameNode administration page at address http://host:50070. This is shown in Figure 5.4.



localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Sun Mar 20 08:52:09 UTC 2011
Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Identifier: 201103200852

Cluster Summary (Heap Size is 6 MB/1000 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	0	1	2	2	4.00	0

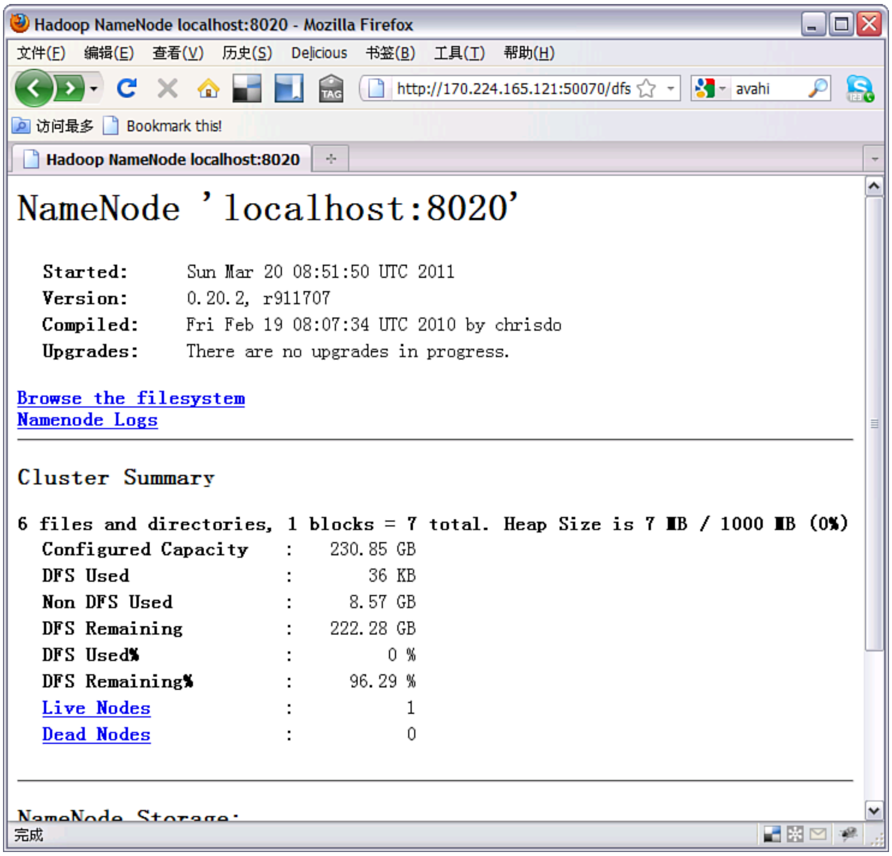
Scheduling Information

Queue Name	Scheduling Information
default	N/A

Figure 5.3 Hadoop MapRed administration page

Business Scenario: Data Management

We chose Hadoop for data management for IoT Data because it is a scalable, low-cost solution. MapReduce is a better framework for managing large amounts of data as a whole, compared to a relational database, which is more efficient at updating small pieces of data and point queries. MapReduce is also a more efficient system when the data is stored once and read many times. In IoT Data's business scenario, the devices store data continuously but never change data after it has been stored. In addition, MapReduce works well on the semistructured or unstructured data that the many different devices store to the data repository. Hadoop is also an extensible framework that will allow IoT Data to grow its business to applications, such as hosting applications to process the data stored by the network of devices. The enterprise might find it useful to support applications that are very data intensive, such as processing environmental data remotely collected from satellites.


Hadoop NameNode localhost:8020 - Mozilla Firefox
文件(F) 编辑(E) 查看(V) 历史(S) Delicious 书签(B) 工具(T) 帮助(H)
http://170.224.165.121:50070/dfs
avahi
访问最多
Bookmark this!
Hadoop NameNode localhost:8020
完成

```
NameNode 'localhost:8020'

  Started:     Sun Mar 20 08:51:50 UTC 2011
  Version:     0.20.2, r911707
  Compiled:    Fri Feb 19 08:07:34 UTC 2010 by chrisdo
  Upgrades:    There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

6 files and directories, 1 blocks = 7 total. Heap Size is 7 MB / 1000 MB (0%)
  Configured Capacity   :   230.85 GB
  DFS Used              :       36 KB
  Non DFS Used          :     8.57 GB
  DFS Remaining         :   222.28 GB
  DFS Used%             :         0 %
  DFS Remaining%        :     96.29 %
  Live Nodes            :           1
  Dead Nodes            :           0

NameNode Storage:
```

Figure 5.4 Hadoop NameNode administration page

This page intentionally left blank

CHAPTER 6

Cloud Services and Applications

The goal of this chapter is to give ideas, show examples, and discuss possible directions for cloud applications, especially those that make use of IaaS management APIs. Many interesting ideas for creating cloud applications combine Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Examples of the recent activity in the area of cloud applications, although not IaaS, are the SalesForce AppExchange, Google Chrome Web Store, and Mozilla Prism, a new framework for hosting web applications that is currently under development.

Creating and Customizing Images

Virtual machine images are important assets in cloud computing and can be a building block for cloud services. Most IaaS clouds enable users to save instances as images and then to reinstantiate that image later, as needed. This capability enables you to easily save images that you customize and share them with your colleagues. However, depending on how widely you want to share your image, you might need to go further to make it easy to use and extend your image. The IBM SmartCloud Enterprise offers many options for customizing and sharing images with others, especially for images that will become master images added to the catalog available to all other cloud users. This section looks at some of these concepts and how to use the features on the IBM SmartCloud Enterprise.

Creating an image for use by others can include these steps:

1. Extend a base operating system image by adding data and software.
2. Customize firewall and other security settings to allow your image to work easily out of the box.

3. Clean up runtime data.
4. Add terms and conditions for the use of your software.
5. Parameterize settings that will vary from user to user.
6. Create startup scripts.
7. Create a topology describing your image.
8. Create user documentation.
9. Create a catalog description.

Virtual machine instances on the IBM SmartCloud Enterprise consist of these elements:

- The bits that make up the image of the virtual machine image, including the operating system and other software.
- Documentation on how to use the virtual machine, including a "Getting Started Guide," stored in the image catalog in Rational Asset Manager.
- Elements that you need to extract or templatize so that they can be set by the user who creates the image. This includes a `parameters.xml` file for passing parameters, such as initial passwords.
- Data values supplied to you by the user who creates an instance of the image, such as the values of initial passwords.
- Data values supplied by the cloud, such as IP address and host name.

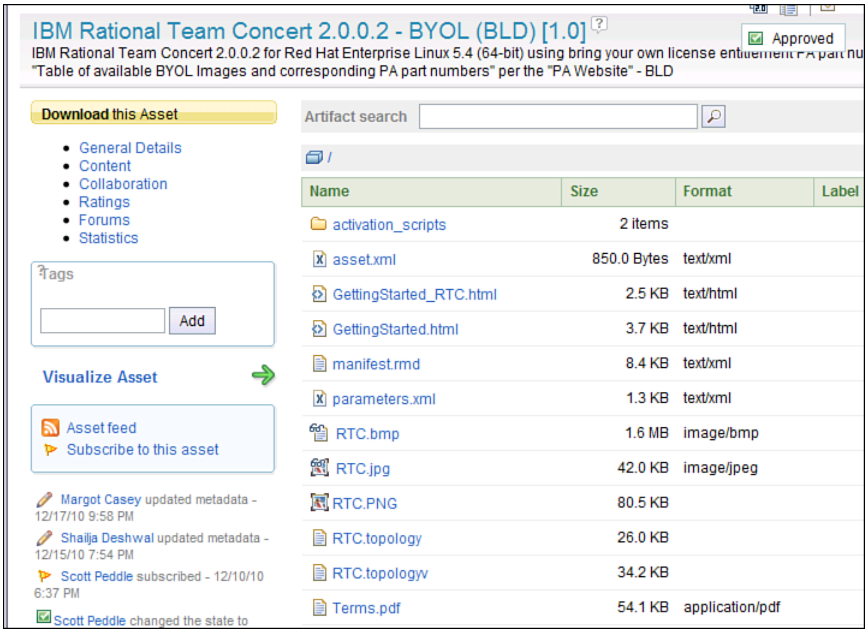
When creating an image that you want to share with others, consider which elements will be unique to each particular instance. IP address and host name will be unique to every instance, but your image might have something special, such as a particular kind of user ID and password or the URL of a service. For example, the WebSphere image used in the section "WebSphere Application Server" in Chapter 2, "Developing on the Cloud," gave the user the choice of a development profile or a production profile when provisioning the WebSphere server. The IBM SmartCloud Enterprise API enables you to pass this data from your application through to the instance when it starts up, without requiring the application to ever connect directly to the instance. As an example, Figure 6.1 shows the files that are included with the IBM Rational Team Concert image.

Parameters passed into the image at creation time will be passed into the instance and will be available under `/etc/cloud/parameters.xml`. A basic image with no parameters passed in at instance creation time has a simple `parameters.xml` file that looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<parameters>
  <firewall>
    <rule>
      <source>0.0.0.0/0</source>
      <minport>1</minport>
      <maxport>65535</maxport>
```

```
            </rule>
          </firewall>
        </parameters>
```

This contains firewall data for iptables.



IBM Rational Team Concert 2.0.0.2 - BYOL (BLD) [1.0]

Approved

IBM Rational Team Concert 2.0.0.2 for Red Hat Enterprise Linux 5.4 (64-bit) using bring your own license entitlement PA part nu
"Table of available BYOL Images and corresponding PA part numbers" per the "PA Website" - BLD

Download this Asset

- General Details
- Content
- Collaboration
- Ratings
- Forums
- Statistics

Tags

Add

Visualize Asset

Asset feed
Subscribe to this asset

Margot Casey updated metadata - 12/17/10 9:58 PM
Shailja Deshwal updated metadata - 12/15/10 7:54 PM
Scott Peddle subscribed - 12/10/10 6:37 PM
Scott Peddle changed the state to

Artifact search

/

Name	Size	Format	Label
activation_scripts	2 items		
asset.xml	850.0 Bytes	text/xml	
GettingStarted_RTC.html	2.5 KB	text/html	
GettingStarted.html	3.7 KB	text/html	
manifest.rmd	8.4 KB	text/xml	
parameters.xml	1.3 KB	text/xml	
RTC.bmp	1.6 MB	image/bmp	
RTC.jpg	42.0 KB	image/jpeg	
RTC.PNG	80.5 KB		
RTC.topology	26.0 KB		
RTC.topologyv	34.2 KB		
Terms.pdf	54.1 KB	application/pdf	

Figure 6.1 Files included with the IBM Rational Team Concert image catalog entry

The Linux operating system provides startup customization integration points using scripts in the `init.d` directory, which are linked to the `rc.d` directories for each of the Linux run levels. The scripts also act as interfaces to services, allowing management by a system administrator. The Red Hat base images contain scripts `/etc/init.d/cloud-startup3.sh` and `/etc/init.d/cloud-startup5.sh` that can be used to provide integration points for startup customization. The SUSE Linux Enterprise images have similar scripts. Windows customization and startup is different because Windows virtual machine images cannot be manipulated as open file systems.

The file `scripts.txt`, stored in the image catalog entry in Rational Asset Manager (RAM), is a special file that stores name-value pairs of files that are copied to the instance during provisioning. The name is the name of the file in RAM, and the value is the target location for the file to be placed on the instance after provisioning. Consider the contents of `scripts.txt` for the IBM Rational Team Concert image:

```
rtcsvc=/etc/init.d/cloud-startup3.sh
```

See the paper "Creating and Customizing Images" [Goodman, 2009] for more on images on the IBM SmartCloud Enterprise. The paper describes a simple cloud storage application based on an NFS server. It uses a Perl script to extract location and permission data supplied by the end user at provisioning time to configure the NFS server. Perl is a good choice of language to do this for Linux operating system features because it is lightweight and present with the Linux images provided on IBM SmartCloud Enterprise.

A cloud virtual machine image is not the same as a snapshot of a virtual machine. An image is intended to be portable and to be reused. A snapshot is intended as a backup of a virtual machine and includes runtime data. To our knowledge, no public clouds support snapshots in this sense. However, the concept is demonstrated by VMWare virtualization products that enable you to save a snapshot of a virtual machine. Multiple snapshots are related to each other, and only incremental data is saved, resulting in storage and performance efficiency. In VMWare terminology, a virtual machine image is called a template.

When creating an image to share with others, make sure that you clean up all logs on the system. Otherwise, you will be sharing log information with people who use your image.

Operating Systems Specifics

Some commands work differently on different systems. In this book, we use the Red Hat Enterprise Linux and SUSE Linux Enterprise operating systems primarily for examples. Many other Linux distributions and UNIX systems are similar. If you need to programmatically determine the operating system and version, you can do so in several ways. To find basic information on the Linux kernel, use the `uname` command:

```
# uname -a
Linux vm-10-200-7-154 2.6.18-194.el5 #1 SMP Tue Mar 16 21:52:39 EDT
2010 x86_64 x86_64 x86_64 GNU/Linux
```

Red Hat systems have a file called in the `/etc` directory—for example:

```
# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 5.5 (Tikanga)
```

Some other Linux distributions maintain entries in the `/proc` directory tree—for example:

```
# cat /proc/version
Linux version 2.6.18-194.el5 (mockbuild@x86-005.build.bos.redhat.com)
(gcc version 4.1.2 20080704 (Red Hat 4.1.2-48)) #1 SMP Tue Mar 16
21:52:39 EDT 2010
```

Modeling Deployment Topologies

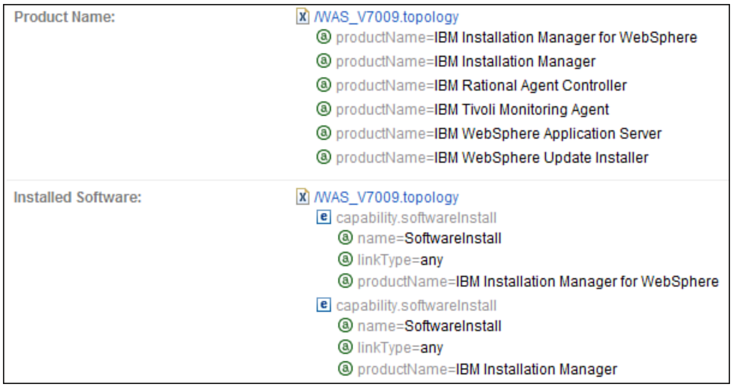
Deployment topology modeling is an area being vitalized with considerable recent activity, and it has important applications in cloud computing. A deployment topology model describes how a solution can be deployed and configured. IBM Rational modeling tools, particularly Rational

Software Architect, can help you create deployment topologies that describe deployment architectures, integrate with UML models, and integrate with Rational Asset Manager. The tools are intended to bridge the gap between design and deployment. Application developers and deployment engineers thus can plan a proper environment for an application, including sufficient hardware and dependent software with correct versions. Enterprises also can thus enforce best practices and standards on deployments, such as standard hardware and software stacks. The topology palette includes middleware, operating systems, hardware, networking, storage, and virtualization, and it enables you to add your own items.

Deployment topology modeling encourages a number of best practices, including these:

- Thinking about deployment earlier in the cycle and prompting designers and developers to think about deployment
- Sharing and reusing deployment topologies
- Enforcing architectural decisions
- Validating deployment scenarios

In the IBM SmartCloud Enterprise, details from the topology models for each image are shown with the asset in the catalog in Rational Asset Manager. Using these, you can reuse the expert knowledge of the people who created the images directly. As an example, Figure 6.2 shows the topology model for the WebSphere Application Server image.



```
Product Name:          /WAS_V7009.topology
                         productName=IBM Installation Manager for WebSphere
                         productName=IBM Installation Manager
                         productName=IBM Rational Agent Controller
                         productName=IBM Tivoli Monitoring Agent
                         productName=IBM WebSphere Application Server
                         productName=IBM WebSphere Update Installer

Installed Software:    /WAS_V7009.topology
                         capability.softwareInstall
                           name=SoftwareInstall
                           linkType=any
                           productName=IBM Installation Manager for WebSphere
                         capability.softwareInstall
                           name=SoftwareInstall
                           linkType=any
                           productName=IBM Installation Manager
```

Figure 6.2 Partial topology for WebSphere application server in IBM SmartCloud Enterprise catalog

Topologies contain units and links. Units are parts of the deployment architecture, such as hardware, an application server, a relational database, an operating system, EAR and WAR files, and so on. Links show the connections between the units. You can begin planning directly with a topology diagram, or you can import elements from a UML model into a topology that provides a logical model of the application. To create a topology model, switch to Deployment view and select File, New, Topology from the main menu in Rational Software Architect.

Topology models can have one or more diagrams. The diagrams show different views, such as a subsystem or a specific aspect, but the model itself contains the items and the links. Different levels of abstraction can also exist—typically these:

- Logical model, which shows the application and infrastructure at a high level
- Physical model, which shows the specific types of servers and devices
- Deployment model, which shows the actual servers to be deployed

You can get started with topology models using the Rational Software Architect 8.0 image available on the IBM SmartCloud Enterprise. This image makes use of the NX Client remote desktop, as explained in the later section "NX Remote Desktop." After provisioning a Rational Software Architect instance and connecting via the NX Client, you should see a desktop like the one in Figure 6.3.

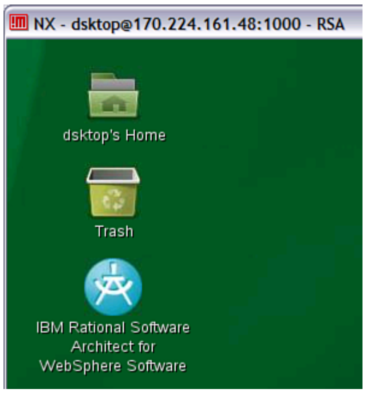

NX - dsktop@170.224.161.48:1000 - RSA
dsktop's Home
Trash
IBM Rational Software Architect for WebSphere Software

Figure 6.3 Rational Software Architect remote desktop

Double-click on the Rational Software Architect desktop icon to start it. Then follow these steps:

1. Dismiss the Welcome screen.
2. Create a new modeling project using the File, New, UML Project menu item. Name the project IoTData.
3. Switch to Deployment view using the menu item Windows, Open Perspective, Other and then selecting Deployment.
4. Create a new topology model using the File, New, Topology menu item.

Modeling at a logical level can be an easy way to begin the deployment model and enable you to start earlier in the development cycle. In a logical model, you typically show the logical components of the application, computing nodes, locations, and constraints. This can include specific constraints, such as the version of a Java runtime environment or other software dependencies, but it can also leave out some items. For example, your application might not depend on a specific operating system, so you can leave that out, even though some kind of operating system is needed for a real deployment. As an example, Figure 6.4 shows a logical model of a storage application.

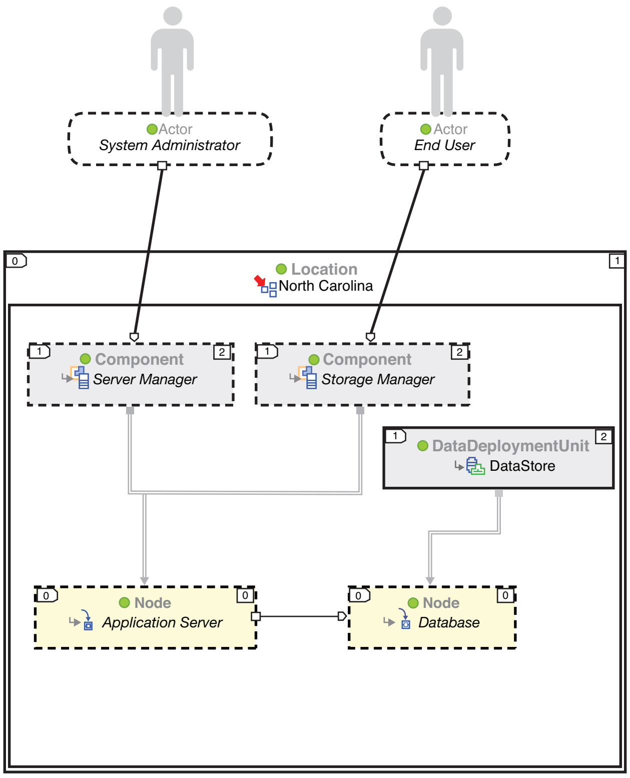
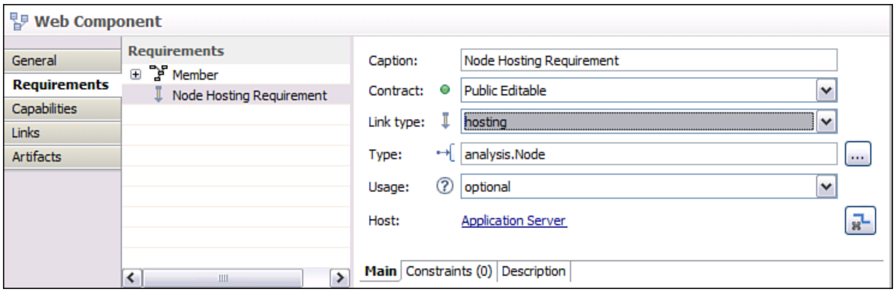


Figure 6.4 Logical topology model of IoT data application

In the logical model, there are two components: `storageManager` and `serverManager`. There is a hosting link between these components and the node that represents the application server they run on. There is also a data store and a hosting link to the database server it resides on. A network connection constraint link exists between the application server and the database server. At this point, the Node items represent servers in general; we do not know what specific kinds of servers they are or what operating systems would run on them. The servers are all located at the North Carolina data center. The model was created in Rational Software Architect by importing a UML model with components and dragging items from the topology palette to the diagram.

A hosting link is a constraint that tells you that you need a unit to a particular component. For example, a web component needs to be hosted on an web application server. The hosting constraint in Figure 6.5 shows how to do that.


Web Component

General | Requirements | Capabilities | Links | Artifacts

Requirements
- Member
- Node Hosting Requirement

Caption: Node Hosting Requirement
Contract: Public Editable
Link type: hosting
Type: analysis.Node
Usage: optional
Host: Application Server

Main | Constraints (0) | Description

Figure 6.5 An example of a hosting constraint

An **infrastructure diagram** shows different network or security zones. An example of an infrastructure diagram for the IoT Data system is shown in Chapter 7, "Security."

A **physical model** shows the specific types of hardware, operating systems, middleware, hardware, and other devices. A realization link connects the logical model to the physical model by introducing a constraint that tells what kind of unit a node can be realized as. A realization link is more specific than a hosting link. For example, your web component has a hosting requirement on a web application server that can be realized with WebSphere Application Server (see Figure 6.6).

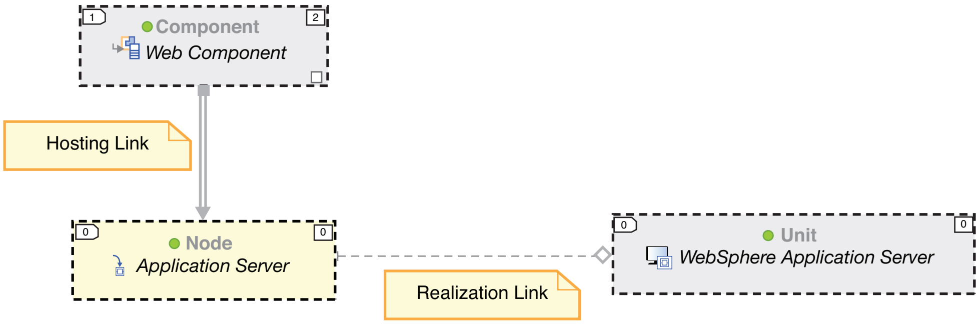


Figure 6.6 Hosting links and realization links

An infrastructure diagram at a physical level might show DMZ, Internet, intranet, and physical network zones.

The topology files for physical models for the images in IBM SmartCloud Enterprise are included in the catalog, hosted in Rational Asset Manager. Find the image in the catalog, click Content, and download the `.topology` file. To see it in Rational Software Architect, copy the topology file to the topologies directory of your Rational Software Architect project and refresh it. Figure 6.7 shows the topology diagram for Websphere 7.0.

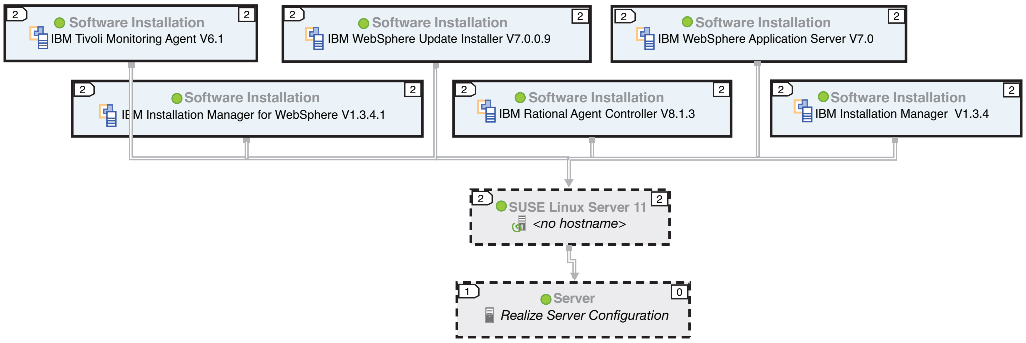

Software Installation
IBM Tivoli Monitoring Agent V6.1
Software Installation
IBM WebSphere Update Installer V7.0.0.9
Software Installation
IBM WebSphere Application Server V7.0
Software Installation
IBM Installation Manager for WebSphere V1.3.4.1
Software Installation
IBM Rational Agent Controller V8.1.3
Software Installation
IBM Installation Manager V1.3.4
SUSE Linux Server 11
<no hostname>
Server
Realize Server Configuration

Figure 6.7 IBM SmartCloud Enterprise WebSphere Application Server 7.0 topology model

The topology shows the software installations included, each of which links to the operating system, SUSE Linux Enterprise 11. The different feature packs and Java Development Kit are included within the WebSphere Application Server installation. The IBM SmartCloud Enterprise catalog includes two topology files for each catalog entry: a `.topology` file that describes the model and a `.topologyv` file that describes the diagram. Figure 6.8 shows the DB2 9.7 topology model.

This model includes the installations included, as well as the database instance and a sample database. These physical items can be linked to the logical model of your application using realization links. You can add these topologies to Rational Software Architect by right-clicking them in the project explorer and selecting the Add to Palette menu item.

A *deployment model* creates an application scenario. It includes the host names of the actual servers the application will be deployed on. In the earlier WebSphere Application Server and DB2 examples, the server configurations are omitted. During this process, Rational Software Architect warns you of unresolved dependencies, which ensures that you have satisfied all the necessary prerequisites for deployment. For production installation, you can go further, including specifying an IP subnet, firewall, and other items.

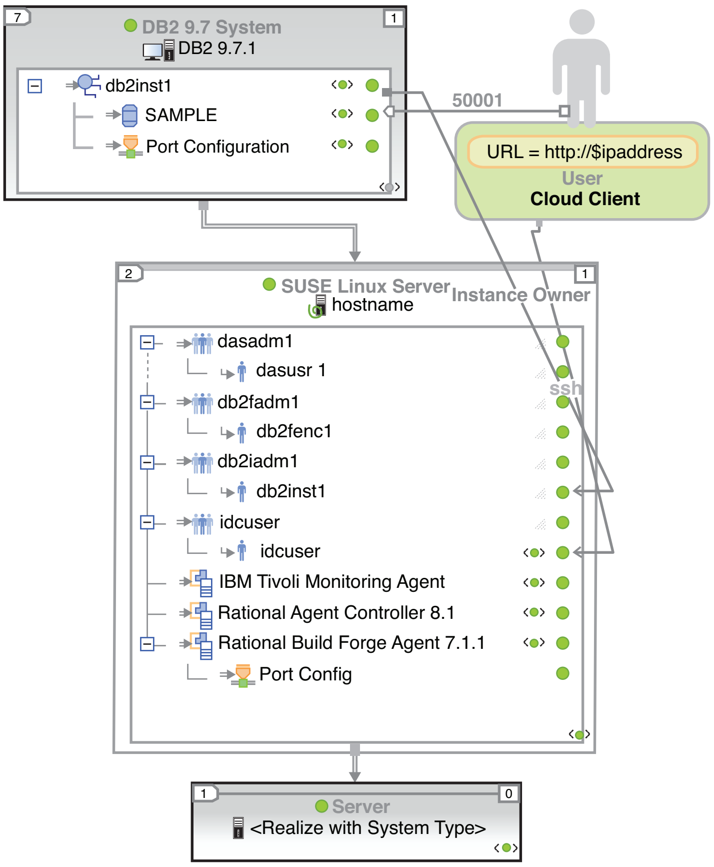


Figure 6.8 IBM SmartCloud Enterprise DB2 Enterprise 9.7 topology model

Deployment models are so specific that they can be used by tools for automated deployment. You can use topology models with Rational Build Forge to model automated deployment tasks. WAS also provides additional functionality for automation with topology models. You can integrate with Tivoli Change and Configuration Management Database. Topology modeling is used later in the section "Cloud Software Bundles."

Services

This section covers some relatively old technologies that are finding new uses in cloud computing, especially in the area of virtual machine image development.

Linux Services

When customizing images, you will probably make use of system services. In Linux, the `init` command manages process control initiation. It is the parent of all processes, invoked as the last step in the kernel boot process. For the RHEL and SUSE images in the IBM SmartCloud Enterprise catalog, this creates the processes stored in the `/etc/inittab` file based on UNIX System V–style run levels. Table 6.1 shows typical Linux run levels as used in Red Hat and SUSE.

Table 6.1 Typical Linux Run Levels

Level	Description
0	Shut down the system
1	Run in single-user mode
2	Run in multiuser mode without networking or starting daemons
3	Run in multiuser mode with networking, with console login only
4	Not used
5	Run in multiuser mode with networking and X Windows
6	Reboot the system

We usually manage services with commands other than `init`, such as the `service` and `chkconfig` commands. You can use the `service` command to find the status of the different services on a system, as shown here:

```
# /sbin/service --status-all
anacron is stopped
atd (pid  2384) is running...
auditd (pid  1741) is running...
automount (pid 2089) is running...
...
```

In general, the command is used to run an init script located in `/etc/init.d/SCRIPT`. It has this form:

```
# service SCRIPT COMMAND [OPTIONS]
```

Most scripts have at least a `start` and a `stop` function. For example, the script for running the firewall iptables service `/etc/init.d/iptables` contains the functions `start`, `stop`, `save`, `status`, and `restart`, among others. You can use the `service` command to start the iptables service, as shown:

```
# /sbin/service iptables start
```

A related command is `chkconfig`, which updates and queries run-level information for system services in the `/etc/rc[0-6].d` directories. To query system services, use a `-list` argument:

```
# /sbin/chkconfig --list
NetworkManager  0:off   1:off   2:off   3:off   4:off   5:off   6:off
acpid           0:off   1:off   2:on    3:on    4:on    5:on    6:off
anacron         0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
...
```

This tells you which services are on and which are off for run levels 0 through 6. A common use of the command is to turn services on and off with this form:

```
# chkconfig [--level levels] name <on|off|reset|resetpriorities>
```

Some images are set up with iptables blocking all inbound traffic except port 22 (SSH). As an example, to turn off the iptables service altogether (a poor security setting), use this command:

```
# /sbin/chkconfig iptables off
```

You can also add and delete services with the `--add` and `--del` options.

The file `/etc/services` contains a file with friendly names of network services, their ports, and their protocol types. Port numbers lower than 1,024 can be bound only by root. The general form of an entry in services is shown here:

```
service-name   port/protocol   [aliases …]
```

`service-name` is the friendly name of the service. The protocol is typically either `tcp` or `udp`. Typical entries in the table are like the examples shown shortly for FTP and SSH.

```
...
ftp             21/tcp
ssh             22/tcp       # SSH Remote Login Protocol
...
```

This means that FTP should run on port 21 with TCP, and SSH should run on port 22 with TCP. A service is not necessarily running because it appears in the `services` file. Use the `service` command to find the status of services.

You can also manage network services with the `service` command. For example, to restart network services, use this command:

```
# /sbin/service network restart
```

The IBM SmartCloud Enterprise has startup scripts `cloud_startup3.sh` and `cloud_startup5.sh` in the `/etc/init.d` directory. You can modify these scripts to customize the behavior of your instance; they will be executed by `root`. To execute commands using another user at startup, use the `su` command:

```
su username -c "command to run as username"
```

The -c option tells su to execute the command following it.

The xinetd daemon is the extended Internet services daemon on Linux. It replaces the inetd daemon on older UNIX systems. Instead of having many network services constantly running and listening on port, the xinetd service wraps them and invokes them when a connection request is made. It takes advantage of the /etc/xinetd.conf file to know what services to wrap and the services file to determine which ports to use. Restart xinetd after making changes to configuration files with this command:

```
# /sbin/service xinetd restart
```

The xinetd service is fine to use for infrequently used daemons, such as those used by system administrators, but it performs poorly for heavily used services provided to external users.

Windows Services

On Windows instances, the parameters file is copied to the location C:\Cloud\parameters.xml. The files FirstProvisionStartup.bat and VMInstanceStartup.bat are executed on startup.

You might find that you need to interoperate between Linux and Windows. Cygwin is an open source tool that can help you do this. It includes many command-line and graphical UNIX utilities. With Cygwin installed, you can run a bash shell and many ports of Linux tools to Windows. Windows is probably the most popular system for personal workstations. Installing Cygwin can be a good option for installation on your personal workstation for connectivity and compatibility with cloud-based Linux tools.

Networking

Some basic knowledge of networking is required to configure your applications. A more detailed understanding is required for composite applications spanning different virtual machines, and an advanced knowledge of networking is required if you develop an application that provides network services. Let's start with some basic settings.

Basic Network Settings

You will always have an IP address to connect to your virtual machine in the cloud. You should also have a host name for the virtual machine, which will be assigned by the cloud provider. To find the host name of your virtual machine in a Linux shell, use the hostname command, which shows or sets the system host name—for example:

```
# hostname
vhost0297
```

This tells you that the host name is vhost0297. The nslookup command queries the domain name system (DNS) server to give information about the network. The most basic use is to find the IP address for a given host name. To find out more details about your own host, use the nslookup command with the -sil options:

```
# nslookup -sil vhost0297
Server:         170.224.55.202
Address:        170.224.55.202#53
Name:   vhost0297.site1.compute.ihost.com
Address: 170.224.161.42
```

In the previous command, substitute the host name of your own server for `vhost0297`. This tells you that the fully qualified host name is `host0297.site1.compute.ihost.com`. The IP address of the DNS (`170.224.55.202`) and the IP address of your own virtual machine (`170.224.55.202`) are also listed.

A reverse lookup finds the host name for a given IP address. Some applications, such as mail servers, do this as a security check. You can check that reverse look-up is working properly by providing your IP address as a parameter for `nslookup`, as shown here:

```
# nslookup -sil 170.224.161.42
Server:         170.224.55.202
Address:        170.224.55.202#53
42.161.224.170.in-addr.arpa     name =
vhost0297.site1.compute.ihost.com
```

Substitute your own IP address in place of `170.224.161.42`. The result here shows that you get the host name expected. If you have trouble with your host name or DNS, you might have to check the configuration settings in the file `/etc/resolv.conf`.

Sometimes you need to check what network services are listening for connections on ports. You can use the `netstat -nl` command, which lists network connections, routing tables, and interface statistics, to do this. The `n` option is for numeric data, and the `l` option is for listening—for example:

```
# netstat -nl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address     State
...
tcp        0      0 :::80              :::*                LISTEN
tcp        0      0 :::22              :::*                LISTEN
tcp        0      0 :::443             :::*                LISTEN
...
```

This tells us that our virtual machine is listening for TCP connections on ports 22, 80, and 443.

On Linux, network ports lower than 1024 can be opened only by root. If you want to run as another user, you need to use a port greater than or equal to 1024. Alternatively, you can switch to root for the purpose of opening the port and then switch back to a less privileged user to run the service. The latter method is the way important services, such as HTTP, mail, FTP, and SSH, run.

Software Installation and Management

Windows and different distributions of Linux have different software installation and management utilities. The IBM SmartCloud Enterprise also has a method for installing software during the instance provisioning process.

Red Hat Package Management and YUM

The Red Hat Package Management tool is a command-line desktop graphical tool for managing software. RPM is also a file format to package software for installation. It can be useful to list installed software, locate software to install, and resolve dependencies needed during installation. However, our focus here is on automation, which can be done via the Yellowdog Updater Modified (YUM). YUM is a package manager that was originally intended to improve the installation of RPM files. It has become Red Hat's main tool for managing software. The Red Hat up2date tool for updating the operating system and installed files is now deprecated in favor of YUM.

YUM enables you to set up your own software repository to augment Red Hat's and software repository and other repositories provided by third parties. To set up your own repository install using YUM, do this:

```
# yum install createrepo
```

Then run `createrepo` to set up the repository. These common YUM commands install a package:

```
# yum install <package name/s>
```

This searches, downloads, and installs the software and any dependencies for you automatically. This command updates a package:

```
# yum update <package name/s>
```

This command checks for any updates for installed software:

```
# yum check-update
```

The next command removes a package:

```
# yum remove <package name/s>
```

YUM is configured with the `/etc/yum.conf` file. The YUM update notifier daemon `yum-updatesd` runs in the background to check for updates. It is configured via the `yum-updatesd.conf` file.

Software Management on SUSE

Software can be installed on SUSE using YaST, apt-get install, Zypper. YaST is a general-purpose graphical tool for system administration. Zypper and apt-get install are command-line tools for package management. Zypper is based on the C library libzypp, which helps resolve package dependencies. In fact, YaST is now based on libzypp.

Zypper can be used to search for installed software, as shown here:

```
$ zypper search --search-descriptions <package_name>
```

`<package_name>` is the name of the package to search for. Zypper can install software, as shown next:

```
# zypper install <package_name>
```

For example, this command installs emacs:

```
# zypper install emacs
```

Some other useful commands are listed here:

- `refresh`, to refresh package information
- `addrepo`, to add a new repository
- `remove`, to uninstall a software package
- `list-updates`, to list the available updates or patches

Cloud Software Bundles

The cloud infrastructure can also install software. The motivation for doing this is to reduce the number of images that need to be created. Suppose you have a software application that is supported on Windows, Red Hat Enterprise Linux, and SUSE for the current and previous version of each operating system. Six images are already needed. If it can be either bundled with another product or provided by itself, the number of images becomes 12. For utility software, such as editors, agents, and database drivers, no good solution uses images alone. The need for installing software in clouds, either in an automated way or via user self-service, is apparent. The IBM SmartCloud Enterprise supports Composable Software Bundles to address this need. It leverages IBM Rational Asset Manager to achieve this.

In the earlier "Creating and Customizing Images" section, we discussed startup scripts and copying files in the Rational Asset Manager catalog entries to virtual machine instances at provisioning time. A software bundle is a Rational Asset Manager asset that describes software to be installed to an instance at startup time, including files to be copied, but does not include a virtual machine image. The files include installation files, configuration files, a parameter specification file, and a description of prerequisites required. Figure 6.9 illustrates this concept.

Composable Software Bundles can delegate installation of software to YUM, Zypper, or another software management utility. You can get started creating a software bundle by copying the software bundle template in the asset catalog. Add an installation script, any other files, and topology files. The topology files include a `TEMPLATE-Automation.topology` file and a `TEMPLATE-Semantic.topology` file. These can be edited by hand or by using Rational Software Architect.

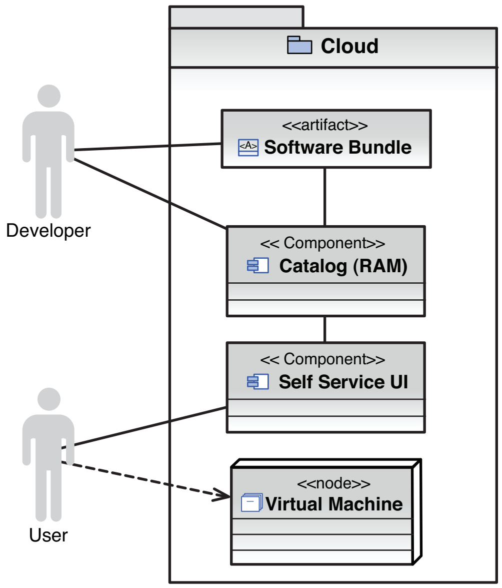


Figure 6.9 Software bundles concept

To browse the software bundles in the IBM SmartCloud Enterprise, in the catalog, remove the default filter and then choose the Composable Software Bundles link in the left margin. Let's use the emacs software bundle as an example. The bundle includes the file `scripts.txt`, which includes these files:

- `ryo.crontab`, a cron job definition
- `ryoCron.sh`, which waits for the network to become available before running the installation script `ryoAutomation.sh`
- `ryoAutomation.sh`, which checks to see if this is the first time it has been executed and, if needed, calls the script
- `installEmacs.sh`, which installs emacs using zypper

To add a software bundle to an image, download the software bundle files and merge the files with your image files.

Open Service Gateway Initiative (OSGi)

The Open Service Gateway initiative (OSGi) is a standard from the OSGi Alliance describing a component configuration architecture for Java. It is used to assist the plug-in framework for a number of products and open source projects, such as Eclipse, IBM Rational Software Architect, IBM Tivoli Federated Identity Manager (FIM), mobile phones, embedded devices, and most J2EE application servers. OSGi plays a critical role in defining versions and dependencies of software bundles and gives software users flexibility in choice of feature use while maintaining good performance. This section offers a brief overview of OSGi, comments on the problems relevant to cloud that OSGi addresses, and shows how to create an Eclipse plug-in that lists your IBM SmartCloud Enterprise virtual machine instances.

OSGi enables these actions:

- Install, uninstall, stop, and start a module
- Run more than one version of a module at the same time
- Manage dependencies

For example, consider the IBM SmartCloud Enterprise Java API has a dependency on the open source library Apache HttpClient 3.1. After this was released, the Apache HttpClient team introduced a new programming model that is incompatible with old versions. What happens if you need to write a program that uses a newer version of Apache HttpClient and also use the IBM SmartCloud Enterprise Java API? This is a problem that you can solve with OSGi.

Eclipse is the poster child for OSGi. In 2003, Eclipse adopted OSGi for its 3.0 version, which was released in 2004. Eclipse is actually a collection of plug-ins based on the Eclipse Rich Client Platform core, which provides an implementation of the OSGi framework called Equinox. Plug-ins include the Rational Software Architect cloud plug-in that enables you to create virtual machine instances in different clouds, such as IBM SmartCloud Enterprise. It is useful to describe OSGi with some examples relating to the Eclipse integrated development environment so that you can learn through concrete examples. Eclipse plug-ins add functionality in a structured manner. This allows the integration of both user interface and behind-the-scenes features.

A fundamental unit in OSGi is the *bundle*, which contains Java classes and other resources and defines a unit of software that can be published, discovered, and bound to a service. Rules for an OSGi bundle are enforced through special class loaders. In an ordinary Java program, all classes can see all other classes. However, the OSGi framework restricts interaction between bundles based on information in each bundle's `manifest.mf` file. Important directives in the `manifest.mf` file include these:

- `Bundle-Name` (name of the bundle)
- `Bundle-Version` (version of the bundle)
- `Require-Bundle` (defines a dependency on another bundle)
- `Export-Package` (exports the package)
- `Bundle-Activator` (class used to start and stop the bundle)

To create a an OSGi bundle in Eclipse 3.6, first create a new Plug-in Project:

1. From the File menu, choose New, Project, Plug-in Project.
2. In the dialog box, enter the name of the plug-in, `com.ibm.example.cloud.plugin`. Choose OSGi framework Equinox.
3. In the next dialog box, choose Java SE 1.6 as the execution environment. Generate an Activator called `com.ibm.example.cloud.plugin.Activator`.
4. In the next dialog box, choose to create the project with the Hello OSGi Bundle template.

A number of files are generated, including the following `manifest.mf` file:

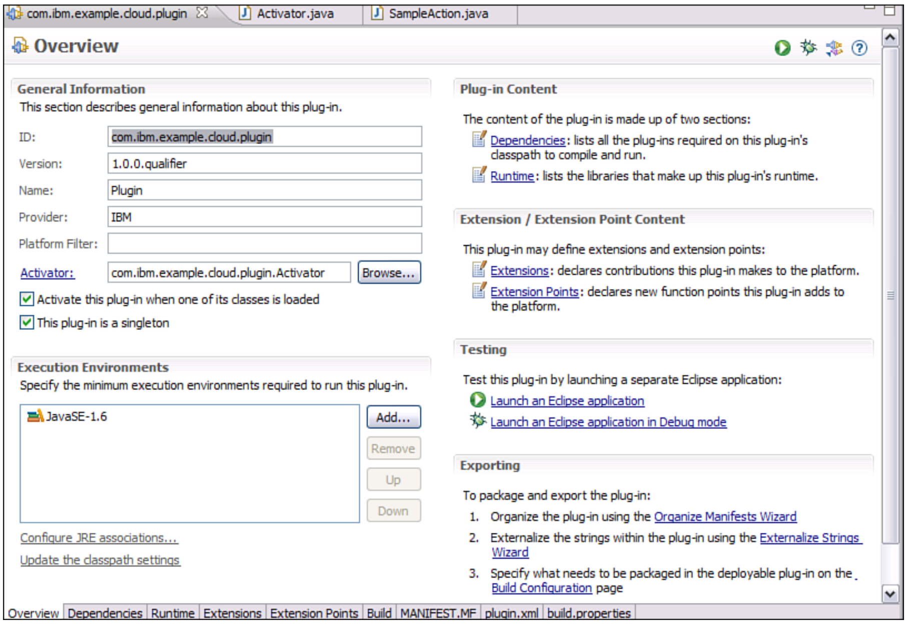
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Plugin
Bundle-SymbolicName: com.ibm.example.cloud.plugin; singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: com.ibm.example.cloud.plugin.Activator
Bundle-Vendor: IBM
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ActivationPolicy: lazy
```

The project wizard creates the two classes `Activator` and `SampleAction`. The `Activator` class is shown in the next listing.

```
package com.ibm.example.cloud.plugin;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

public class Activator extends AbstractUIPlugin {

      // The plug-in ID
      public static final String PLUGIN_ID =
"com.ibm.example.cloud.plugin"; //$NON-NLS-1$

      // The shared instance
      private static Activator plugin;

      public Activator() {      }
```

```
            public void start(BundleContext context) throws Exception {
                  super.start(context);
                  plugin = this;
            }

            public void stop(BundleContext context) throws Exception {
                  plugin = null;
                  super.stop(context);
            }

            public static Activator getDefault() {
                  return plugin;
            }

            public static ImageDescriptor getImageDescriptor(String path) {
                  return imageDescriptorFromPlugin(PLUGIN_ID, path);
            }
      }
```

The `BundleContext` parameter provides a number of methods for discovering the bundle's environment and listening for changes in that environment. Eclipse 3.6 has a Design view for the plug-in shown in Figure 6.10.


com.ibm.example.cloud.plugin | Activator.java | SampleAction.java

Overview

General Information
This section describes general information about this plug-in.

ID:	com.ibm.example.cloud.plugin
Version:	1.0.0.qualifier
Name:	Plugin
Provider:	IBM
Platform Filter:	
Activator:	com.ibm.example.cloud.plugin.Activator

[✓] Activate this plug-in when one of its classes is loaded
[✓] This plug-in is a singleton

Execution Environments
Specify the minimum execution environments required to run this plug-in.
JavaSE-1.6

Configure JRE associations...
Update the classpath settings

Plug-in Content
The content of the plug-in is made up of two sections:
Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.
Runtime: lists the libraries that make up this plug-in's runtime.

Extension / Extension Point Content
This plug-in may define extensions and extension points:
Extensions: declares contributions this plug-in makes to the platform.
Extension Points: declares new function points this plug-in adds to the platform.

Testing
Test this plug-in by launching a separate Eclipse application:
Launch an Eclipse application
Launch an Eclipse application in Debug mode

Exporting
To package and export the plug-in:
1. Organize the plug-in using the Organize Manifests Wizard
2. Externalize the strings within the plug-in using the Externalize Strings Wizard
3. Specify what needs to be packaged in the deployable plug-in on the Build Configuration page

Browse...
Add...
Remove
Up
Down
Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

Figure 6.10 Design view for plug-in in Eclipse 3.6

You can launch the bundle using the Launch the Framework link. A new instance of Eclipse is launched, with an additional menu called Sample Menu. Under the menu is a menu item called Sample Action. If you click that menu item, a dialog box pops up with the text “Hello, Eclipse World.”

By default, the classes in the plug-in are visible to all the other classes inside the plug-in but are not visible to classes outside the plug-in. So going back to the IBM SmartCloud Enterprise example, you can add Apache HttpClient 3.1 to your plug-in project without affecting use of a later version of HttpClient outside the plug-in. Let’s modify our code to display a list of instance owned by a SmartCloud Enterprise user:

1. Create a directory call lib under the project folder. Add the JARs `commons-codec-1.3.jar`, `commons-httpclient-3.1.jar`, `commons-lang-2.3.jar`, `commons-logging-1.1.1.jar`, and `DeveloperCloud_API_Client_JAR.jar` to this folder.
2. Refresh the project.
3. In the Runtime tab of plug-in Design view, add the jars to the classpath.

After completing the previous steps, create the class SmartCloud with this code:

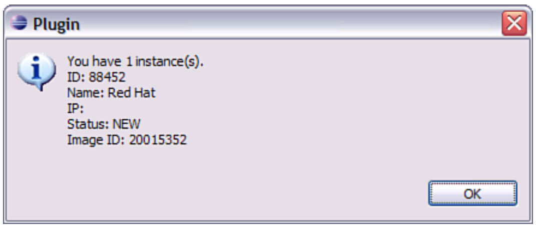
```
package com.ibm.example.cloud.plugin;

import java.util.List;

import com.ibm.cloud.api.rest.client.DeveloperCloud;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.Instance;

public class SmartCloud {
      public static final String USERNAME = "a@example.com";
      public static final String PASSWORD = "********";
      private DeveloperCloudClient client =
DeveloperCloud.getClient();

      public SmartCloud() {
            client.setRemoteCredentials(USERNAME, PASSWORD);
      }

      public String describeInstances() {
            try {
                  List<Instance> instances =
client.describeInstances();
                  String message = "You have " + instances.size() + "
instance(s).\n";
                  for (Instance instance: instances) {
                        message += "ID: " + instance.getID() + '\n';
```

```
                                message += "Name: " + instance.getName() +
'\n';
                                message += "IP: " + instance.getIP() + '\n';
                                message += "Status: " + instance.getStatus() +
'\n';
                                message += "Image ID: " +
instance.getImageID() + '\n';
                        }
                        return message;
                } catch(Exception e ) {
                        return "Error getting virtual machine instances: " +
e.getMessage();
                }
        }


}
```

Modify the class `Activator`, adding the code shown:

```
public void run(IAction action) {
        SmartCloud smartCloud = new SmartCloud();
        String message = smartCloud.describeInstances();
        MessageDialog.openInformation(
                window.getShell(), "Plugin", message);
}
```

Launch the plug-in from Plug-in Design view. Now when you click on the Sample Action menu item, your list of virtual machine instances will be shown as in Figure 6.11.



```
You have 1 instance(s).
ID: 88452
Name: Red Hat
IP:
Status: NEW
Image ID: 20015352
```

Figure 6.11 Example Cloud Eclipse plug-in

Eclipse plug-ins have some Eclipse-specific dependencies, but Eclipse also enables you to create a standard OSGi bundle without any Eclipse-specific code. You can do this by checking the OSGi framework option in the New Plug-in Project Wizard. We did this in IBM Rational Software Architect 7.0 and called the plug-in `com.ibm.example.cloud.osgi`. The New Plug-in Project Wizard generated the manifest file shown here:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Osgi Plug-in
Bundle-SymbolicName: com.ibm.example.cloud.osgi
Bundle-Version: 1.0.0
Bundle-Activator: com.ibm.example.cloud.osgi.Activator
Bundle-Vendor: IBM
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
```

Notice that there are no Eclipse dependencies, so you can use this OSGi bundle in standard containers. The wizard generates the class `Activator`, shown here:

```
package com.ibm.example.cloud.osgi;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

      public void start(BundleContext context) throws Exception {
            System.out.println("Hello World!!");
      }

      public void stop(BundleContext context) throws Exception {
            System.out.println("Goodbye World!!");
      }

}
```

The class `Activator` implements `BundleActivator`, which is an OSGi interface. To run the bundle, right-click on the root of the plug-in project and select Run As, Run Configurations. Change the name to Cloud Plug-in and uncheck all the target platform plug-ins, leaving only the workspace selected. Set Autostart to `True`. Click Apply and then Run. You should see the OSGi command line:

```
osgi> Hello World!!
```

We can issue commands in the OSGi console. Try entering the status command `ss`. You should see output similar to this:

```
osgi> ss

Framework is launched.
```

```
id      State       Bundle
0      ACTIVE       org.eclipse.osgi_3.4.3.R34x_v20081215-1030
1      ACTIVE       com.ibm.example.cloud.osgi_1.0.0
```

This tells you that the Eclipse OSGi framework plug-in and your own plug-in are launched. You can stop your bundle by entering the `stop` command:

```
osgi> stop 1
Goodbye World!!
```

When you stop the bundle, the text that you added to the `stop()` is printed. For a more serious plug-in, you could put clean-up code in this method. You can also add, update, and remove bundles using the OSGi console.

Example: OSGI Cloud Service

In this example, you examine use of OSGi import/export and hiding the implementation of the service. You might want to do this, for example, to write a cloud-management console that connects to multiple different clouds. This is important because different cloud clients might have incompatible libraries, and you need to be able to connect to the clouds in a way that is transparent to the client.

You will create two OSGi bundles. This bundle exports a service that connects to IBM SmartCloud Enterprise and lists the images belonging to a user:

```
com.ibm.example.cloud.osgi.smartcloud
```

This bundle imports the service and invokes it:

```
com.ibm.example.cloud.osgi
```

Create the `com.ibm.example.cloud.osgi.smartcloud` OSGi bundle using the techniques described. Define the interface `CloudClient` shown next:

```
package com.ibm.example.cloud.osgi.client;

/**
 * Example interface for a cloud client
 */
public interface CloudClient {

      void describeMyImages();

}
```

Add the SmartCloud API–dependent libraries to the bundle, including the JARs `commons-codec-1.3.jar`, `commons-httpclient-3.1.jar`, `commons-lang-2.3.jar`, `commons-logging-1.1.1.jar`, and `DeveloperCloud_API_Client_JAR.jar`. Create the class `SmartCloudEnterpriseClient`, shown later, that implements `CloudClient`.

```
package com.ibm.example.cloud.osgi.smartcloud.impl;

import java.util.List;

import com.ibm.cloud.api.rest.client.DeveloperCloud;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.Image;
import com.ibm.example.cloud.osgi.client.CloudClient;

public class SmartCloudEnterpriseClient implements CloudClient {

      public static final String USERNAME = "a@example.com";
      public static final String PASSWORD = "********";
      private DeveloperCloudClient client =
DeveloperCloud.getClient();

      public SmartCloudEnterpriseClient() {
            client.setRemoteCredentials(USERNAME, PASSWORD);
      }

      @Override
      public void describeMyImages() {
            try {
                  List<Image> images = client.describeImages();
                  for (Image image: images) {
                        if (image.getOwner().equals(USERNAME)) {
                              System.out.println("ID: " +
image.getID());
                              System.out.println("Name: " +
image.getName());
                              System.out.println("Status: " +
                              image.getState());
                              System.out.println("Location: " +
                               image.getLocation() + '\n');
                        }
                  }
            } catch(Exception e ) {
                  e.getMessage();
            }
      }

}
```

Now you can register the service using OSGi APIs in the `Activator` class, as shown here:

```
package com.ibm.example.cloud.osgi.smartcloud;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

import com.ibm.example.cloud.osgi.client.CloudClient;
import
com.ibm.example.cloud.osgi.smartcloud.impl.SmartCloudEnterpriseClient;

public class Activator implements BundleActivator {
      private ServiceRegistration serviceRegistration;

      public void start(BundleContext context) throws Exception {
            System.out.println("SmartCloud client started");
            CloudClient cloudClient = new
SmartCloudEnterpriseClient();
            serviceRegistration =
             context.registerService(CloudClient.class.getName(),
            cloudClient, null);
      }

      public void stop(BundleContext context) throws Exception {
            System.out.println("SmartCloud client stopped");
            serviceRegistration.unregister();
      }

}
```

Next, you export the `CloudClient` service from the `com.ibm.example.cloud.osgi.smartcloud` bundle. Do this by adding `com.ibm.example.cloud.osgi.client` to the exported packages in the Runtime tab of the bundle designer. Now import the package `com.ibm.example.cloud.osgi.client` to the `com.ibm.example.cloud.osgi.client` bundle by adding it in the Dependencies tab of the bundle designer. Finally, in the `Activator` class of the `com.ibm.example.cloud.osgi.client` bundle, you can use the service, as shown:

```
package com.ibm.example.cloud.osgi;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;

import com.ibm.example.cloud.osgi.client.CloudClient;
```

```
public class Activator implements BundleActivator {
      private ServiceReference serviceReference;

      public void start(BundleContext context) throws Exception {
            System.out.println("Cloud plug-in started");
            serviceReference =
             context.getServiceReference(CloudClient.class.getName());
            CloudClient cloudClient =
             (CloudClient)context.getService(serviceReference);
            cloudClient.describeMyImages();
      }

      public void stop(BundleContext context) throws Exception {
            System.out.println("Cloud plug-in stopped");
      }
\}
```

This executes the service when the bundle is loaded. Now you are ready to run the plug-in. Right-click the `com.ibm.example.cloud.osgi.client` bundle and select Run As, Run Configurations in RSA. This starts an OSGi console and loads the bundles, creating the following output:

```
osgi> SmartCloud client started
Cloud plug-in started
ID: 20017833
Name: WebSphere Hadoop
Status: AVAILABLE
Location: 41

ID: 20022207
Name: RSA 8.0 for Cloud Project
Status: AVAILABLE
Location: 101

ID: 20017142
Name: PHP Examples
Status: AVAILABLE
Location: 41
```

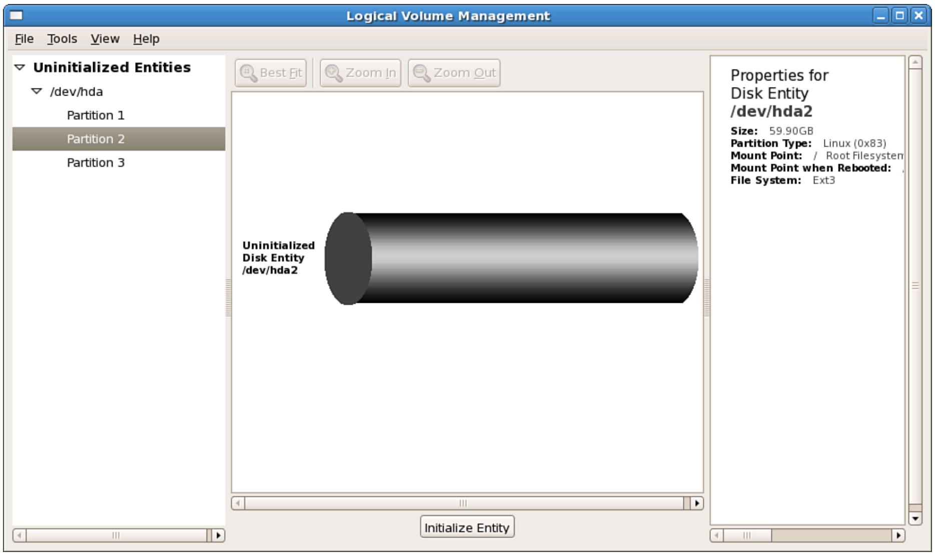
By importing on the service interface, the implementation classes for the IBM SmartCloud Enterprise client are hidden. This demonstrates an extensible method for creating a cloud client that can connect to many clouds without the risk of incompatible libraries.

Storage

Storage is one of the application areas for clouds that is most important to enterprises. We introduced storage virtualization concepts earlier in the "Storage" section in Chapter 1. Here we discuss some basics of storage usage, focusing on Linux, and explore how these can be translated into cloud applications.

Block Storage

After you have provisioned storage volumes and attached them to virtual machine instances on the cloud, you can manage the storage with native operating system tools. These include both command-line and graphical tools. Figure 6.12 shows the Logical Volume Manager on RHEL.



Properties for Disk Entity **/dev/hda2**

Size: 59.90GB
Partition Type: Linux (0x83)
Mount Point: / Root Filesystem
Mount Point when Rebooted:
File System: Ext3

Figure 6.12 Logical Volume Manager user interface

Figure 6.12 shows a local volume on a SmartCloud Enterprise Copper RHEL virtual machine. A 64-bit Copper compute size has 2 CPUs, 4 GB of memory, and 60 GB of storage.

One of the interesting points about Linux block storage is that you can make block storage devices out of files; these are called *loop devices*. This has some interesting virtualization and cloud applications, particularly in virtual machine image creation, customization, and management. With basic Linux tools, we can make direct attached storage, network attached storage, and a loop storage device all look like the same thing to a guest operating system.

The dump data `dd` command can be used to copy and convert from physical disks. To demonstrate the use of `dd`, we use the input file and output file options. For example, to copy a file as a sequence of bytes, use this sequence of commands:

```
# echo test  > test
# dd if=test of=test.bak
0+1 records in
```

```
0+1 records out
5 bytes (5 B) copied, 4.8692e-05 seconds, 103 kB/s
# cat test.bak
test
```

Using the `dd` command with the input file (`dd`) and output file (`of=test.bak`) options simply copies the input file to the output file. We could have done the same thing with the `cp` command, but `dd` enables us to specify lower-level details, such as the block size with the `bs` option and the number of blocks to copy with the `count` option. The `seek` option tells `dd` how many blocks to skip. We can put these options together to create a sparse disk file, which is a file where the space available is not taken up until it is actually used. The advantage of a sparse disk file over a preallocated disk file is that you can make more efficient use of physical storage by making virtual storage look large but delaying physical allocation until it is needed. Also, the sparse storage volume can be created much more quickly. An example of sparse file creation is shown here:

```
# dd if=/dev/zero of=sparse_file bs=1k seek=4096k count=1
1+0 records in
1+0 records out
1024 bytes (1.0 kB) copied, 5.7697e-05 seconds, 17.7 MB/s
# ls -lh sparse_file
-rw-r--r-- 1 root root 4.1G May 31 12:26 sparse_file
# du sparse_file
16      sparse_file
```

The input file option value feeds `dd` from `/dev/zero`, which is a special file with many nulls. The `bs` option creates the file with 1 K blocks. The seek option tells `dd` to skip 4,096×1,024 bytes before writing anything. The `count` option writes one block. The result is that the `ls` command sees the file as 4.1 G in size, although it is really only 16 K in size, as shown by the disk usage command `du`.

The `losetup` command creates and manages loop devices. Linux systems usually come preconfigured with a small number of loop devices. You can check for the next one available with the `losetup` command with the `-f` option:

```
# /sbin/losetup -f
/dev/loop0
```

This shows that the next available loop device is `/dev/loop0`. We can map our sparse file to the loop device:

```
# /sbin/losetup /dev/loop0 sparse_file
```

After creating the device, you can use `fdisk` to partition it and use `kpartx` to make the partitions available to the system. This is useful if you plan to use the device to create a virtual machine image. The partitions can be used to create a volume group with the `vgcreate` command and a logical volume with the `lvcreate` command. The `mkfs` command can also be used to create a file system on the device. This is discussed in the upcoming section, "File Systems."

For details on how to use the Linux Logical Volume Manager on the IBM SmartCloud Enterprise, see the article "IBM SmartCloud Enterprise Tip: Configure the Linux Logical Volume Manager" [Criveti, 2011].

File-Based Storage

Linux and UNIX systems present the file system as one large tree. The `mount` command attaches devices to this tree. The basic form of the mount command is

```
# mount -t type device dir
```

Here, `type` is the type of file system, `device` is the device to be attached, and `dir` is the directory to mount the file system as. The `mount` command without any arguments lists all the devices mounted to the file system. The file `/etc/fstab` contains the default locations of device mount points and is read by the operating system during startup. It is also consulted when mounting a removable device such as a CD in a standard location. A common operation in virtual environments is to store ISO image as files so that they can moved easily and then mount them with the `mount` command to look in an analogous way as a CD is mounted when access to the data stored in the ISO image is needed. This can be done with the following commands:

```
# mkdir /mnt/cdrom
# mount -t iso9660 -o loop myimage.iso /mnt/cdrom
```

The `-o loop` option indicates that it is a loop device—that is, the device is a file itself. The loop device features is a basic element of the Linux system that enables virtual machine images to be stored as files, manipulated using the file system, and then run when needed.

You can also mount a file as if it were a floppy with this command:

```
# mkdir /mnt/floppy
# mount -t msdos -o loop myimage.img /mnt/floppy
```

For DVD options, us the `udf` file type.

To mount a shared file system from a remote server using NFS, use these commands:

```
# mkdir /mnt/share
# mount hostname:/mountpoint /mnt/share
```

`hostname` is the remote server DNS name, and `mountpoint` is the remote directory to mount.

NFS runs a daemon process called `nfsd` to make data available to other computers. It relies on the Linux remote process call (RPC) library that allows C programs to make procedure calls on other machines across the network. The files to export are listed in the file `/etc/exports` using the `exportfs` command. The `mountd` daemon processes mount requests from NFS clients. The `portmap` daemon is also used to map RPC program numbers to DARPA port numbers and the `lockd` NFS lock manager. To share files on an instance using NFS, create firewall rules for `portmap` (port 111), `nfsd` (port 2049), `mountd` (port 1011), and `lockd` (port 35000). See the "Firewalls" section in Chapter 7 for instructions on how to do this. Add entries for `mountd` and `lockd` in the `/etc/services` file. See the earlier "Services" section for instruction on how to do that. Start the `nfsd` and `mountd` daemons with these commands:

```
# /usr/sbin/rpc.nfsd
# /usr/sbin/rpc.mountd
```

Create the directory `~idcuser/myshare` and edit or create the `/etc/exports` file with the line shown here:

```
/home/idcuser/myshare ip(rw,no_root_squash,async)
```

`ip` is the IP address or range addresses of the client(s) you need to allow to mount the export. Use a slash (`/`) to create ranges, as in `9.200.7.171/175`. Use `*` for any IP address (a bad option in the cloud). Start the NFS server with these commands:

```
# service portmap start
# service nfs start
```

Now you can mount your remote file system on the client computer using the `mount` command.

When troubleshooting NFS problems, you can check the output of the `rpcinfo` command, as shown here:

```
# /usr/sbin/rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100021    1   udp  59880  nlockmgr
    100021    3   udp  59880  nlockmgr
    100021    4   udp  59880  nlockmgr
    100021    1   tcp  43109  nlockmgr
    100021    3   tcp  43109  nlockmgr
    100021    4   tcp  43109  nlockmgr
```

The result should show entries for `portmapper` and `nlockmgr` (`statd`) for a client. If you do not see, try restarting `nfslock` with this command:

```
# /sbin/service nfslock start
```

NFS can be relatively complex to operate and troubleshoot, especially with SELinux. It transfers files without encryption, which can be an issue in a cloud environment. If you just need to transfer files in a limited number of situations, consider using SCP, as described in the "Secure Shell (SSH)" section in Chapter 7.

File Systems

File systems have been thought of as a method of organizing files. However, you saw that the Network File System is really a protocol for connecting to a remote file system; that could be ext3 or something else. From that example, it is apparent that the underlying storage system does not always matter. This section discusses some aspects of file systems that are relevant to cloud computing. This section mostly applies to Linux, but the concepts are similar on Windows.

We can divide file systems into three basic types:

- Local file systems
- Shared or network file systems
- Distributed file systems to support network storage systems

A local file system is a fundamental element of operating systems. Local file systems have **inodes** that point to the locations where **data blocks** containing data are stored. The inodes also contain data about files, such as file name, owner information, size, last modified time, and access permissions. To share files among multiple computer systems, you use a **network file system**, such as NTFS. To store large amounts of data in a central location you use network file systems. A **distributed file system** supports files stored over multiple nodes. An example is the Hadoop File System (HDFS).

Linux supports many different file systems. These are the most interesting to us:

- **ext3**—The standard file system for local storage volumes.
- **ext4**—The successor to ext3.
- **NTFS**—A Windows file system that can also be used locally on Linux.
- **proc file system**—A special file system that stores system-related data.
- **Network File System (NFS)**—A protocol for sharing files remotely on Linux and UNIX systems.
- **Common Internet File System (CIFS)**—Previously known as Server Message Block (SMB). A Microsoft protocol for sharing files remotely across systems. It works on Linux as well.
- **ISO file system**—Used for optical drives.

You can find out what types of file systems are used using the `mount` command without any arguments—for example:

```
# mount
/dev/vda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/vda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

This shows that the `ext3`, `proc`, `sysfs`, `devpts`, `tmpfs`, and `rpc_pipefs` file systems are in use. The `ext3` file system is used to store files on disks and is the file system we are most interested in. The others are virtual file systems have specialized purposes. The `tmpfs` file system is based on RAM for fast access. The `proc` file system is used to store system information. The `sysfs` file system is used to store device information.

You can find out about the disk space used by file systems on your system with the `df -T` command, as shown:

```
#df -T
Filesystem    Type   1K-blocks      Used Available Use% Mounted on
/dev/vda2     ext3    61824020   6260048  53051528  11% /
/dev/vda1     ext3      101086     11919     83948  13% /boot
tmpfs        tmpfs     2022028         0   2022028   0% /dev/shm
```

The `-T` option adds file system type information to the `df` command. The output shows that `ext3` is used for the `/dev/vda2` and `/dev/vda1` file systems, and that the `tmpfs` type is used for `tmpfs`.

The ext3 file system is the default file system used by many popular Linux distributions, including the Red Hat Enterprise Linux and SUSE distributions on IBM SmartCloud Enterprise. It is a journaled file system, which means that it keeps track of changes in a journal (a log) before committing the changes. In the event of a power outage or sudden shutdown from some other cause, the files can be more easily recovered than a file system without journaling.

The ext3 file system has a maximum size, depending on its block size. For a block size of 1 kibibyte (KiB), the maximum file system size is 1 tebibyte (TiB); with a block size of 4 KiB, the maximum file system size is 16 TiB. It can support a maximum file size from 16 GiB (1 KiB block size) to 2 GiB (4 KiB block size). You can create an ext3 file system with the Linux `mke2fs` or `mkfs.ext3` command and find out about a file system with the `dumpe2fs` command. To create an ext3 file system on the loop device you created earlier, use the `mkfs.ext3` command, shown here:

```
# /sbin/mkfs.ext3 /dev/loop0
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
524288 inodes, 1048576 blocks
52428 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1073741824
32 block groups
32768 blocks per group, 32768 fragments per group
16384 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Converting from Old Units of Measure to New

Here's a quick reference for the conversion from the old units of measure to the new:

New unit	No. bytes	Old unit	No. bytes
kibibyte (KiB)	1,024 bytes or 2^{10}	kilobyte (KB)	1,000 bytes or 10^3
mebibyte (MiB)	1,048,576 or 2^{20}	Megabyte (MB)	10^6
gibabyte (GiB)	2^{30}	Gigabyte (GB)	10^9
tebibyte (TiB)	2^{40}	Terabyte (TB)	10^{12}

The command creates the inode tables, file system accounting information, and journal. The command output shows the details of the file system created, including the number of blocks, number of inodes, and other information. You can mount the loop device just like a physical device, except that you need to use the `lomount` command instead of the regular `mount` command.

The ext4 file system, the successor to ext3, was added to the Linux kernel 2.6.28 in 2008. It can handle volumes up to 1 exabyte in size and files up to 16 TiB in size. It improves performance for large file sizes and reduces fragmentation.

The native file system for Windows is NTFS. Files are shared across Windows systems using the Common Internet File System (CIFS), previously known as the Server Message Block (SMB). You can interoperate with CIFS on Linux systems using Samba, an open source implementation of the CIFS protocol. In a cloud environment, you might prefer to restrict use of CIFS to VPN access only, for security reasons. You have ways to share files across remote systems securely using SSH, for example, with SFTP and the SSH file system (SSHFS).

Virtual Machine Disk (VMDK) is a documented file format developed by VMWare for virtual machines. It is supported by the various VMWare products and by both third parties and open source projects, such as QEMU. Virtual Machine Disk files are given a `.vmdk` extension by convention and are often included in virtual machine packages, such as Open Virtualization Format.

Network Storage Systems

Network storage systems were developed to support the exploding data storage requirements of enterprises as the IT industry has developed over the last 30 years. They emerged as an evolution of shared file systems. Network storage systems are essential to cloud computing infrastructure to provide sufficient storage for the combined needs of many users. Network storage systems can provide both block-based and file-based storage services. Some of the challenges that network storage systems address are listed here:

- Backing up data from applications, such as email systems and financial systems.
- Helping to manage different classes of data in an organized way. Data can be active or inactive, but it may be hard to know if or when inactive data will be needed.
- Helping users store their data so that they are not solely dependent on personal workstations or laptops.
- Providing scalable solutions for applications that use huge amounts of data, such as data warehouses.
- Providing virtualization of storage so that users do not need to know or care how or where data is stored.

Previously, network storage was a tool that large enterprises and governments used because it was expensive to buy and difficult to implement, requiring specialized skills to install and configure. One of the great things about cloud computing is that it has expanded the influence of network storage beyond large enterprises and makes it available to us all.

A **storage area network** (SAN) is a dedicated network that provides access to block-level storage. Storage area networks can be used to create storage devices that appear locally attached to the operating system. They are often used to support applications such as data mining that need rapid access to large amounts of data.

Several network-based storage systems work at the block level. ATA over Ethernet (AoE) is a network-based storage system designed for use over local area networking using an Ethernet protocol. It has no security built into it and can allow anyone on the same logical switch to access AoE drives exported by the server.

An iSCSI (Internet Small Computer Interface) network storage system operates at the network layer, such as TCP, and can potentially work over a wide area network. It works at the block level and supports authentication of a client to the server for security. It exports a SCSI drive over a network as if it is were a normal SCSI drive. An iSCSI server is called a target. There are open source and commercial implementations of iSCSI targets, as well as hardware implementations, such as IBM's DS family of storage servers.

Network attached storage can also be implemented with Fibre Channel devices using a host bus adapter (Fibre Channel HBA). The use of optic fiber components greatly increases the performance of this class of storage device.

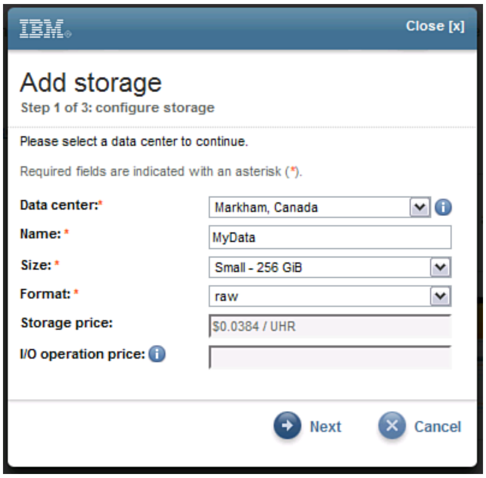
Structured Storage

The file systems we discussed earlier are often referred to as supporting **unstructured data** storage—that is, you have no idea what data is in the files. With **structured data,** you have visibility into the content of the data. Usually it is stored in some kind of database, relational or otherwise. The main method of accessing relational databases in Infrastructure as a Service clouds is via virtual machine images. Another approach is to provide a central relational database service that is shared among multiple tenants; this is a Platform as a Service option.

An alternative to relational databases are other database management systems, such as NoSQL, that do not maintain relational integrity. These do not require fixed schemas, avoid join operations, and do not provide ACID (atomicity, consistency, isolation, durability) guarantees. They basically work as key–value stores, and their attractiveness results from their low-cost way of storing, indexing, and searching in a scalable and fault-tolerant way. NoSQL databases have been used for storing data for large social networking sites where the web sites have provided free services to huge user populations.

Managing Storage on IBM SmartCloud Enterprise

Block storage volumes can be provisioned with IBM SmartCloud Enterprise both by the user interface and via the web user interface. We demonstrated provisioning storage using the command line and API in Chapter 3, "Developing with IBM SmartCloud Enterprise APIs." In these examples, the storage volume options were ext3 format with a mount point provided. It is also possible to provision raw storage volumes, as shown in Figure 6.13.



Add storage

Step 1 of 3: configure storage

Please select a data center to continue.

Required fields are indicated with an asterisk (*).

Data center:*	Markham, Canada
Name: *	MyData
Size: *	Small - 256 GiB
Format: *	raw
Storage price:	$0.0384 / UHR
I/O operation price:	

Figure 6.13 Provisioning a raw storage volume in the IBM SmartCloud Enterprise

You need to wait for a few minutes before you can use the storage volume in a virtual machine instance provisioning request. When status has changed from New to Nonattached, you can use it. After you have created a new virtual machine instance with the disk attached, you need to format and mount it using the tools already discussed.

Mounting a storage volume on Windows is similar in principle to doing so on Linux. However, it is usually done via the graphical user interface. After a storage volume is provisioned on the IBM SmartCloud Enterprise, it can be added to Windows virtual machines during provisioning. At that point, the storage volume is attached but not mounted and is still unformatted. After the virtual machine has booted up, you must mount and format the volume using the Storage Disk Management Wizard on Windows 2008 Server.

Remote Desktop Management

The section "Desktop Virtualization" in Chapter 1, "Infrastructure as a Service Cloud Concepts," briefly discussed the main concepts of this topic, and the section "Linux, Apache, MySQL, and PHP" in Chapter 2, "Developing on the Cloud," gave an example of using VNC on Red Hat Enterprise Linux. This section gives more details on the tools available on typical cloud images.

X Windows

X Windows, or just X, is the basis for most Linux remote display tools. Even for VNC, which works at a lower level, X Windows is still involved. It provides the basic primitives for rendering and interacting with graphical user interface environments and fonts on Linux and UNIX. Certain basic concepts are important for orientation and troubleshooting. We covered some general background material on the various open source projects behind X Windows in the introductory section "Desktop Virtualization" in Chapter 1. Here we discuss practical use. In addition to its use for individual work, X Windows is the basis for a number of workstation, server, and device display products.

X uses a client/server architecture that allows it to be used over the network. The X Session Manager manages desktop sessions and can save and restore the state of running applications. A protocol called the X Session Management Protocol defines how the session manager interacts with applications. In particular, the Window Manager interacts with the X Session Manager for the existence and placement of windows and icons. Different implementations of X Session Managers exist, including the default session manager xsm and the KDE session manager kmsserver. Different implementations of window managers also exist, such as metacity or GNOME and KWin for KDE. The X Display Manager facilitates the starting of sessions on local or remote computers using the X Display Manager Control Protocol. An X server runs on the computer that provides the display and graphical input devices. After it starts, it connects to the display manager. In a typical scenario of a local connecting from a local workstation to a virtual machine on the cloud, the local workstation runs the X server and the virtual machine runs the display manager. Besides the original X Display Manager (xdm), other implementations include gdm for GNOME and KDM for KDE.

An X system can support many displays, local and remote. The display is known from the `DISPLAY` variable. To find the value of your display variable, type this command:

```
$ echo $DISPLAY
:0.0
```

This indicates that the display is 0.0. To set the display, use the `bash` command:

```
$ export DISPLAY=host:n.n
```

You can omit `host` if you mean localhost, and `n.n` is the display number. You do not always need the part after the decimal, making the short form possible:

```
$ export DISPLAY=:0
```

This means display 0 on localhost. If you do not have the `DISPLAY` variable set, you will get errors trying to open graphical applications. For Cygwin, you can find the `DISPLAY` value by hovering over the tray icon.

The X Windows authorization system enables remote use and also local use with multiple users in a safe way. Access can be based on host, cookie, or user. Cookies are created and stored in a `.Xauthority` file in the user's home directory. If you get errors because this file is not writeable, you can use the `xauth` command `or`. This sometimes happens when you try to start up multiple sessions and the home directory of the root user has not been set properly. The `xauth` command is used to create and edit authorization information for connecting to the X server. You can also use it to break any locks with this form:

```
$ xauth -b quit
```

Alternatively, delete the file `.Xauthority` file to solve the problem.

The `xhost` command adds or deletes a host from being able to connect to your X server—for example:

```
$ xhost + remote_host
```

After logging into `remote_host`, you are authorized to connect back to the X server where you issued the `xhost` command.

Xterm is a terminal emulator for X that is commonly used to start a command line on a system that uses X.

By default, many of the images on the IBM SmartCloud Enterprise are configured to run at run level 3 by default. This means that the X services, which operate at run level 5, are not started. That does not prevent you from using VNC or NX, as described shortly, but it can prevent you from doing port forwarding to X services. To change the default run level, edit the file `/etc/inittab`, changing the line shown here:

```
id:5:initdefault:
```

Then restart the service and bring up the system to run level 5 with the following commands:

```
# /sbin/service xinetd restart
# /sbin/init 5
```

Virtual Network Computing (VNC)

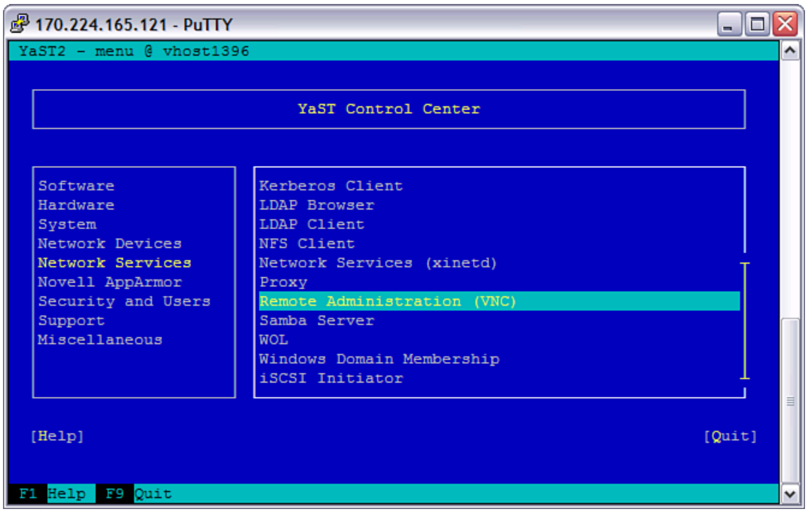
Virtual Network Computing (VNC) is bundled on many cloud images and can be freely and easily installed on those it is not included with. Users need to download a VNC viewer to use VNC from a client computer. You can freely download a VNC from the RealVNC web site. For performance reasons, the native VNC view is preferred over the Java applet loaded via a web page.

After installation, you must open a port in the firewall. By default, VNC uses TCP port 5900+*N*. *N* is the display number, :1 for the first remote display, :2 for the second, and so on. When you start up the vncserver, it tells you what display it is using. There is also a service over HTTP on port 5800+*N* to serve a VNC viewer in a Java applet.

On SUSE, start the YaST administration tool using the `yast` command:

```
# yast
```

Select Network Services, Remote Administration (VNC), as shown in Figure 6.14.



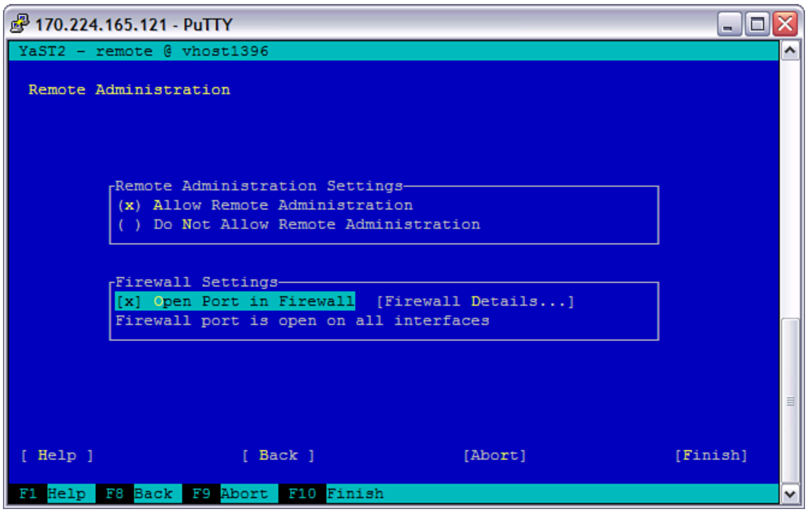
```
170.224.165.121 - PuTTY
YaST2 - menu @ vhost1396

                     YaST Control Center

Software              Kerberos Client
Hardware              LDAP Browser
System                LDAP Client
Network Devices       NFS Client
Network Services      Network Services (xinetd)
Novell AppArmor       Proxy
Security and Users    Remote Administration (VNC)
Support               Samba Server
Miscellaneous         WOL
                      Windows Domain Membership
                      iSCSI Initiator

[Help]                                              [Quit]

F1 Help  F9 Quit
```

Figure 6.14 SUSE YaST administration tool

In the next dialog box, select Allow Remote Administration and Open Port in Firewall, as shown in Figure 6.15.

The changes should take effect immediately. If you have connection problems, check that the firewall port for 5900+*N* is open, where *N* is the display number.

By default, the RFB protocol that VNC uses is not secure. However, it can also be tunneled over SSH. The open source ssvnc client does this. It requires an x11vnc server on the virtual machine. See the article “Secure Multi-user Access to IBM Cloud Instances with VNC and SSH” [Hudacek, 2011] for details on doing this. Another option to tunnel VNC over SSH is to use port forwarding. This is explained in the section “Port Forwarding” in Chapter 7.



```
170.224.165.121 - PuTTY
YaST2 - remote @ vhost1396

 Remote Administration


          ┌Remote Administration Settings────────────────────────┐
          │(x) Allow Remote Administration                       │
          │( ) Do Not Allow Remote Administration                │
          └──────────────────────────────────────────────────────┘

          ┌Firewall Settings─────────────────────────────────────┐
          │[x] Open Port in Firewall  [Firewall Details...]      │
          │Firewall port is open on all interfaces               │
          └──────────────────────────────────────────────────────┘


[ Help ]              [ Back ]              [Abort]              [Finish]

F1 Help  F8 Back  F9 Abort  F10 Finish
```

Figure 6.15 YaST Remote Administration (VNC) dialog box

NX Remote Desktop

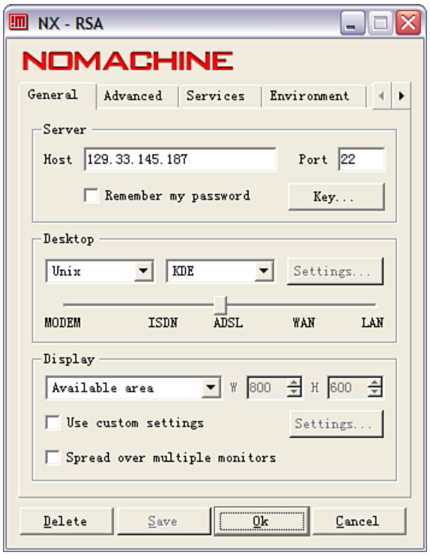
The NX remote desktop technology relies on an NX server and an NX client. You used the NX desktop in the section "Rational Application Developer" in Chapter 2. The NX server is available commercially from NoMachine or open source distributions. Some images in the IBM SmartCloud Enterprise uses the FreeNX open source package. The NX server is responsible for user authentication and session activation. A third component is the NX node. `nxservice` and related commands are not daemons but, rather are activated by the SSH server.

Configuration files are usually in the directory `/etc/nxserver`, which also includes the keys needed for authentication. If you have trouble, a useful setting is `NX_LOG_LEVEL`, in the `node.cfg` file. You can set it to 4 for server/client communication. The location of the log file is also configured in `node.cfg`, with a default setting of `/var/log/nxserver.log`.

You can freely download the NX client from the NoMachine web site. After installing, import and save your SSH key (unconverted by Putty) and set the IP or host name of your virtual machine in the Server field. Figure 6.16 shows the configuration screen.

The client configuration user interface enables you to choose a desktop, such as KDE or Gnome. This can be a source of errors, so try a different value if you have trouble establishing a session after authentication. The client configuration and session files are stored in the user's `.nx` directory (`C:\Documents and Settings\Administrator\.nx` on Windows). In case of errors, check the session directory, also under `.nx`.

See the "Secure Shell (SSH)" section in Chapter 7 for more on SSH.



Server
Host 129.33.145.187 Port 22
Remember my password

Desktop
Unix KDE
MODEM ISDN ADSL WAN LAN

Display
Available area W 800 H 600
Use custom settings
Spread over multiple monitors

Figure 6.16 NX Client Connection Wizard

Composite Applications

In some cases, you might need to create a composite application that includes a group of systems—for example, a web application with a web server, an application server, and a relational database. In the IBM Cloud catalog, a number of applications fall into this category, such as IBM Tivoli Monitoring (ITM) and Rational Team Concert (RTC). One way to do this is on a large machine with many CPUs. The IBM cloud supports virtual machines with up to 32 CPUs. The main task you face when creating a composite application on a single instance is installing all the different software items and managing the licenses. Software bundles, described in the earlier section, "Software Installation and Management," can help with this.

You might also need to create a group on different machines. You might find images with the software you need already installed. Several tasks in creating a group of different instances might be relevant to your particular application:

- Configuration of the topology
- Startup sequence
- Runtime parameters, such as system-to-system credentials

Email

Email is a basic utility that many applications use. For example, IBM Rational Team Concert uses email to invite team members to join a project. Open source mail servers can be used in your cloud applications, including Sendmail, qmail, and Apache James. Most application programming languages include a library that you can use in the applications you develop, for interfacing with a mail server to send and manage mail. In Java, this is provided by JavaMail. Your basic challenges in email enabling a cloud application are to find or install a mail server, configure it for relaying, and configure your software application to make use of it.

Mail is most often sent using the Simple Mail Transport Protocol (SMTP) and is retrieved by mail clients from their local mail servers with Post Office Protocol (POP) or Internet Mail Access Protocol (IMAP). For basic email enabling of an application, you must configure SMTP but probably not POP. SMTP relaying is the process of sending a request to a mail server to send that mail to an address on a different server. You need to do this when you connect a mail-enabled application. Because of problems with spam and mail abuse, mail servers tightly manage relaying, which provides a challenge for legitimate use. Out of the box, the Sendmail installation on Red Hat Enterprise Linux allows relaying only from localhost.

Setting up an SMTP Server

Let's start by setting up an SMTP server to send mail from a cloud application. The Red Hat Enterprise Linux images shipped in the IBM SmartCloud Enterprise include the SMTP server Sendmail. Let's look at that as an example. Check the status of the mail server with this command:

```
# /sbin/service sendmail status
sendmail is stopped
```

This tells you that the mail server is not running. Confirm the fully qualified domain name of your virtual machine with the `nslookup` command, as described in the earlier section "Basic Network Settings." If it looks okay, try starting the Sendmail service with this command:

```
# /sbin/service sendmail start
Starting sendmail:                         [  OK  ]
Starting sm-client:                        [  OK  ]
```

The output tells you that the mail server started successfully. You can check the ports and IP addresses that Sendmail is listening on with this command:

```
# netstat -nl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address          State
…
tcp        0      0 127.0.0.1:25        0.0.0.0:*                LISTEN
```

This tells you that Sendmail is listening for connections from localhost on port 25, the standard SMTP port. If your application is on another server, you must change the `DaemonPort Options` setting in the Sendmail configuration file at `/etc/mail/sendmail.cf`. Next, add the aliases for your server in the file `/etc/mail/local-host-names`. Add your server name, not just the domain name. Remember that many other cloud users share the same domain name. If your application is not on the local machine, you also need to add your application server host name to the access database to allow relaying. To do this, edit the file `/etc/mail/access.db`, add your application server host name, and execute the command `make access.db`. Restart Sendmail with this command:

```
# /sbin/service sendmail restart
Shutting down sm-client:                       [  OK  ]
Shutting down sendmail:                        [  OK  ]
Starting sendmail:                             [  OK  ]
Starting sm-client:                            [  OK  ]
```

Now verify your setup by trying the SMTP service from the command line, as shown here:

```
$ mail a@example.com
Subject: testing
hello
.
Cc:
```

Do this from the idcuser account, not from root. Enter your own email address instead of the address given, and check that you receive the email. End the message with a single period on the last line.

For troubleshooting, see the log file at `/var/log/maillog`.

Software as a Service

Software as a Service (SaaS) originated before Infrastructure as a Service and the term *cloud* became popular. During the initial period, SaaS offerings were built directly on physical hardware and required large capital investments. Salesforce is the prime example of SaaS. It began as a customer relationship application and has now grown to include a wide range of business applications and a Platform as a Service, force.com. The capital investment required for SaaS is now much reduced by running SaaS on top of cloud infrastructure.

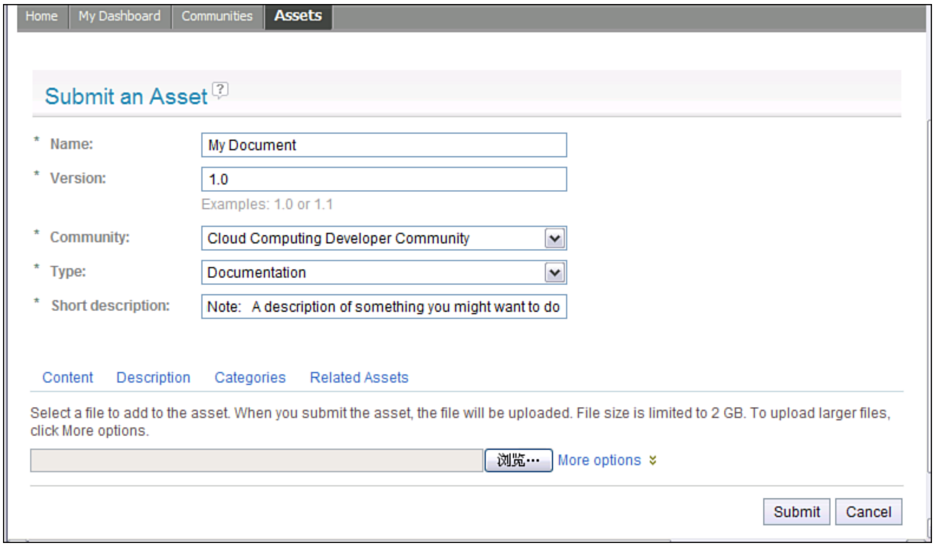
Document-Management Systems

A document-management system manages files so that they can be stored in a central location for convenient access, indexing, and searching. Modern document-management systems enable users to add their own comments about the documents and have Web 2.0 features such as tagging and ratings. They support role-based access management in managing the documents, and you can place workflows around documents to ensure that they are reviewed and approved before publishing.

In the IBM SmartCloud Enterprise, document management is done by IBM Rational Asset Manager (RAM). This is especially appropriate because most of the documents in the cloud are virtual machine images, which themselves are software assets. RAM is a standard IBM product that you can use also in your own projects. The image catalog has an entry for RAM that enables you to start up your own server.

All the images in the catalog of the SmartCloud Enterprise have image assets that have public visibility. When you save a virtual machine as an image in the SmartCloud Enterprise, an image asset is created. You can add documents, such as help in getting started and screen shots, to aid your image users. RAM also supports a forum where your users can ask questions. Initially, your image asset has private visibility—that is, it will be visible only to you. However, you can also make it visible to the other members of your enterprise.

RAM supports other types of documents as well, such as text documents. Most of the support documents for the SmartCloud Enterprise are stored as RAM documents assets. When creating a new document in RAM, you will see the screen in Figure 6.17.



* Name:	My Document
* Version:	1.0
	Examples: 1.0 or 1.1
* Community:	Cloud Computing Developer Community
* Type:	Documentation
* Short description:	Note: A description of something you might want to do

Select a file to add to the asset. When you submit the asset, the file will be uploaded. File size is limited to 2 GB. To upload larger files, click More options.

Figure 6.17 Creating a document in RAM

You can add a name, a short description, and a long description and attach files. You can configure an approval workflow to automatically be enforced before the document is made visible to others. Figure 6.18 shows an example step in the approval workflow.

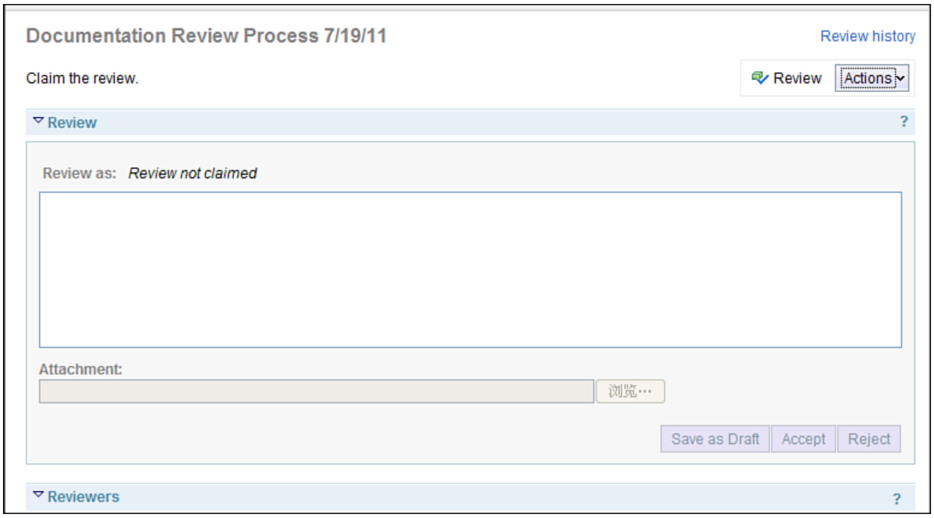


Figure 6.18 Review of a document submitted to RAM

In this case, there are two steps: claim review and then accept/reject the document.

Email and Collaboration Suites

Cloud-based collaboration tools are useful in project environments where team members change dynamically and include members outside your own organization. These tools include chat, wikis, web conferences, file sharing, and email. Using cloud-based collaboration tools, you remove the burden of managing the infrastructure to support them from your own IT organization. You easily give access to people who need access without giving access to your organization's private network. In fact, establishing email services is not cheap and convenient enough to do on a project basis if you needed to, say, because the project spans multiple enterprises.

IBM SmartCloud for Social Business is a cloud-based collaboration suite from IBM that includes email and collaboration tools. IBM SmartCloud for Social Business also includes extension points for third-party applications. Two patterns for integration exist:

- **Outbound integration**—Integrate other applications into IBM SmartCloud for Social Business
- **Inbound integration**—Integrate IBM SmartCloud for Social Business into other services

IBM SmartCloud for Social Business allows single sign-on using Security Access Markup Language (SAML) or OAuth. See the section on OAuth in Chapter 7 for more details on that approach.

Business Scenario: The IoT Data Application

We decided on the IoT Data application architecture early in the book before we discussed the different options available on the cloud. Now we can discuss the reasons for the choices.

For our IoT Data scenario, we create our own custom images with Hadoop installed. Then we use the IBM SmartCloud Enterprise REST Java client to create virtual machine instances. We will not provide these images to outside users, so we will leave visibility as private. We create start-up scripts to automatically start Hadoop when the operating system is booted and the network becomes available. We discussed start-up scripts in the earlier section "Linux Services."

IoT Data makes heavy use of storage services. Before creating the virtual machine instances, we create storage volumes with the REST API Java client. The storage volumes are provided as block storage, but we select ext3 as the format when they are provisioned. This acts as local file system. We attach a storage volume with each virtual machine instance. We use the Hadoop Distributed File System (HDFS) to span multiple nodes and scale to a large number of files. We need additional information about our files, especially on the customer who owns the files. We use Hadoop to store this metadata.

We do not need a software installation or management services at this point. When new updates to Hadoop become available, we will re-evaluate this. We also have no need for remote desktop management at this point. We do make use of email to inform users of the completion of long-running jobs in Hadoop. We use the Sendmail system provided with RHEL for this.

One important problem is that, when bringing up a new Hadoop node, we need to join it to our cluster. Our plan for this is to upload the SSH private key when creating the virtual machine. We demonstrated how to do this in the section "Business Scenario: File Uploads" in Chapter 4, "Standards." Then we can use the IP information from SmartCloud Enterprise to join the new node to the cluster.

Our IoT Data application is an example of a composite application. It consists of three parts:

- The management application, running as a J2EE application on WebSphere
- The relational database, used to store customer and device information
- The Hadoop cluster

Our main need for automation is in scaling the file system by adding Hadoop nodes. We believe that a single management application and database will be sufficient, so we do not have an initial need to automate that using IaaS APIs. However, we still have performance, availability, and maintenance to consider, which are topics for Chapter 8, "Performance, Availability, Monitoring, and Metering."

CHAPTER 7

Security

Background

This section gives an overview of security considerations when developing a cloud-based application and highlights security differences between traditional software development and cloud-based software. The differences arise from several factors, which might or might not apply to the particular application you are developing:

- Exposure on the public Internet
- Multitenant awareness
- Automated, delegated, or self-service administration

One of the basics of being on the Internet is that your application must be network aware. Usually, that implies an HTML interface, but it can also be accomplished with a network-aware client application. Cloud applications should be built with the security implementations of exposure on the network and the greater surface of attack that a web user interface presents in mind.

A tenant is an organization that has many users consuming a service and that is isolated from other tenants on the same system. If you are building an application that is intended for use only within your organization, it does not need to be multitenant aware. The need for automated, delegated, or self-service administration stems from changes in user expectations to be able to instantly manage their own user accounts and also from the financial drive to reduce support center costs. Building enterprise applications that are multitenant aware has been a major trend in application development over the past five to ten years.

The Cloud Security Alliance outlines additional risks in its "Top Threats to Cloud Computing." Additional threats include these:

- Abuse of cloud computing services
- Insecure interfaces and APIs
- Malicious insiders
- Shared infrastructure
- Account hijacking

Actually, these either are included in the three differences between cloud and traditional computing here or are no different than traditional security threats. However, they are important issues to keep in mind. The Cloud Security Alliance also describes possible security benefits of cloud computing, which include these:

- Centralized data storage
- Use of standard images

In the context of security, **plain text** is data to be encrypted or data that has been decrypted, **cipher text** is the encrypted text, and a **cipher** is an algorithm for converting plain text to cipher text. A key is data that the cipher algorithm uses. A **symmetric key cipher** uses the same key to both encrypt and decrypt data. In this method, the key must be kept secret. An **asymmetric key cipher** uses one key to encrypt data and another to decrypt the data. This is the basis for public key cryptography. One of the keys, the **private key**, must be kept secret; the other key, the **public key**, can be shared with other parties. A mathematical relationship usually exists between the two keys, which requires a generation procedure. A **secure hash algorithm** provides an irreversible transformation of data to a unique value that can be used to validate the authenticity of the data. Secure hash algorithms are the basis for digital signatures.

Business Scenario: IoT Data Security Context

The IoT Data application supports multiple tenants and also has limited support for unregistered users. It operates in the public Internet. Figure 7.1 shows an infrastructure diagram for the network zones for the IoT Data business scenario.

We define three security zones:

- The Public Zone, which anyone on the Internet can access. It contains a welcome page.
- The Registered Users Zone, which only registered users can access. It contains the storage management component.
- The Administration Zone, which contains the Server Management component.

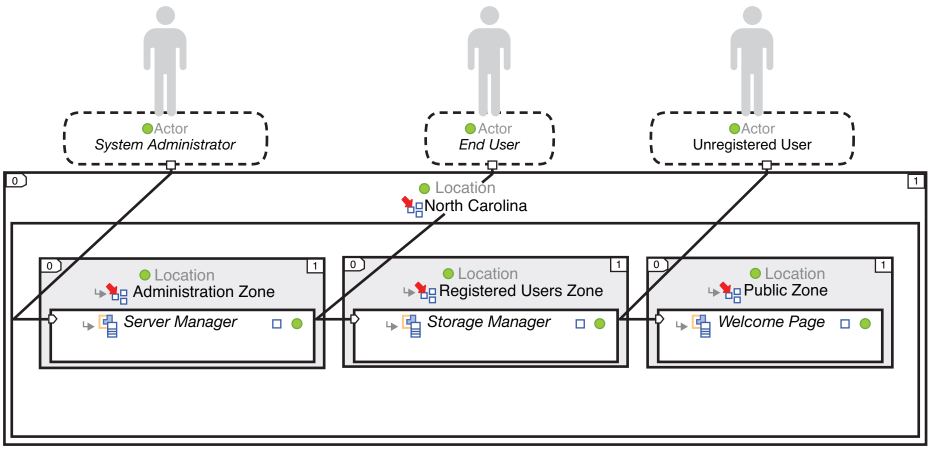


Figure 7.1 Infrastructure diagram for the IoT Data security zones

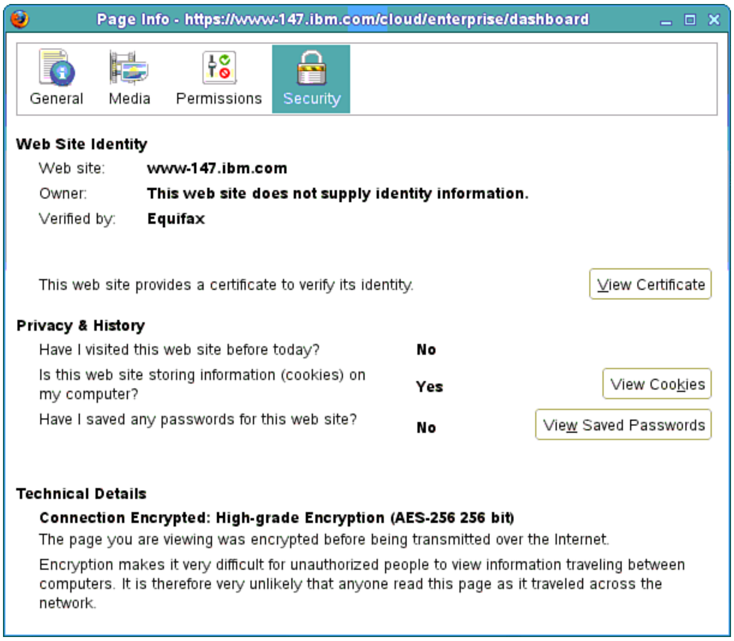
Public Key Infrastructures and Certificates

One of the first issues you will encounter enabling working on the cloud is being able to interoperate with the different protocols that are commonly used, such as SSH and HTTPS. Although these protocols are not specific to cloud computing, they are important building blocks for it. To be able to use these protocols, you need to deal with certificates. This is why we discuss keys and certificates first.

A public key infrastructure is one of the most important enabling elements of secure communications and e-commerce. Public key infrastructure uses the asynchronous key cipher cryptographic methods mentioned previously to guarantee that you can trust the identity of the party sending messages to you by checking that party's public certificate. The certificate should be signed by a certificate authority (CA) that you trust. In a more general case, a **trust chain** should consist of a chain of signatures finally leading to a certificate authority that you trust. The most commonly used standard for certificates is X.509.

The IBM SmartCloud Enterprise web site and the REST API service use HTTPS to transmit all data. That is backed by a certificate to verify its identity. The certificate is signed by Equifax, a well-known certificate authority that all major browsers recognize. You can find details about it in Firefox by clicking on the icon to the left of the address bar. Clicking the More Information button brings up the detail screen shown in Figure 7.2.

You can click the View Certificate button to see the contents on the certificate, as shown in Figure 7.3.


Page Info - https://www-147.ibm.com/cloud/enterprise/dashboard
General
Media
Permissions
Security

Web Site Identity

Web site:	www-147.ibm.com
Owner:	This web site does not supply identity information.
Verified by:	Equifax

This web site provides a certificate to verify its identity.

Privacy & History

Have I visited this web site before today?	No	
Is this web site storing information (cookies) on my computer?	Yes	View Cookies
Have I saved any passwords for this web site?	No	View Saved Passwords

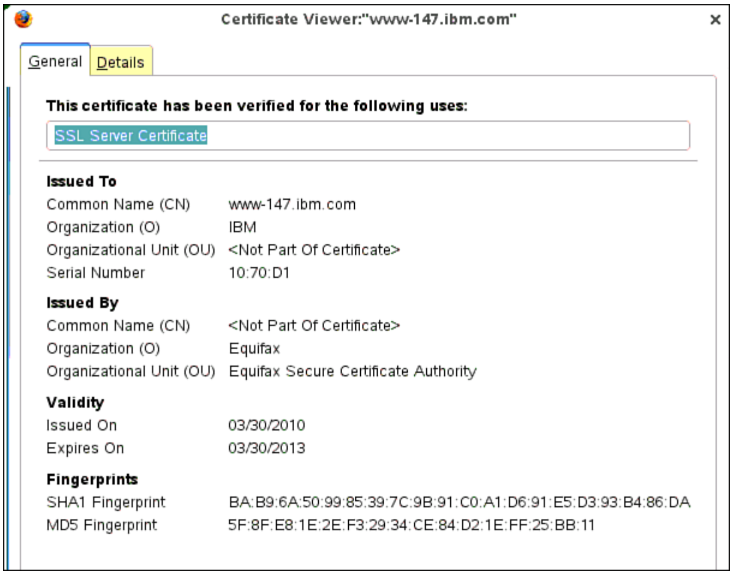
Technical Details

Connection Encrypted: High-grade Encryption (AES-256 256 bit)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it very difficult for unauthorized people to view information traveling between computers. It is therefore very unlikely that anyone read this page as it traveled across the network.

Figure 7.2 Firefox dialog box for security details for the IBM SmartCloud Enterprise web site



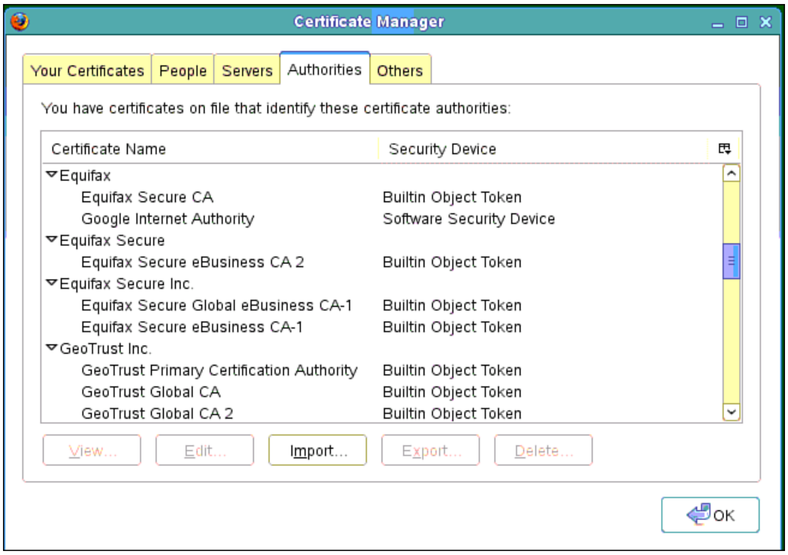
This certificate has been verified for the following uses:

SSL Server Certificate

Issued To	
Common Name (CN)	www-147.ibm.com
Organization (O)	IBM
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	10:70:D1
Issued By	
Common Name (CN)	<Not Part Of Certificate>
Organization (O)	Equifax
Organizational Unit (OU)	Equifax Secure Certificate Authority
Validity	
Issued On	03/30/2010
Expires On	03/30/2013
Fingerprints	
SHA1 Fingerprint	BA:B9:6A:50:99:85:39:7C:9B:91:C0:A1:D6:91:E5:D3:93:B4:86:DA
MD5 Fingerprint	5F:8F:E8:1E:2E:F3:29:34:CE:84:D2:1E:FF:25:BB:11

Figure 7.3 Firefox dialog of certificate contents

You can check your browser's trusted certificate authorities in Firefox under Tools, Preferences, Advanced, Encryption, Certificates, Authorities, as shown in Figure 7.4, and verify that Equifax is on the list. You can select it and export it, which will come in handy later when dealing with your own programs. Save it in `.pem` format.



You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
▾Equifax	
Equifax Secure CA	Builtin Object Token
Google Internet Authority	Software Security Device
▾Equifax Secure	
Equifax Secure eBusiness CA 2	Builtin Object Token
▾Equifax Secure Inc.	
Equifax Secure Global eBusiness CA-1	Builtin Object Token
Equifax Secure eBusiness CA-1	Builtin Object Token
▾GeoTrust Inc.	
GeoTrust Primary Certification Authority	Builtin Object Token
GeoTrust Global CA	Builtin Object Token
GeoTrust Global CA 2	Builtin Object Token

Figure 7.4 Trusted certificate authorities in Firefox

This explains how your browser trusts the IBM SmartCloud Enterprise. Whether a program that you might write trusts the certificate depends on language platform support and configuration.

In the Java platform the Key and Certificate Management Tool, `keytool`, manages cryptographic keys, certificate chains, and trusted certificates. It stores these in a **keystore**. A special file for the public certificates of certificate authorities, called `cacerts`, is located at `java.home\lib\security`. This includes the Equifax certificate, which you can check with this command on Windows and a similar command on Linux:

```
"%JAVA_HOME%\bin\keytool" -list -keystore
"%JAVA_HOME%\jre\lib\security\cacerts" -storepass changeit
```

In some environments, some applications might use the `jssecacerts` keystore. JSSE is the Java Secure Sockets Extension.

In your own applications, you can use your own certificates. You can do this with a certificate-generating tool such as OpenSSL. However, the certificates should still be signed by a trusted CA. Requesting that a trusted certificate authority sign your own certificate costs money.

You will need to do this for a production system but will probably prefer to avoid it in development projects. In this context, you can sign your own certificates. If you use a certificate that is not in the trusted keystore, you must add it yourself. To add a certificate to the trust store, use the `keytool` command, shown here:

```
keytool -import -noprompt -keystore jssecacerts -storepass changeit -
alias myca -file myca.pem
```

This uses the `import` option to add the certificate file to the `jssecacerts` keystore, opening the keystore with the password `changeit` and saving it with the alias `myca`. You can also interact with the keystore using the `java.security` API. The following example demonstrates looking up the key added using the previous command line.

```
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.cert.Certificate;
import java.security.KeyStore;

public class CertUtil {
            public void getCert(String alias) throws Exception {
            KeyStore ks =
KeyStore.getInstance(KeyStore.getDefaultType());
            char[] password = {'c', 'h', 'a', 'n', 'g', 'e', 'i', 't',
};
            InputStream is = new FileInputStream("jssecacerts");
            ks.load(is, password);
            Certificate c = ks.getCertificate(alias);
            System.out.println(alias + " is a " + c.getType());
      }

      public static void main(String[] args) throws Exception {
            CertUtil util = new CertUtil();
            util.getCert("myca");
      }
}
```

The program gets an instance of the `KeyStore` class and loads it using a `FileInputStream`. Then the program looks up the certificate using the alias that you used on the command line. Notice that the password uses a char array instead of a String. This is so that you can zero out the password when you are finished, which is something we neglected to do. The output of the command is shown here:

```
myca is a X.509
```

As expected, we found the certificate that we imported with the command line.

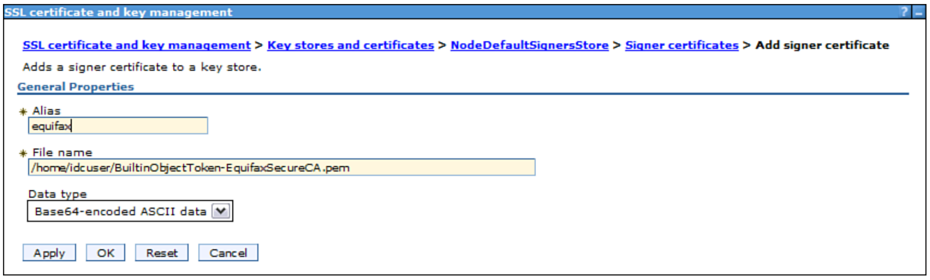
We demonstrate how to create self-signed certificates for setting up HTTPS in the section "Example: Setting up HTTPS on the IBM HTTP Server," later in the chapter.

Example: Trusted Certificate Signing Authorities in WebSphere Application Server

Java 2 Enterprise servers, such as WebSphere Application Server (WAS), build on the capabilities of the Java Standard Edition, including support of different key stores for incoming versus outgoing connections. WAS does not have the Equifax certificate in the trust store by default, creating a challenge that this example solves. If you try to call the IBM SmartCloud Enterprise APIs from within a WAS server, you get an error similar to this:

```
Error: com.ibm.jsse2.util.g: PKIX path building failed:
java.security.cert.CertPathBuilderException: PKIXCertPathBuilderImpl
could not build a valid CertPath.; internal cause is:
      java.security.cert.CertPathValidatorException: The certificate
issued by OU=Equifax Secure Certificate Authority, O=Equifax, C=US is
not trusted; internal cause is:
      java.security.cert.CertPathValidatorException: Certificate
chaining error
```

This is as designed. The error message tells you that the certificate could not be trusted because it could not be traced back to a trusted signing authority. To add Equifax as a trusted signing authority, you can use the Equifax Secure certificate exported from Firefox and the WAS server from the "Java 2 Enterprise Edition" section in Chapter 2, "Developing on the Cloud." After you export the certificate from Firefox, as described earlier, copy it to the WAS server using WinSCP. Log into the WAS administration console and navigate to the SSL Certificate and Key Management menu. Select Manage Endpoint Security Configurations, BaseAppSrvNode, Key Stores and Certificates, NodeDefaultTrustStore, Signer Certificates, and click Add. Enter the path to the certificate on the server and enter an alias for the certificate—say, equifax—as shown in Figure 7.5.


SSL certificate and key management
SSL certificate and key management > Key stores and certificates > NodeDefaultSignersStore > Signer certificates > Add signer certificate
General Properties
Apply
OK
Reset
Cancel

Adds a signer certificate to a key store.

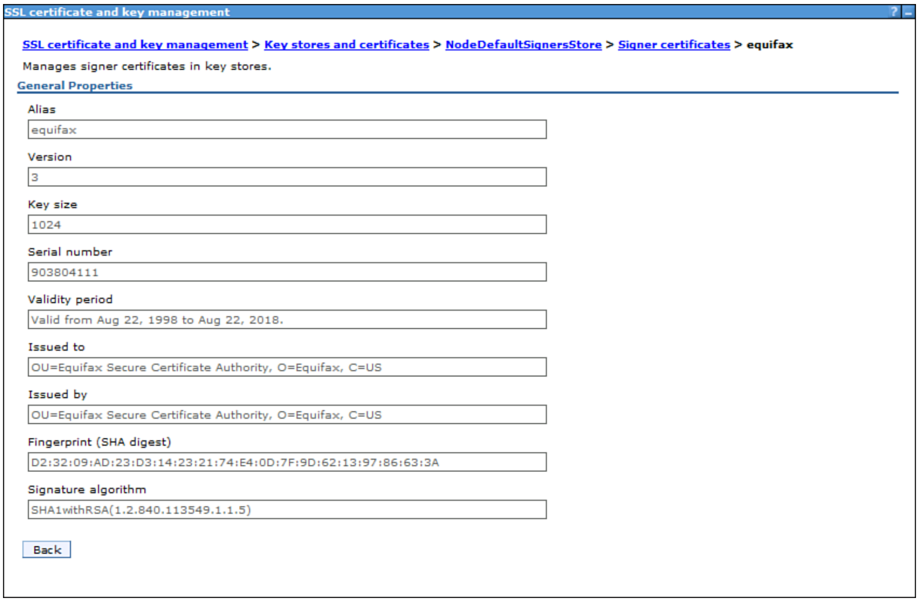
* Alias
equifax

* File name
/home/idcuser/BuiltinObjectToken-EquifaxSecureCA.pem

Data type
Base64-encoded ASCII data

Figure 7.5 Adding a trusted certificate signer in WAS 7

Save the configuration and check that it is good by viewing the certificate information from the list of trusted signers, as in Figure 7.6.


SSL certificate and key management

SSL certificate and key management > Key stores and certificates > NodeDefaultSignersStore > Signer certificates > equifax

Manages signer certificates in key stores.

General Properties

Alias	equifax
Version	3
Key size	1024
Serial number	903804111
Validity period	Valid from Aug 22, 1998 to Aug 22, 2018.
Issued to	OU=Equifax Secure Certificate Authority, O=Equifax, C=US
Issued by	OU=Equifax Secure Certificate Authority, O=Equifax, C=US
Fingerprint (SHA digest)	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
Signature algorithm	SHA1withRSA(1.2.840.113549.1.1.5)

Figure 7.6 Viewing a trusted certificate signer in WAS 7

After checking that the certificate was added properly, restart the server. This is done either with the IBM SmartCloud Enterprise or by logging on to the server with Putty.

Now let's write an application to see this work. The following servlet lists all the images in the catalog:

```
package com.ibm.cloud.examples.servlet;

import java.io.IOException;
import java.io.Writer;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.cloud.api.rest.client.bean.Image;
import com.ibm.cloud.examples.cache.CatalogCache;

/**
 * Servlet returns a list of images
 */
```

```
public class DescribeImagesServlet extends HttpServlet {

      protected void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
            Writer writer = response.getWriter();
            try {
                  List<Image> images =
CatalogCache.getInstance().getImages();
                  writer.write("Found " + images.size() + "
image(s).\n");
                  for (Image image: images) {
                        writer.write("ID: " + image.getID() + ", " +
image.getName());
                  }
            } catch(Exception e) {
                  writer.write("Error: " + e.getMessage());
            }
      }

}
```

You saw from other examples that retrieving a list of images and other data from the REST services can take some time because of calls across the Internet. However, in a web application, many users want certain sets of relatively unchanging data. This servlet uses a cache implemented by the `CatalogCache` class, shown here:

```
package com.ibm.cloud.examples.cache;

import java.io.IOException;
import java.util.List;

import com.ibm.cloud.api.rest.client.DeveloperCloud;
import com.ibm.cloud.api.rest.client.DeveloperCloudClient;
import com.ibm.cloud.api.rest.client.bean.Image;
import com.ibm.cloud.api.rest.client.exception.UnknownErrorException;

/**
 * Singleton class caches catalog data
 */
public class CatalogCache {

      private static final String USERNAME = "aamies@cn.ibm.com";
      private static final String PASSWORD = "hsing1yun";
      private static final long REFRESH_INTERVAL = 1000*60*60*24;  //
refresh every day
```

```
        private static CatalogCache instance;
        private List<Image> imageCache;
        private long lastRefreshed;

        private CatalogCache() {
        }

        public static CatalogCache getInstance() {
              return instance;
        }

        public List<Image> getImages() throws UnknownErrorException,
  IOException {
              if ((imageCache == null) || ((System.currentTimeMillis() -
  lastRefreshed) > REFRESH_INTERVAL)) {
                    DeveloperCloudClient client =
  DeveloperCloud.getClient();
                    client.setRemoteCredentials(USERNAME, PASSWORD);
                    imageCache = client.describeImages();
                    lastRefreshed = System.currentTimeMillis();
              }
              return imageCache;
        }

        public void invalidate() {
              imageCache = null;
        }
  }
```

The class calls the IBM SmartCloud Enterprise Java API and stores the data in a List, where it can be refreshed periodically.

Identity and Access Management

Identity and access management refers to the domain of management of users, groups, and user authentication and authorization. Being exposed on the public Internet, you must make sure that passwords and encryption methods are strong. This means that cloud applications should help users by preventing them from using weak passwords. The implication of multitenant awareness is that you need to know which tenant a logged-in user belongs to. The need for automation implies that you need to a way for users to either self-register or delegate user administration to responsible administrators. For example, the IBM SmartCloud Enterprise enables each tenant to have a user that can manage other users within that tenant. In addition, you need users to recover lost password information without needing to call a support center. The IBM SmartCloud Enterprise uses a central user- and password-management mechanism based on the IBM.com identity-management system. Coding all these in a small or medium-size development project is a tall order, so try to leverage standards and use third-party solutions wherever possible.

Authorization includes making sure that users can access only the data they are entitled to. This is based on membership in user roles. For example, in the IBM SmartCloud Enterprise, there are two user roles: End users may see only their own resources, and customer administrators may manage other users accounts and view the resources of other users. It is a best practice to externalize authentication and access control from your application.

A number of standards relate to identity and access. HTTP basic access authentication is an authentication method allowing a HTTP client to include a user name and password pair in an HTTP header. The user name and password are concatenated as `user_name:password` and are base 64 encoded before being added as an HTTP header. The IBM SmartCloud Enterprise uses HTTP basic authentication over HTTPS. Basic and Digest Access Authentication are defined in RFC 2617 [Franks, et al., 1999]. These methods may be acceptable over an HTTPS connection but would not provide good confidentiality guarantees over HTTP.

Storing user identity data can be done with different user repositories. Lightweight Directory Access Protocol (LDAP) is a protocol to access directory servers that often store identity data in a standard way. OpenID is a method for central user repositories operated by identity providers to store user account information that can be used across many web sites. The advantage is that it avoids the need for users to register multiple times at different web sites and maintain a long list of user names and passwords. Many applications opt to store user data in the same relational database that they use for transactional and other data. User management can be implemented with the IBM Tivoli Identity Manager in a central way across many different kinds of systems, including role management, delegated administration, and self-service. We discuss identity management in more detail in the upcoming sections "Managing Users with Lightweight Directory Access Protocol" and "Enabling an Application for Multitenant Access."A number of tools help manage identity and access in a standards-based and centralized way. The IBM Tivoli Access Manager is a system for authentication and authorization. The Tivoli Access Manager WebSEAL component is installed at the outer edge of the application in the DMZ to make sure that users gaining access to the application are first authenticated. Without using a centralized approach like this, you require developers to be very disciplined in enforcing authentication and run the risk of a careless web page developer forgetting to require access in a location. WebSEAL also allows for single sign-on, avoiding the need for users to have to enter their passwords more than once as they cross web pages that are driven by different back-end systems. The Tivoli Access Manager allows for different authentication methods, including LDAP.

In some cases when working in cloud environments, you might not feel confident in password authentication alone. You can increase the level of security by adding another factor of authentication—in other words, using a **two-factor authentication** method. A two-factor authentication method simply requires the use of two different authentication methods, such as a password and a private key. By default, SSH access to instances in the SmartCloud Enterprise by user idcuser requires a private key but no password. It is a good practice to set the password for idcuser the first time you access the account, which makes accessing the virtual machine a two-factor authentication method. You also need to edit the following line in the SSH server configuration file:

```
PasswordAuthentication yes
```

The next time you log in, you will still need your SSH key, but you will be prompted for the password as well.

Configuring Authentication and Access in J2EE Applications

In J2EE applications, you can configure authentication and access by editing the `web.xml` file. First, we add a stanza for a `security-constraint`:

```
<security-constraint>
        <web-resource-collection>
             <web-resource-name>User Personalization</web-resource-
name>
                <url-pattern>/username</url-pattern>
      </web-resource-collection>
      <auth-constraint>
               <role-name>authenticated-users</role-name>
        </auth-constraint>
</security-constraint>
```

This indicates that all users wanting to access any URL with the pattern `/username` must belong to the role `authenticated-users`. Next, you define a login method:

```
<login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>File Realm</realm-name>
</login-config>
```

This means that users should log in with HTTP Basic authentication using the authentication realm called `File Realm`. Finally, you define the security roles:

```
<security-role>
      <description>
              The role required to access the user interface
      </description>
      <role-name>authenticated-users</role-name>
</security-role>
```

This defines the role `authenticated-users`.

You can test this with a servlet that queries the user's name and role:

```
package com.ibm.cloud.example.security;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class UserNameServlet extends HttpServlet {
      private static final long serialVersionUID = 1L;

      protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
            PrintWriter writer = response.getWriter();
            Principal principal = request.getUserPrincipal();
            String userName = "Unknown";
            boolean userInRole = false;
            if (principal != null) {
                  userName = request.getUserPrincipal().getName();
                  userInRole = request.isUserInRole("authenticated-
users");
            }
            writer.println("User Name: " + userName +
            ", user in role: " + userInRole);
      }
}
```

The servlet gets the `Principal` object from the servlet request. If the user does not authenticate, this is null. If it is not null, you get the user name. In addition, you check whether the user is in the role `authenticated-users` with the `HttpServletRequest.isUserInRole()` method. Finally, you print the result. To try this, deploy the servlet into a web application called basicauth and map the servlet to URL `/username`. You can check this with a simple file realm, such as in Apache Tomcat. The default user store is in the file `conf/tomcat-users.xml`. Add the following entries to create a test user:

```
<tomcat-users>
      <user name="alex" password="***" roles="authenticated-users" />
        <role rolename="authenticated-users"/>
</tomcat-users>
```

Bring up the URL http://host:8080/basicauth/username. You should be prompted for the name and password that you entered into `tomcat-users.xml`. The servlet should print the following output:

```
User Name: alex, user in role: true
```

You are on the right track but have to make some adjustments. This example has several shortcomings:

- You have embedded security into your application.
- Your application is an island of security.
- You have not accounted for isolation of tenants.

Let's see how to address these points.

Managing Users with Lightweight Directory Access Protocol

In the previous section, you stored user information in an XML file. This has obvious scalability and interoperability limitations. Let's look at an improved way of storing user information.

Lightweight Directory Access Protocol (LDAP) is a protocol implemented by directory servers that provides a standard way of storing and looking up information on users and computing entities. Many directory servers implement LDAP, including IBM Directory Server, OpenLDAP, Novel Directory Server, and Microsoft Active Directory. LDAP servers are one of the most popular means of storing user information, and many tools support it. For example, many web servers, J2EE servers, and email servers can use LDAP as a user repository out of the box. LDAP directories are often used to centralize identity information for an enterprise in a single place, usually called an enterprise directory. In Java, you can interact with LDAP servers with the Java Naming and Directory Interface (JNDI). LDAP also supports transmission over SSL and TLS for security.

In LDAP, each entity is identified by its Distinguished Name (DN). Entities can support a flexible list of attributes that you can modify by editing the schema. Standard object classes are defined for user and identity management, such as inetOrgPerson, group, and organization. The object classes define the attributes allowed or required for any given entity. Entities can have multiple object classes. The standard objects are one of the keys enabling a common understanding of user and group objects. They also can be easily extended with additional attributes specific to your application context.

OpenLDAP is part of Red Hat Enterprise Linux distributions. It includes a directory server and management and client tools. We use it to demonstrate the use of LDAP for user management. You can install OpenLDAP on the RHEL virtual machines on the IBM SmartCloud Enterprise using this command:

```
# yum install openldap openldap-servers
```

This installs both server and client tools. It creates the file `/etc/openldap/ldap.conf`, which is the configuration file for the OpenLDAP client tools, and the file `/etc/openldap/slapd.conf`, which is the configuration tool for the server. The schema files are stored in the directory `/etc/openldap/schema/`. The top of the directory tree is called the **directory context**, abbreviated to dc, and is used as a suffix in the distinguished names for your objects. Edit `/etc/openldap/slapd.conf` to set the suffix for your organization and the root user:

```
suffix          "dc=example,dc=com"
rootdn          "cn=Manager,dc=example,dc=com"
```

Set the password for the root user using the `slappasswd` command. Start the LDAP service using the `service` command, as shown here:

```
# /sbin/service ldap start
```

You can check whether it is running by using this command:

```
# ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
```

The `ldapsearch` command is a tool for searching the directory. It shows an error if the server is not running properly.

Directory data can be exchanged using the LDAP Data Interchange Format (LDIF), which is a text format that is simple to edit manually. Add a root entry at the top of the directory tree to represent your organization. Switch to user ldap using the Linux `su` command. Create a file called `organization.ldif` that contains this text:

```
dn: dc=example,dc=com
objectclass: dcObject
objectclass: organization
o: Example Company Inc.
dc: example
```

This is an example of the LDIF format mentioned earlier. Two object classes exist: `dcObject` and organization. Add this to the directory using the `ldapadd` command:

```
ldapadd -x -D "cn=Manager,dc=example,dc=com" -W -f organization.ldif
```

You can check that it has been added by using this command:

```
ldapsearch -x -b 'dc=example,dc=com' '(objectclass=*)'
```

You should see output that matches your LDIF file.

LDAP can support a hierarchical structure to organize resources. It can help organize users, groups, and other objects in different branches of a **directory information tree**. This term describes the topology of the organization of objects within a directory. A simple directory information tree separates people into one branch of the tree and roles in another branch, as shown in Figure 7.7.

To create a branch of the tree for people, create an LDIF file called `ou_people.ldif` with the following content:

```
dn: ou=people,dc=example,dc=com
objectclass:organizationalunit
ou: people
description: A branch for people
```

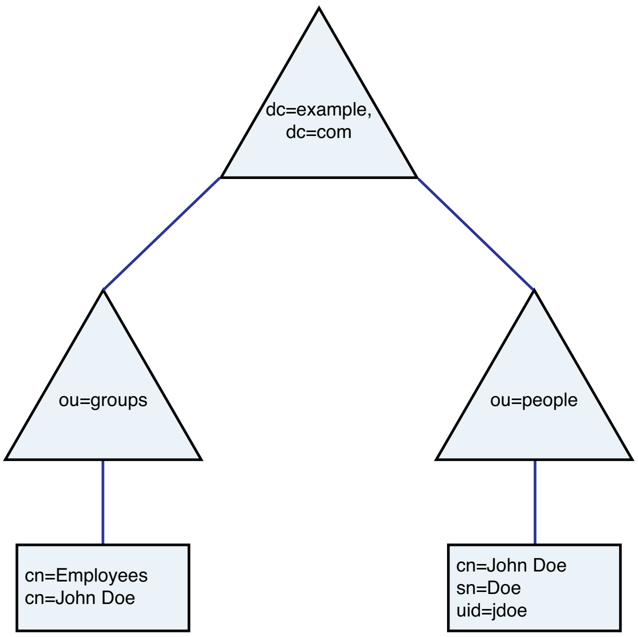


Figure 7.7 Example directory information tree

In LDAP, `ou` is the attribute name for organizational unit. Add the `ou` entity with the following `ldapadd` command:

```
ldapadd -x -D "cn=Manager,dc=example,dc=com" -W -f group_people.ldif
```

You can also add users to the directory using LDIF. Create an LDIF file with the following text:

```
dn: cn=John Doe,ou=people,dc=example,dc=com
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: John Doe
sn: Doe
uid: jdoe
userpassword: secret
```

This represents a user called John Doe. In a real application, you would normally use a user interface or other tool to add users. In cloud applications, users are often added by self-registration. However, LDIF has its uses in bulk import of user data. Add the user with the `ldapadd` command, as before.

```
# ldapadd -x -D "cn=Manager,dc=example,dc=com" -W -f person.ldif
```

You can check that the user was added properly with the `follow` command:

```
# ldapsearch -x -b 'ou=people,dc=example,dc=com'
'(objectclass=inetorgperson)'
```

The previous LDIF data should be returned. With a large number of entities in the directory server, organizing groups into their own branch of the directory tree is also convenient. Create an LDIF file called `group_ou.ldif` with the following content:

```
dn: ou=groups,dc=example,dc=com
objectclass:organizationalunit
ou: groups
description: A branch for groups
```

Add the `ou` entity with `ldapadd`, as in the previous examples. Next, create an LDIF file called `employees.ldif` with the group definition, as shown here:

```
dn: cn=employees,ou=groups,dc=example,dc=com
objectclass: groupofnames
cn: employees
description: Example Company employees
member: cn=John Doe,ou=people,dc=example,dc=com
```

This adds the user John Doe to the group employees. Add the group entity with `ldapadd`, as in the earlier examples.

You can configure the IBM WebSphere Application Server to use an LDAP directory server as a user repository. You can do this in either federated or stand-alone mode. Federated mode combines the out-of-the-box file realm with an LDAP realm. Stand-alone mode is the best mode for a production system, for performance reasons. However, you have the additional challenge of replicating the WebSphere administrative user and group to LDAP. You perform the following steps with WAS 7.0 for federated mode.

1. Click Security, Global Security.
2. Select Federated Repositories and click Configure.
3. Click Add Base Entry to Realm.
4. Click Add Repository.

Enter the following values in the form:

Repository identifier: ldap
LDAP Server: Custom
Primary Host name: localhost

Port: 389
Bind distinguished name: dc=example,dc=com
Bind password: ***

Click OK and Save. You must register the group object class (`groupofnames`) and group membership attribute (`member`) in the WebSphere console as well. You can do this under Global Security, Federated Repositories, Manage Repositories, {Realm Name}, Group Attribute Definition. Restart the WebSphere server. Now you can deploy a web application with a security description similar to the previous example and verify that users both have valid entries in the LDAP directory server and belong to the correct group.

Enabling an Application for Multitenant Access

According to the NIST definition of cloud computing, multitenant access is one of its essential characteristics. You can think of a tenant as an enterprise. For this, you need to isolate a tenant's data from all other tenants. Two basic steps are required to achieve this:

1. When the user logs in, the web server authenticates him or her and also finds the tenant the user belongs to. This is usually stored in the user's session.
2. Every time the user accesses or modifies data in the data store, you check that all the data is owned by that tenant.

For the first step, you must associate a tenant identifier with the user record. Typically, each user record in the user repository contains at least these attributes:

- **User ID**—How the application recognizes the user
- **Credential**—For example, a password
- **User name**—How other users recognize this user
- **Role**—Linked to the actions that a user can perform, such as end user or administrator
- **Tenant ID**—An identifier for a business entity that wants to separate all its resources from other business entities

With this scheme, you can look up the user's role and tenant ID at the time you check the credentials upon login. You also need a tenant ID associated with data in the data store (usually a relational database) to identify which tenant owns what data. With the LDAP approach for managing users described earlier, you can extend the user object schema to include a tenant ID.

Federated Identity Management

Federated identity management involves being able to share identity information between a federation of identity and service providers. The most frequently needed business case for this is single sign-on among multiple identity providers. For example, consider an employee wanting to view health insurance benefits. The employee navigates from the employer's intranet and follows a link to the health insurance provider's web site. With federated single sign-on, the health insurance provider's web site can trust the authentication at the employer's intranet. This is interesting in a cloud environment because you want to provide users with a federation of cloud services.

Federated single sign-on is usually accomplished by first setting up a trust relationship between identity and service providers and then making use of a cryptographic token service to exchange authentication information. A commonly used standard for the token service is Security Access Markup Language (SAML). The tokens exchanged represent user identity information and assertions and come in different flavors, including Lightweight Third Party Authentication (LTPA), SAML, Passticket, X.509 Certificate, and Kerberos.

The IBM Tivoli Federated Identify Manager (FIM) is a software product that enables federated single sign-on and other federated identity-management scenarios. It can integrate with many authentication and access platforms and protocols, including SAML, WS-Federation, Liberty ID-FF, Information Card Profile, OpenID, .NET, and SAP Dreamweaver.

OAuth

OAuth is a protocol to allow secure API authorization. It can allow consumer applications written by enterprises or independent third-party software vendors to interact with protected data from service providers. OAuth enables users to access their data via these third-party applications without sharing passwords. OAuth 1.0 is described in RFC 5849, and OAuth 2.0 is currently in draft. IBM SmartCloud for Social Business, Twitter, Google, LinkedIn, and Facebook are example platforms that use OAuth for the authentication of users and third parties with their APIs. OAuth can support the various different scenarios that these popular web sites need.

Traditional client/server or web applications have no standard model for third parties to access user data, other than the user sharing a password with the third-party application. This is not desirable from a security standpoint because it gives the owner of the third-party application too much control over user data. It can also lead to practices in which the password is left exposed in application configuration files and scripts. This is the motivation for development of the OAuth standard.

Four roles are defined in OAuth: client, resource server, resource owner, and authorization server. In a cloud context, the resource server is the cloud provider. The client is a third-party application that interacts with the resource server to access a protected resource, which is owned by the resources owner. The authorization server ensures that access to the protected resource is authorized by the resource owner. The authorization server then issues the client a token. Typically, the client is a third-party application that leverages a platform provided by the resource server. The authorization server can be a component of the resource server or a separate entity. The client must also have credentials that the authorization server recognizes, typically through a registration process. Figure 7.8 shows a schematic diagram of the OAuth flow.

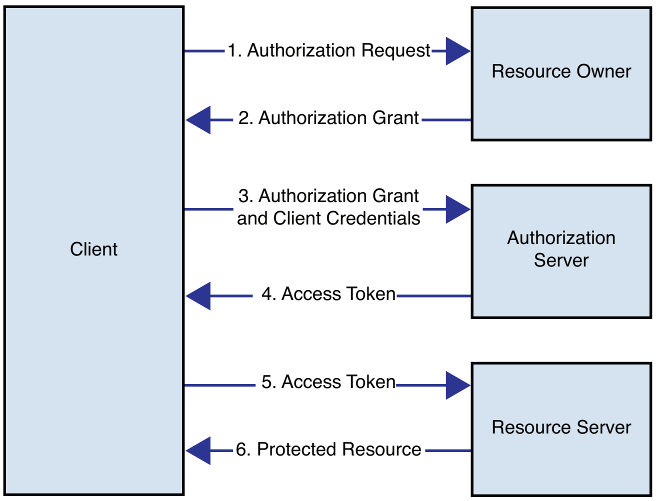


Figure 7.8 Schematic of OAuth flow

Consider a more detailed description of the steps:

1. The client requests access to the resource. Usually, this is via the authorization server.
2. The client receives an authorization grant.
3. The client requests an authorization grant.
4. The authorization server issues an access token.
5. The client uses the access token to request access to the protected resource.
6. The resource server returns the protected resource to the client.

The flow ensures that the resource owner authenticates with the resource server so that the resource owner never needs to share authentication credentials with the client. When the access token is issued to the client directly, it is termed an implicit grant. This is useful for browser-based clients.

Three kinds of credentials are used in OAuth: client credentials, temporary credentials, and token credentials. Client credentials authenticate the client. Token credentials are used in place of a resource owner's user name and password. The client uses the token credentials instead of asking the resource owner for a user name and password. Token credentials can be limited in scope and duration.

The Twitter REST API methods enable access to Twitter data using OAuth 1.0. Let's look at it to get a feel for what using OAuth 1.0 is like. To use the Twitter API, you need to register your application first. In this step, each client application is issued with a consumer key and a secret that are used to sign each API call. When you register your application with Twitter, you must provide the follow information:

- Application name
- Application description
- Web site
- Organization
- Type (browser or desktop)
- Callback URL

You can enter your own application name and description. To get a web site, create a new instance of a LAMP server, as described in the section “Linux, Apache, MySQL, and PHP” in Chapter 2. You can leave the callback URL blank and supply this as a parameter in your API calls when you develop your application. After you register, you are given a 22-character alphanumeric key and a 41-character secret.

To authenticate to Twitter, these steps take place:

1. The application requests a token.
2. Twitter grants a token and directs the user to the service provider.
3. The user grants authorization.
4. Twitter directs the user to the consumer.
5. The application requests an access token.
6. Twitter grants an access token.
7. The application can access the protected resources.

The Twitter OAuth implementation requires that requests be signed using the HMAC-SHA1 message digest algorithm.

What if hosting a web site is not convenient for you? Since the OAuth standard 1.0 was published, several related use cases have been proposed that it does not handle well. These use cases include JavaScript embedded in HTML pages and native applications that run on desktops or mobile phones. The OAuth 2.0 draft standard has four different flows to handle these use cases:

- Authorization code flow, Section 4.1 (classic 3-legged from OAuth 1.0)
- Implicit grant flow, Section 4.2 (browser-based clients that cannot maintain a secret, authorized by a user)
- Resource owner password credentials, Section 4.3 (end user gives user name and password to application client)
- Client credentials flow, Section 4.4 (2 and 3 legged flow for non-browser-based clients)

OAuth 2.0 is not compatible with OAuth 1.0. It does not require use of a secret code issued to the client, except in the specific flows described earlier (the first in the previous list). It also adds a refresh token to allow the client to continue to access protected resources without having to seek additional access tokens after the first one has expired.

OAuth 2.0 considers different types of clients:

- **Web applications**—This type of client is capable of securely storing client credentials so that the client credentials flow is suitable for these clients.
- **Browser-based applications (user agent–based applications)**—These clients are not capable of securely storing client credentials, so the implicit grant flow is most appropriate.
- **Native applications**—These applications execute on the same operating system as the resource owner uses. This type of client is capable of securely storing dynamically issued credentials but not client credentials, so the authorization code flow is suitable for these clients.

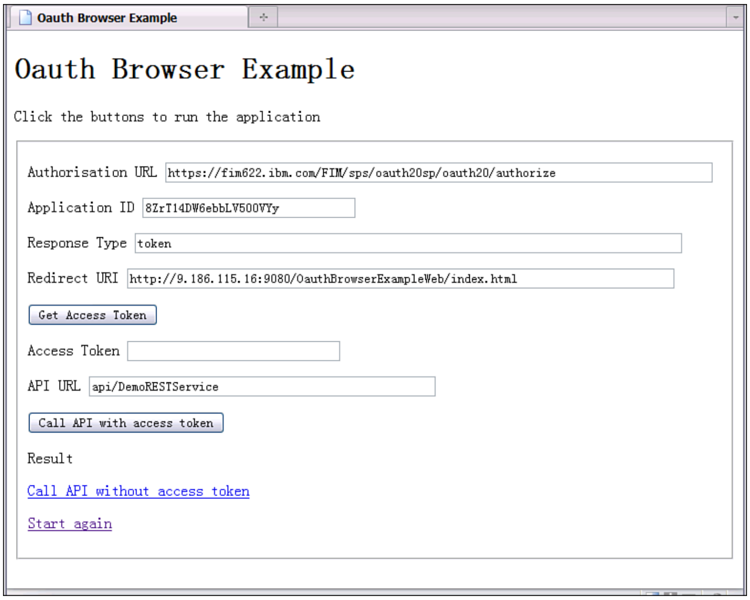
A key weakness with using OAuth 1.0 with HTML/JavaScript clients is the need to store a secret key in an HTML page, which would be available to all users to view. OAuth 2.0 introduces a different flow for this scenario, with two assumptions:

- The user can log on with the same browser the JavaScript code is executing within.
- The transaction takes place with HTTPS, so that there is no need to sign messages.

OAuth 2.0 defines authorization endpoints and token endpoints. An authorization endpoint is used to obtain authorization from a resource owner. A token endpoint is used to obtain an access token from an authorization grant. Not every flow described in OAuth 2.0 uses both endpoints. In issuing an authorization grant, the authorization server must first authenticate the resource owner. Because the user credential may be passed in clear text, HTTPS or some other protocol over transport-layer security (TLS) must be used. The authorization server should require the client to register a redirection URI in advance so that the access token is sent via user agent redirection. This is needed to prevent a session fixation attack.

Let's look at the user experience for a browser-based OAuth 2.0 example. We can explain the flow for a simple HTML client with the following screens and sample code. Suppose that the user wants to log in and then make use of a third-party service that invokes REST APIs provided by the cloud platform. Figure 7.9 illustrates this.

Of course, the user would not see the token fields displayed in the figure. This HTML page would be written by a third-party service provider who created the service by making some REST calls. The client requests an access token from the authorization server using HTML form submit. This is followed by a user consent form (not shown) displayed by the authorization server. After the authorization server obtains user approval for request for an application to call API, it forwards the access token back to the application. Figure 7.10 shows this.



Oauth Browser Example

Click the buttons to run the application

Authorisation URL https://fim622.ibm.com/FIM/sps/oauth20sp/oauth20/authorize

Application ID 8ZrT14DW6ebbLV500VYy

Response Type token

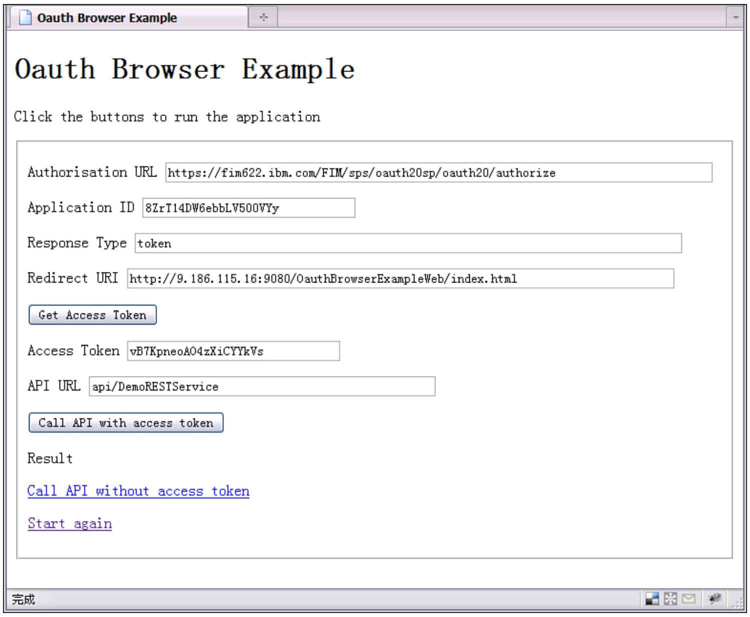
Redirect URI http://9.186.115.16:9080/OauthBrowserExampleWeb/index.html

Access Token

API URL api/DemoRESTService

Result

Figure 7.9 OAuth demonstration: user navigates to a screen where a cloud REST API is invoked



Oauth Browser Example

Click the buttons to run the application

Authorisation URL https://fim622.ibm.com/FIM/sps/oauth20sp/oauth20/authorize

Application ID 8ZrT14DW6ebbLV500VYy

Response Type token

Redirect URI http://9.186.115.16:9080/OauthBrowserExampleWeb/index.html

Access Token vB7KpneoA04zXiCYYkVs

API URL api/DemoRESTService

Result

Figure 7.10 OAuth provider generates an access token and sends it to the redirect URL

The OAuth provider generates an access token and redirects back to the application at the redirect URL. The REST API call now is called with code similar to this:

```
function callAPI() {
    var accessToken = $('access').value;
    var tokenString = 'Bearer ' + accessToken;
    var url = $('url').value + '?' + new Date().getTime();
    new Ajax.Request(url, {
        method:'get',
        requestHeaders: ['Authorization', tokenString],
        onSuccess: function(transport){
        var response = transport.responseText || 'no response text';
            $('result').innerHTML = response;
        },
        onFailure: function(transport){
            var response = transport.responseText || 'no response
text';
            alert('callAPI failure: ' + response);
      }
});
return false;
}
```

The JavaScript function uses the open source Prototype JavaScript library. The result looks like Figure 7.11.

The application invokes the REST API using AJAX with an access token in the header and prints output to the page.

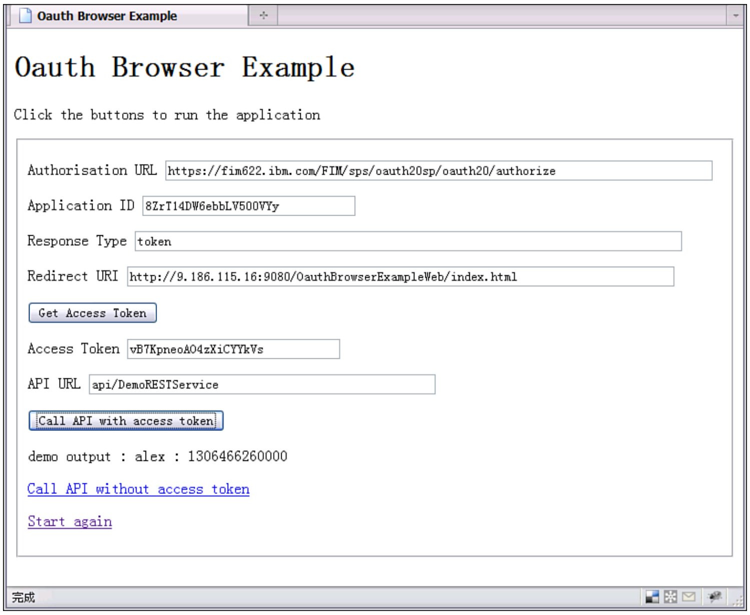
Network Security

Because all resources reside on the network in cloud computing, network security is essential. This section describes some basic concepts, considerations, and tools in network security.

Firewalls

An individual firewall is a firewall that is installed on the same server as the resource it is protecting. This is an essential tool in cloud computing. Most modern operating systems, including all the images on the IBM SmartCloud Enterprise, are packaged with an individual firewall. On Linux virtual machines, this is iptables; on Windows, it is a Microsoft solution. On the IBM SmartCloud Enterprise, a firewall also exists between the hypervisor and the virtual machines that it manages.

A **firewall rule** specifies a set of criteria for a network packet and a target. When a network packet arrives, each rule is checked. If the packet does not meet the criteria for the rule, the next rule is checked.



Oauth Browser Example

Click the buttons to run the application

Authorisation URL https://fim622.ibm.com/FIM/sps/oauth20sp/oauth20/authorize

Application ID 8ZrT14DW6ebbLV500VYy

Response Type token

Redirect URI http://9.186.115.16:9080/OauthBrowserExampleWeb/index.html

Access Token vB7KpneoA04zXiCYYkVs

API URL api/DemoRESTService

demo output : alex : 1306466260000

Figure 7.11 Application invokes REST API

On SUSE machines, you can use the YaST administration utility to add firewall rules. For example, to allow port 50030 (used by Hadoop) from any outside address, start YaST from the command line by typing the command `yast` as root and navigate to Security and Users, Firewall using the Tab key; then click Enter. You will see a screen similar to Figure 7.12

Navigate to Custom Rules and click Enter. Navigate to Add and click Enter. Enter 0/0 for the Source Network, which indicates any source computer, and 50030 for the port, which is the Hadoop port you are interested in (see Figure 7.13).

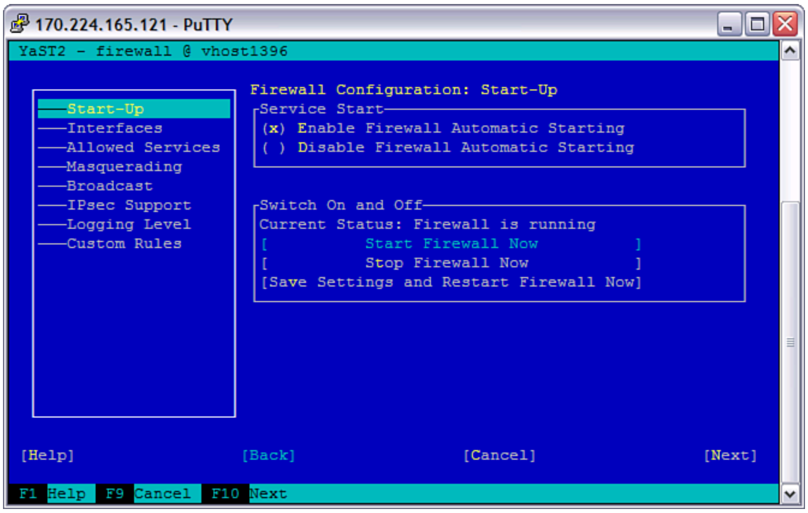
Click Add, Next, and Finish. You do not have to manually restart; YaST takes care of this.

On Red Hat images, you can use the `iptables` command to manage firewall rules. This is the basic form of an `iptables` command:

```
# iptables [-t table] -[AD] chain rule-specification [options]
```

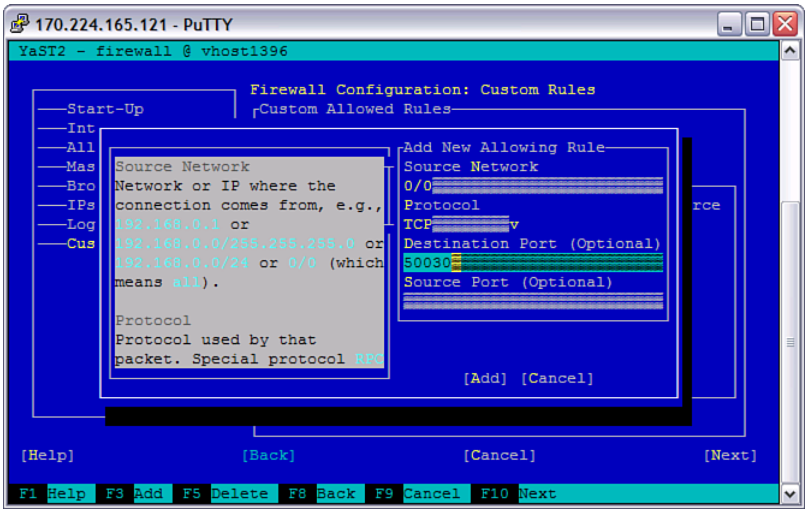
The actions associated with a firewall rule include `ACCEPT`, `DROP`, `QUEUE`, and `RETURN`. If you don't want to accept a network packet, you should specify a `DROP` action. In the `iptables` command, A appends a rule, and D deletes one.

Three firewall tables exist. The default table is named `filter`. This table contains three chains: `input`, `forward`, and `output`. The `input` chain is for packets coming in to the local sockets, the `forward` chain is for packets that are routed, and the `output` chain is for locally generated packets.



```
170.224.165.121 - PuTTY
YaST2 - firewall @ vhost1396

                              Firewall Configuration: Start-Up
 ──Start-Up                   ┌Service Start────────────────────────────────┐
 ──Interfaces                 │(x) Enable Firewall Automatic Starting       │
 ──Allowed Services           │( ) Disable Firewall Automatic Starting      │
 ──Masquerading               └─────────────────────────────────────────────┘
 ──Broadcast
 ──IPsec Support              ┌Switch On and Off────────────────────────────┐
 ──Logging Level              │Current Status: Firewall is running          │
 ──Custom Rules               │[          Start Firewall Now          ]     │
                              │[          Stop Firewall Now           ]     │
                              │[Save Settings and Restart Firewall Now]     │
                              └─────────────────────────────────────────────┘

[Help]               [Back]               [Cancel]               [Next]
F1 Help  F9 Cancel  F10 Next
```

Figure 7.12 YaST firewall management utility



```
170.224.165.121 - PuTTY
YaST2 - firewall @ vhost1396

                              Firewall Configuration: Custom Rules
 ──Start-Up                   ┌Custom Allowed Rules─────────────────────────┐
 ──Int
 ──All                                        ┌Add New Allowing Rule────────┐
 ──Mas    Source Network                      │Source Network               │
 ──Bro    Network or IP where the             │0/0                          │
 ──IPs    connection comes from, e.g.,        │Protocol                     │ rce
 ──Log    192.168.0.1 or                      │TCP        v                 │
 ──Cus    192.168.0.0/255.255.255.0 or        │Destination Port (Optional)  │
          192.168.0.0/24 or 0/0 (which        │50030                        │
          means all).                         │Source Port (Optional)       │
                                              │                             │
          Protocol                            └─────────────────────────────┘
          Protocol used by that
          packet. Special protocol RPC              [Add] [Cancel]

[Help]               [Back]               [Cancel]               [Next]
F1 Help  F3 Add  F5 Delete  F8 Back  F9 Cancel  F10 Next
```

Figure 7.13 Custom firewall rule in YaST

As an example, to allow network packets from any source on port 80, the default HTTP port, use this command:

```
# /sbin/iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

This adds a rule to the INPUT chain of the filter table for TCP packets on port 80 with an ACCEPT action. The -p parameter specifies the protocol—tcp, in this case. The --dport 80 option is the destination port—80, in this case. The -j (jump) option is the target—ACCEPT, in this case. It can be a good practice to leave firewall rules in place only for as long as you need them. The command form is ideal for doing this. However, often, you will prefer to keep the rules permanently, including after the next time you restart the instance. To do this, edit the file /etc/sysconfig/iptables. A typical iptables file looks like this:

```
*filter
:INPUT DROP [67:14849]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [346:34696]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
COMMIT
```

This specifies the rules for the filter table. All incoming packets from ports 67 to 14849 are dropped. No forwarding is allowed, all outgoing packets on ports 346 to 34696 are allowed, and incoming packets on port 22 (SSH) are allowed. After you have made the edits and saved the file, start or restart the iptables service with this command:

```
# /sbin/service iptables restart
```

If you have made changes with the iptables command, you can save them with this command:

```
# /sbin/service iptables save
```

Check the status of the firewall with this command:

```
# /sbin/service iptables status
```

Some performance advantages can be gained from using the hypervisor firewall rules. However, you cannot manage them as dynamically as you can the rules for the local firewall, and you can do so only for your personal or community-level visibility images. You can set the hypervisor firewall rules at instance creation time using the parameters.xml file on the IBM SmartCloud Enterprise. You can find this file in the asset catalog entry for your image. A typical parameters.xml file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<parameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="parameters.xsd">
   <firewall>
      <rule>
         <source>0.0.0.0/0</source>
         <minport>1</minport>
         <maxport>65535</maxport>
```

```
                </rule>
            </firewall>
        </parameters>
```

This specifies that any source can connect to the instance on any port, from 1 through 65535. You can tighten this by editing and adding `minport` and `maxport` elements. For example, to restrict network traffic to only ports 22 (SSH) and 80 (HTTP), you can use these rules:

```
        <?xml version="1.0" encoding="UTF-8"?>
        <parameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="parameters.xsd">
           <firewall>
               <rule>
                  <source>0.0.0.0/0</source>
                  <minport>22</minport>
                  <maxport>22</maxport>
               </rule>
               <rule>
                  <source>0.0.0.0/0</source>
                  <minport>80</minport>
                  <maxport>80</maxport>
               </rule>
           </firewall>
        </parameters>
```

After editing the file, upload it to the image catalog for your image, replacing the old file, and start a new instance.

Linux firewalls can also be used to protect servers other than the server on which the firewall resides. Actually, this is a preferred configuration because it provides an additional level of isolation. To do this, you need to isolate the servers that you are protecting from the Internet by placing them on a VLAN. You must add two IP addresses on the firewall virtual machine. Finally, you need to configure rules for forwarding between the Internet and the VLAN in `iptables` (see Figure 7.14).

In Figure 7.14, virtual machine VM1 is directly connected to the Internet. The firewall VM2 has two IP addresses: One is Internet accessible, and one is on the private VLAN. Virtual machine VM3 has a single IP address that is accessible only on the VLAN. A firewall rule allowing inbound access from the Internet and mapping the firewall's IP to VM3's IP for a certain port allows inbound access from the Internet on that port only. For example, if this was a web server and a user attempted to access the firewall's IP address on port 80, that request would be forwarded to a web server on VM3. No other ports on VM3 would be open. Another firewall rule allows outbound access for VM4. This can be done by using ports dynamically for an individual TCP connection. For example, if VM4 attempts to open a TCP connection to the Internet, the firewall can find an available port and change the TCP packet to contain this port and the firewall's own IP address. When it receives a packet back from the Internet in response to the request from VM4, it does the reverse mapping. In this way, the virtual machine VM4 can open connections to the Internet without being visible on the Internet.

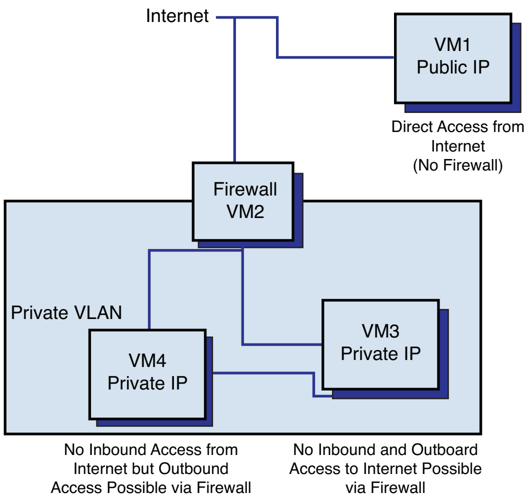

Internet
VM1
Public IP
Direct Access from Internet (No Firewall)
Firewall
VM2
Private VLAN
VM3
Private IP
VM4
Private IP
No Inbound Access from Internet but Outbound Access Possible via Firewall
No Inbound and Outboard Access to Internet Possible via Firewall

Figure 7.14 Schematic diagram of a firewall protecting a VLAN

Example: Connecting to a VLAN through a Firewall

Our scenario in this section is that the composite application is hosted on a VLAN, and the users of the application are on the Internet. We host the database, application server, and other elements of our composite application on a VLAN for better security. The goal in this section is to allow general access to the web interface of the composite application via a firewall rule.

Operating System Network Security Mechanisms

A separate network security mechanism on Linux is the `hosts.allow` and `hosts.deny` files in the `/etc` directory and related services. These files use the `host_access` format, which is a simple access-control language based on client host name, address, and user name. This is typically used with the `xinetd` and `rpc` daemons. If a request matches an entry in the `hosts.allow` file, it is allowed. Otherwise, the `hosts.deny` file is checked: If it matches, it is denied; if not, the request is allowed. By default, many of the images on the IBM SmartCloud Enterprise are empty for both `hosts.allow` and `hosts.deny`. This allows any host to connect to all services. To tighten access, add an entry like this in `hosts.deny`:

```
ALL: ALL
```

This denies access to all services from all hosts. Then add rules for specific services in `hosts.allow`:

```
ALL: LOCAL
ALL: jumpbox.example.com
```

This allows local access to all services and also all services from the JumpBox. To access these services from your local PC, you must log into JumpBox and then log into the server with the services.

Business Scenario: Network Deployment and Firewall Rules

In our IoT Data scenario, we have a typical multitiered application with the addition of the Hadoop cluster. Figure 7.15 shows the data flows.

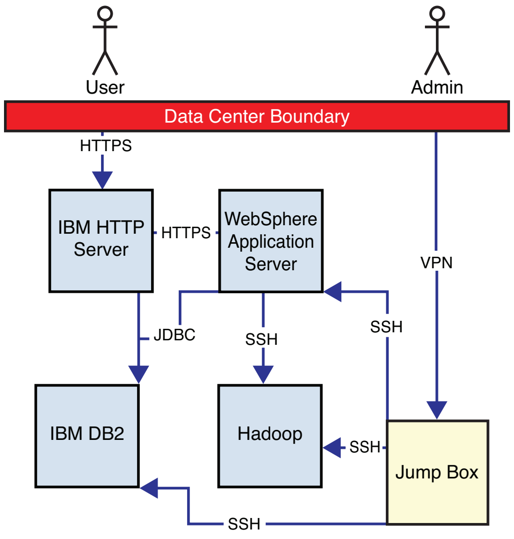


Figure 7.15 Network deployment for IoT Data web application
Legend Blue: Internet; yellow: VPN

Our goal in formulating a network security architecture is to reduce the risk of servers being compromised by reducing Internet exposure and layering using network zones. We do this using the firewalls on the individual systems and using the VLAN feature of the IBM SmartCloud Enterprise. The only traffic that we allow through the data center boundary is from end users to the HTTP server via HTTPS and from administration users to the JumpBox located on the VLAN. Access to the other systems is regulated according to the firewall rules in Table 7.1.

Table 7.1 Description of Firewall Rules for IoT Data Web Application

Source	Destination	Protocol	Description
Users	HTTP server	HTTPS	Access to web site
Admin users	JumpBox	VPN	Administration
HTTP server	WebSphere	HTTPS	Proxy to application server
WebSphere	DB2	JDBC	Database access
WebSphere	Hadoop	SSH	File access
JumpBox	HTTP server	SSH	Administration
JumpBox	WebSphere	SSH	Administration
JumpBox	DB2	SSH	Administration
JumpBox	Hadoop	SSH	Administration

The firewalls should be configured on the destination machines.

Proxy Servers

Proxy servers provide security benefits by isolating clients from servers. Proxy servers are most often used with HTTP but can be used with other protocols. Proxy servers can be either forward or reverse proxies. The can work at the application protocol level or at a deeper level. SOCKS is a protocol to allow for proxies at a lower level than the application protocols, such as HTTP and FTP. It is defined in RFC 1928.

A **forward proxy** is an intermediate server that is located between a client and an **origin server**. It helps protect the client from direct access to the Internet and can also regulate which web sites are visible to a client. To access the origin server, the client must send a request to the proxy naming the origin server. The proxy then returns data from the origin server to the client. The client needs to be configured to use the proxy. Most often, a forwarding proxy is used as a security feature to protect clients from direct access to the Internet and to ensure that clients do not have access to content on the Internet that is not appropriate for the workplace where the client is located. In this case, it is providing some of the features of a firewall. It is important to allow only authorized clients to access your proxy. Otherwise, the proxy may be abused, and HTTP requests from the malicious user will appear to have originated from your proxy.

A **reverse proxy** appears to the client as an ordinary web server but actually forwards requests to one or more servers. This is most often done for the purposes of authentication or load balancing. For example, to provide single sign-on across multiple web servers, an authentication server can intercept requests and check whether the user has authenticated before accessing the server. If not, the authentication server prompts the user to log on. After authentication, no matter which HTTP server the user attempts to access, the authentication server passes an authentication token containing the identity of the user to the HTTP server in a HTTP header. The IBM Tivoli Access Manager acts as a reverse proxy in this role to provide single sign-on.

Apache can be configured as either a forward or reverse proxy. The Apache `mod_proxy` provides basic proxy capabilities for the Apache HTTP server. Several other proxy modules, including `mod_proxy_http`, `mod_proxy_ftp`, `mod_proxy_ajp`, and `mod_proxy_balancer`, provide extensions to this for specific protocols and capabilities. See the section "Linux, Apache, MySQL, and PHP" in Chapter 2 for how to set up the Apache HTTP server. To configure the proxy, edit the file `/etc/httpd/conf/httpd.conf`, removing the commented-out section from the default file:

```
<IfModule mod_proxy.c>
ProxyRequests On

<Proxy *>
    Order deny,allow
    Deny from all
    Allow from .example.com
</Proxy>
```

Replace `.example.com` with your IP address or range or host name pattern. Try it by editing the proxy settings on your browser. On Firefox, this is under Tools, Options, Internet, Connections. Select Manual Setting and enter the IP address of your proxy server. To test the proxy, enter the URL of a web site in the browser address bar. The page should display successfully in your browser, and you should be able to see a record of the request in the proxy server access log at `/var/log/httpd/access_log`, as shown here for the URL Google.com:

```
202.108.130.138 - - [04/Jul/2011:06:49:35 +0000] "GET
http://www.google.com/ HTTP/1.1" 200 16384 "-" "Mozilla/5.0 (Windows;
U; Windows NT 5.1; zh-CN; rv:1.9.2.18) Gecko/20110614 Firefox/3.6.18"
```

Some enterprises require all Internet access to be via a proxy. This can interfere with Java programs that use REST services, such as the IBM SmartCloud Enterprise REST API. For some programs that use only basic networking methods, you can solve this by adding standard Java VM properties to the command line that the Java program using the SCE APIs is launched from for the proxy:

- **http.proxyHost**—The host name of the proxy server.
- **http.proxyPort**—The port number, with a default value of 80.
- **http.nonProxyHosts**—A list of hosts that should be reached directly, bypassing the proxy. This is a list of regular expressions separated by `|`. Any host that matches one of these regular expressions is reached through a direct connection instead of through a proxy.

The properties can also be set programmatically within the Java program using the `System.setProperty(String, String)` method. This is a Java virtual machine–wide setting. You can check that the proxy is actually being used by checking the access log, as described earlier. You can try it with this program:

```
package com.ibm.cloud.examples.rest;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;

public class ProxyTest {

      public static void main(String[] args) throws Exception {
            URL url = new URL("http://www.google.com");
            URLConnection connection = url.openConnection();
            BufferedReader reader = new BufferedReader(
                        new
InputStreamReader(connection.getInputStream()));
            String line = null;
            while ((line = reader.readLine()) != null) {
                  System.out.println(line);
            }
      }
}
```

You can invoke it like this:

```
java -Dhttp.proxyHost=proxy.example.com
om.ibm.cloud.examples.rest.ProxyTest
```

The main drawback is that it does not work with the Apache HttpClient, which the IBM SmartCloud Enterprise Java REST client uses. To set a proxy with Apache HttpClient, use code similar to this:

```
httpclient.getHostConfiguration().setProxy("proxy.example.com", 80);
  httpclient.getState().setProxyCredentials("example-realm", "
examplehost",
  new UsernamePasswordCredentials("username", "password"));
```

The IBM SmartCloud Enterprise REST Java API has an interface to set the proxy in DeveloperCloudClient.

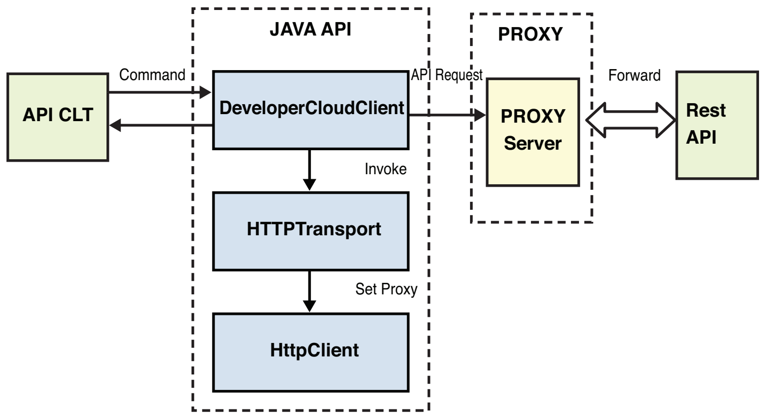


Figure 7.16 Use of proxy by the IBM SmartCloud Enterprise Java client

Virtual Private Networks

Virtual private networks (VPNs) rely on encryption to create an extension of a private network over the Internet. VPNs enable several network scenarios that are valuable to enterprises. A traditional use of VPNs is to connect the local area networks of different offices of an enterprise into a **wide area network (WAN)**. These types of connections are site-to-site. When VPNs were introduced for this purpose, they replaced the use of leased lines, greatly reducing cost for the enterprises. Another traditional use of a VPN is to enable employees to access an enterprise's private network remotely, for example, to work from home. In this scenario, the enterprise provides a VPN gateway that is accessible from the Internet, and the employee installs a VPN client that he or she installs on a laptop to access applications such as email. This is termed a **mobile virtual private network** because one of the endpoints (where the employee is located) does not have a fixed IP address.

When a client sends a packet through a VPN gateway, an authentication header is added, the data is encrypted, and the data is placed in an Encapsulating Security Payload. The receiving VPN server decrypts the data and routes the packet to the destination according to information in the header.

The encryption provided by VPNs is at a low level, so all communication to the enterprise is encrypted. This can be at either OSI Layer 2 (data link layer) or Layer 3 (network layer) and can include any of these methods:

- IPSec
- SSL/TLS
- Datagram Transport Layer Security (Cisco)
- Microsoft Point-to-Point encryption
- SSH tunneling

VPNs can use bridging to create a virtual wide area Ethernet LAN. This has some advantages over using routing because it can work with any protocol that can work over Ethernet. Bridging is needed if you are using non-IP protocols or Windows file sharing. On the other hand, setting up a VPN with routing can be easier and allows finer control over access to specific hosts and ports.

Many enterprises prefer to use cloud computing to extend the capacity of their IT infrastructure. To support this scenario, the VPN is configured via a gateway in the enterprise network to a private VLAN in the cloud. This is termed a site-to-site connection. The virtual machines on the cloud private network are not visible from the Internet but can be accessed only either via the VPN or via another virtual machine on the VLAN (see Figure 7.17).

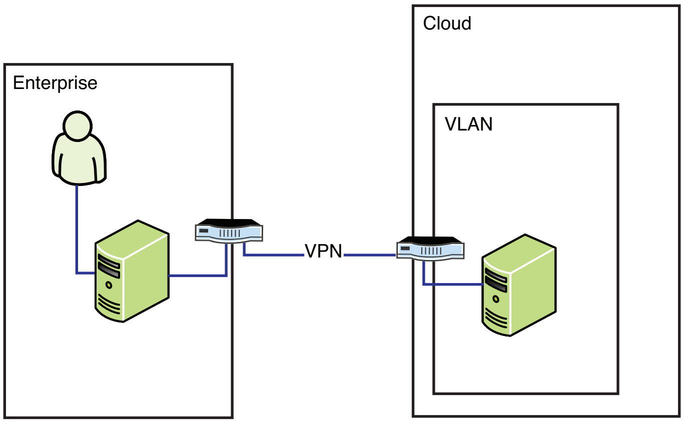


Figure 7.17 Use of a VPN to extend an enterprise network

In this way, the cloud acts as an extension of the enterprise network. The extension in the cloud is isolated from the Internet and from virtual machines in the cloud outside the VLAN. The IBM SmartCloud Enterprise supports this scenario as a fundamental offering. Unlike the work-from-home scenario, the employee does not have to do any special installation of a VPN client.

You can also support a network scenario that uses the cloud to support a VPN without needing an enterprise to own an on-premises VPN gateway. The CohesiveFT VPN-Cubed images in the IBM SmartCloud Enterprise catalog can provide a VPN gateway. This can support employees working from home with a VPN client, such as the OpenVPN client. Figure 7.18 shows this concept.

OpenVPN is an open source VPN client and server solution that can manage point-to-point and site-to-site connections. It uses the OpenSSL encryption library. The OpenVPN install image can be downloaded from the OpenVPN web site. It includes both client and server software and must be installed on both client and server machines. You can install using the RPM package on RHEL machines and using the `apt-get` command on SUSE or other Debian-based systems. It is possible to install on other Linux systems from the tarball using `make`. There is a self-extracting installer for Windows and also client-only install images that you can direct end users to.

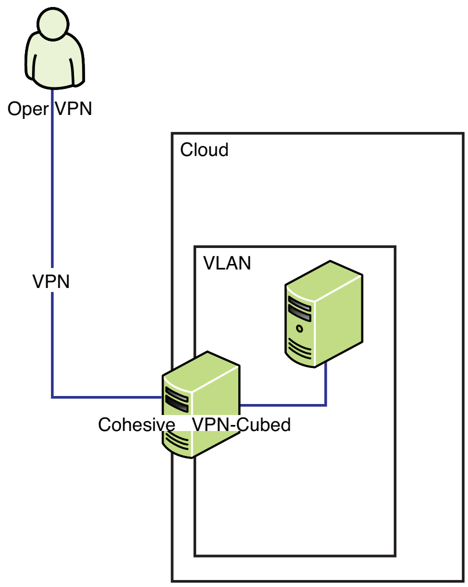


Figure 7.18 Use of VPN gateway in the cloud to access a VLAN

Setting up the VPN with Open VPN involves setting up a public key infrastructure, including a certificate with a public key, a private key for each server and client, and a certificate authority (CA) certificate to sign each of these certificates. By default, VPN does mutual authentication of both server and client using these certificates. However, you can configure alternate authentication methods. It is also possible to set up an Open VPN server on a dynamic IP address.

The VPN-Cubed solution can support many network configurations that can be useful in creating more complex virtual network configurations within a cloud and to connect computing resources over multiple clouds with a VPN. VPN-Cubed can act as a virtual router, virtual bridge, VPN concentrator, and firewall.

Browser Security

Browser security is important to consider when developing applications that are accessible on the Internet. In addition, some providers support extensible platforms that allow third parties to embed applications. For example, social networking platforms such as Facebook and IBM

SmartCloud for Social Business provide ways of embedding applications within their web sites. The end user sees a web page that consists of the combination of elements of the cloud platform and some embedded widgets that are provided by independent third parties. This is the opposite of a mash-up, in which a lightweight web site is created by embedding widgets from more powerful providers such as Google Maps or Yahoo! Search. In either case, web browsers place security restrictions on web browsers to protect users. One of the most important of these security restrictions is the same origin policy for JavaScript.

The same origin policy for JavaScript, as implemented in mainstream browsers, enforces that the JavaScript code in a HTML page can communicate only back to the server it came from. This must be the same host, same port, and same protocol. Table 7.2 shows some examples for a page loaded from `http://a.example.com/folder/index.html`.

Table 7.2 Result of the Same Origin Policy for JavaScript for the Page http://a.example.com/folder/index.html

URL	Result	Notes
http://a.example.com/folder2/index.html	Success	Only path is different
http://a.example.com/folder/innner/index.html	Success	Only path is different
http://a.example.com:8080/folder/index.html	Fails	Different port
https://a.example.com/folder/index.html	Fails	Different protocol
http://b.example.com/folder/index.html	Fails	Different host

The two solutions to the need to load content into a page from another server are as follows:

- Proxy the application content from the server of origin.
- Load the external content into an iFrame.

Using the server of origin as a proxy, a third-party server generates content, and the server of origin loads this into an area of a template page. Using this strategy, the browser sees all the content coming from the same server. Figure 7.19 shows a schematic of this solution.

Using an iFrame, the page provides a URL to a third-party server where the content is generated. By using the iFrame, the browser allows content to be loaded into the page, and it appears as if the content is integrated to the user.

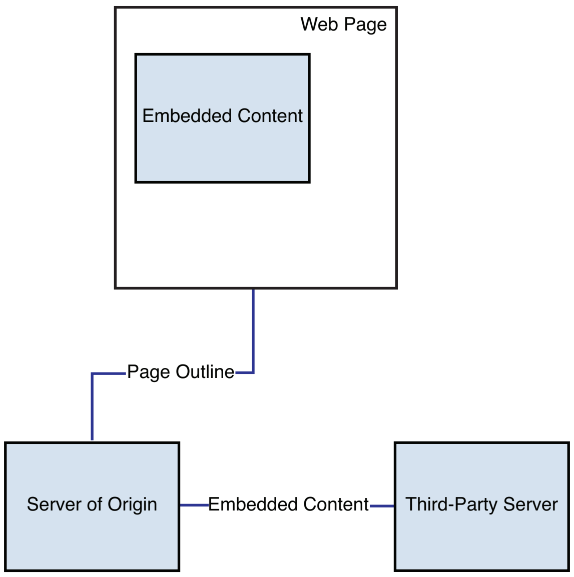


Figure 7.19 Embedding content into a web page using a proxy to a third-party server

Application Hardening

Application hardening refers to preventing malicious users from gaining unauthenticated access or having gained authenticated access, from using tricks to subverting authorization mechanisms. For example, malicious users can manipulate URLs, inject scripts, or misuse another user's session. The risks are serious because this can lead to theft of data.

IBM Rational AppScan® is a tool that can scan your application for vulnerabilities to these styles of attack and provide recommendations to address them. It scans all the pages in your application, sends different kinds of invalid data, and scans for weaknesses. It also examines port and protocol settings for poor practices.

Cross-Site Scripting

Many sites allow users to enter data that other users can view. For example, if a user registers at a site with her or his name, an administrator can view that information in a list of users. A cross-site scripting (XSS) attack adds a script to the information submitted so that the script is executed later when someone else views the data. For example, if an attacker registers and adds a JavaScript script to the end of his name, that script may be executed later when the administrator views the list of users.

Cross-Site Request Forgery

A cross-site request forgery (CSRF) attack is an attack method that forces an end user to execute unwanted actions on a web application to which he is currently logged on. An attacker can force a user of a web application to execute actions selected by attacker just with the help of social engineering tricks, such as via a link sent by email or chat software. When CSRF makes an attack directed to a user, data and operation instructions of the end user are threatened.

Consider an example in which a user logs into an online bank to check his balance of deposit and then goes to his favorite forum without exiting the online bank system. If the attacker has a well-constructed malicious link in the forum, the user's fund in the bank account can be transferred to an account the attacker specifies. For example, the forum might include an image tag with a link back to the bank web site:

```
<img src='http://bank/account/transfer-money?to=bad_guys_account'/>
```

This results in an HTTP `GET` request by the user back to the bank web site just by viewing the page; the fact that there is not really an image behind the web site does not matter. Other variations of CSRF exist, including these [Bath, 2008]:

- Leveraging the user's browser to send requests to servers behind a firewall. In this case, the user is behind an enterprise firewall and has access to servers that are not accessible from the Internet.
- Changing variables describing a browser's state, such as cookies and client certificates.
- Logging a CSRF attack. In this case, the attacker uses his own login credentials to log into a public service that is customized with some malicious personal customizations. This allows the attacker to collect information about the real user. Also, an attacker can force the login with the user's own credentials.

Different attack methods require different levels of sophistication. At the low end, a simple attack on a poorly designed web site can be done using basic HTML page design. At the high end, control on some point of the network that the user requests travel over is required.

CSRF attacks can be prevented in three ways:

- Generate a secret token and bind it to the user's session, making sure that it is received with each request.
- Check the HTTP Referrer header to make sure that the request came from your web site.
- Include a secret token in XMLHttpRequest. This applies to sites that use AJAX.

The most popular method is to include a secret token in every HTTP request that causes an update to the system, such as a fund transfer action. The secret token is maintained during the HTTP session and cannot be guessed before the HTTP session is established. A random number–generation algorithm can be leveraged for the token generation.

In this example, a code snippet in Java is used to generate a 32-bit secret token for your reference. It uses the JDK SecureRandom API.

```
private SecureRandom sr = new SecureRandom();

private String generateToken(){
      StringBuilder sb = new StringBuilder();
      for(int i=0;i<CSRFController.TOKENLENGTH;i++){
            int index = sr.nextInt(CHARSET.length);
            sb.append(CHARSET[index]);
      }

      return sb.toString();
}

private final static char[] CHARSET = new char[] {
      'A','B','C','D','E','F','G','H','I','J','K','L','M',
      'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
      'a','b','c','d','e','f','g','h','i','j','k','l','m',
      'n','o','p','q','r','s','t','u','v','w','x','y','z',
      '0','1','2','3','4','5','6','7','8','9'
};
```

An interceptor (in J2EE, it is called Filter) between the server and the browser is required to check the existence and correctness of the secret token for every HTTP request. Figure 7.20 shows the work flow of the interceptor.

SQL and Other Injection Attacks

Most web sites need relational databases and collect information from users using HTML forms saved in a database. A SQL injection attack uses HTML forms to submit data that attempts to break out of the contents of SQL statements and create their own SQL statements.

IBM Rational AppScan can also scan for this style of attack.

Secure Communication Protocols

The three secure protocols that we are most concerned with in cloud computing are Secure Shell (SSH), HTTPS, and IPSec. Each has its own use. SSH is a fundamental tool for developers and administrators, primarily used for remotely logging into a console. However, it has also become widely used to tunnel (in other words, carry embedded in) other protocols for file transfer and other purposes. HTTPS is the most common protocol for securing communications for application users over the web. IPSec is a protocol used to secure virtual private networks (VPNs) for all communications in general. It is useful to people working in locations remote from the computing resources they use. It usually requires special client setup. Although these protocols existed before cloud computing, we cover them because they are fundamental tools and have new relevance in cloud computing.

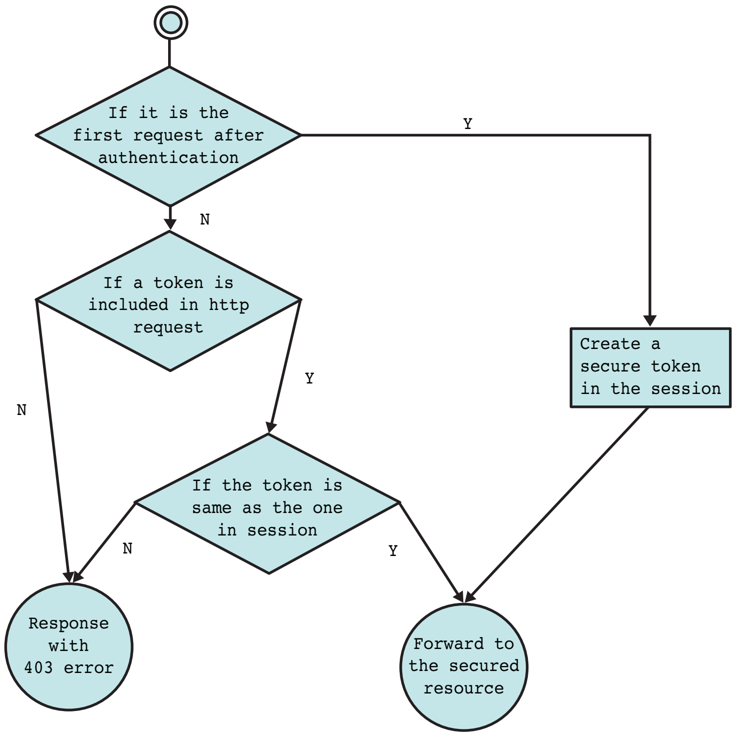

If it is the first request after authentication
Y
N
If a token is included in http request
Create a secure token in the session
Y
N
If the token is same as the one in session
N
Y
Response with 403 error
Forward to the secured resource

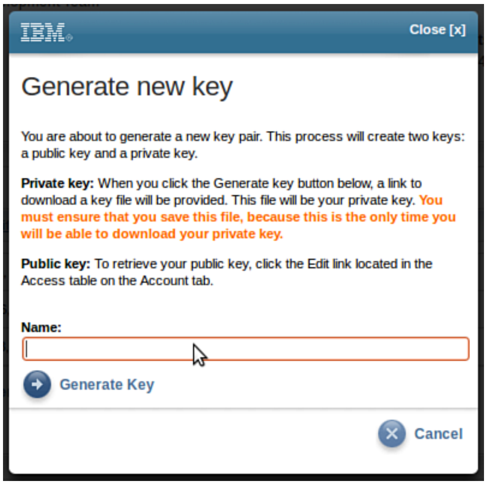
Figure 7.20 CSRF attack-prevention work flow of the interceptor

Secure Shell (SSH)

As you saw in earlier sections, SSH is a fundamental tool in cloud computing. It can be worth learning as a power user to solve numerous practical problems in cloud computing. SSH was designed as a secure replacement for Telnet but now is also commonly used programmatically for many applications. The two main versions of SSH are SSH-1 and SSH-2. The original version, SSH-1, has been superseded by SSH-2. SSH-2 was developed as a standard by the Internet Engineering Task Force and was released in 2006. The most popular implementation of SSH is the OpenSSH open source project. On Windows, PuTTY is the most popular client. See the section "Linux, Apache, MySQL, and PHP" in Chapter 2 for the basic use of PuTTY.

SSH uses public key cryptography to authenticate the remote computer and the user if necessary. Besides remote login, SSH can be used for tunneling, forwarding, X11 connections, and file transfer. Files can be copied with the SCP (Secure Copy) protocol and SFTP, which both use SSH.

Let's take a closer look at the use of keys and authentication with SSH. SSH is most often done with public–private key pairs. It can be configured other ways, but on the IBM Smart Cloud Enterprise, most Linux images are configured by default for authentication via public-private key pairs. You can manage the keys associated with your account via the Control Panel user interface using the Account, Generate New Key button. When you generate a key, you are prompted with a dialog box similar to the one shown in Figure 7.21.



You are about to generate a new key pair. This process will create two keys: a public key and a private key.

Private key: When you click the Generate key button below, a link to download a key file will be provided. This file will be your private key. **You must ensure that you save this file, because this is the only time you will be able to download your private key.**

Public key: To retrieve your public key, click the Edit link located in the Access table on the Account tab.

Name:

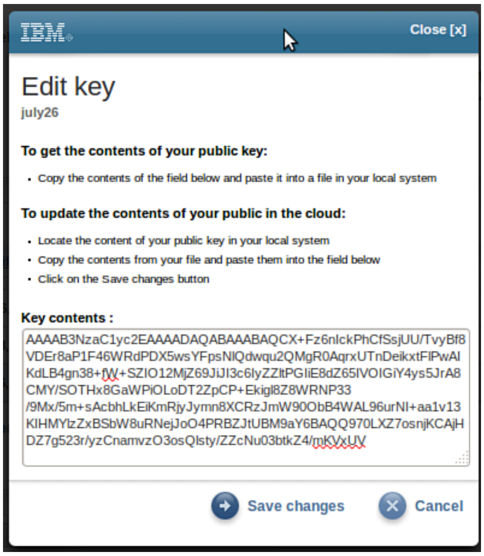
Figure 7.21 Generating a new key in the IBM SmartCloud Enterprise

When you click Generate Key, SCE generates a public-private key pair. You are given the option to save your private key. You should save it in a safe place; the cloud will not save it for you. However, the cloud will save the public key. You can also retrieve this in the Control Panel from the Account, Security Key Pairs table and clicking the Edit link. You should see something like Figure 7.22.

If you need your public key at any point, you can get it from this dialog box. Alternatively, if you prefer to use your own public key, you can generate it with OpenSSH or PuTTY and paste it here. The IBM SmartCloud Enterprise embeds the public key in the virtual machine so that you can SSH to it. Let's see what happens when you log in with the key you just generated. You saved it as a file named `july26_key` and changed the file permissions to none for group and others. First, provision a Linux image using the key. Then, using the OpenSSH client with the `-v` verbose option, you will see something like this output:

```
$ ssh -v idcuser@170.224.193.53 -i july26_key
OpenSSH_5.8p1 Debian-1ubuntu3, OpenSSL 0.9.8o 01 Jun 2010
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 170.224.193.53 [170.224.193.53] port 22.
debug1: Connection established.
...
debug1: Remote protocol version 2.0, remote software version
OpenSSH_4.3
```

```
...
The authenticity of host '170.224.193.53 (170.224.193.53)' can't be
established.
RSA key fingerprint is
7b:ce:14:93:c6:63:72:fa:27:6b:aa:d7:fa:c2:2a:80.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '170.224.193.53' (RSA) to the list of known
hosts.
debug1: ssh_rsa_verify: signature correct
...
debug1: Next authentication method: publickey
debug1: Trying private key: july26_key
debug1: read PEM private key done: type RSA
debug1: Authentication succeeded (publickey).
Authenticated to 170.224.193.53 ([170.224.193.53]:22).
debug1: channel 0: new [client-session]
debug1: Entering interactive session.
IBM WebSphere Application Server V7.0
Documentation ->
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=compass
&product=was-base-dist
Please switch to WebSphere Administrator user ID "wasadmin" before
executing WebSphere commands.
[idcuser@vhost0308 ~]$
```



Edit key

july26

To get the contents of your public key:

- Copy the contents of the field below and paste it into a file in your local system

To update the contents of your public in the cloud:

- Locate the content of your public key in your local system
- Copy the contents from your file and paste them into the field below
- Click on the Save changes button

Key contents :

```
AAAAB3NzaC1yc2EAAAADAQABAAABAQCX+Fz6nIckPhCfSsjUU/TvyBf8
VDEr8aP1F46WRdPDX5wsYFpsNlQdwqu2QMgR0AqrxUTnDeikxtFlPwAl
KdLB4gn38+fW+SZIO12MjZ69JiJI3c6lyZZltPGIiE8dZ65IVOIGiY4ys5JrA8
CMY/SOTHx8GaWPiOLoDT2ZpCP+Ekigl8Z8WRNP33
/9Mx/5m+sAcbhLkEiKmRjyJymn8XCRzJmW90ObB4WAL96urNI+aa1v13
KIHMYlzZxBSbW8uRNejJoO4PRBZJtUBM9aY6BAQQ970LXZ7osnjKCAjH
DZ7g523r/yzCnamvzO3osQIsty/ZZcNu03btkZ4/mKVxUV
```

Figure 7.22 IBM SmartCloud Enterprise Portal Panel for editing the SSH key

The `@` symbol in the `ssh` command invocation delineates the user name from the host name or IP address. The `-i` option specifies the name for the private key `july26_key`. This saves you from having to type it and is a good form to use in scripts. When the client begins, it prints the version of OpenSSL used. OpenSSL is an open source cryptography library used by OpenSSH. Next, it reads the client configuration file `ssh_config`. Many options in this file relate to security and protocol options. We discuss some of these here. Then the versions of software and protocol are printed out. Its remote protocol is SSH version 2.0. The client finds that it is not familiar with this server and asks whether you trust it. You answer yes, and the SSH client adds the server IP and RSA fingerprint to the list of known hosts. You can delete that later with the `-R` option of the `ssh-keygen` command. (We have deleted a few lines showing a number of authentication methods tried.) The final method tried is public key. It tries the private key file `july26_key`, supplied on the command line, which it finds to be an RSA type of key in the PEM format. RSA stands for Rivest, Shamir, and Aldeman, the three people who discovered this break-through cryptographic algorithm in the 1970s. The algorithm bases the public-private key pair on large prime numbers, which, when multiplied together, form a number so large that it is difficult to factorize, unless you happen to have the private key. The PEM (Privacy Enhanced Email, a name no longer meaningful, so we just use the acronym) file is a kind of certificate format that is Base 64 encoded. If you open the file, it is text and looks like this:

```
-----BEGIN RSA PRIVATE KEY-----
...
-----END RSA PRIVATE KEY-----
```

A number of alphanumeric characters show up in between. Other authentication methods are supported, in case you prefer not to use RSA.

Then you get a message that authentication has succeeded, a new channel has been opened, and a session begun.

Consider this common scenario: You create two instances on the cloud, server1 and server1, and you need to open an SSH session from server1 to server2. You should already have your private key on your local PC, so you need to copy it to server1. You can do this with WinSCP. Server2 already is configured to accept this key because it was created with the public key in the authorized key list. Execute these commands:

```
> chmod 0700 mykey
> ssh -i mykey server2
```

`mykey` is the name of your private key here. You should keep your private key from being visible to other users with the `chmod` command. If you do not do this, `ssh` complains and might ignore the key. The `-i` option uses the key that you just copied in the SSH session. The default is `~/.ssh/id_rsa`. The `server2` argument is the host name or IP address of server2. Make sure that you have the right permissions on the directory `.ssh`. If some of the files are owned by root, change them to idcuser with the `chown` command.

To generate a new SSH key, use the `ssh-keygen` command—for example:

```
> ssh-keygen -t rsa -P 'My Passphrase' -f ~/.ssh/mykey
```

This generates an RSA type (`-t` flag) with the passphrase `'My Passphrase'` (`-P` flag), places the private key in the file `~/.ssh/mykey` (`-f` flag), and places the public key in the file `~/.ssh/mykey.pub`. If you do not use the `-f` option, the private key is written to `~/.ssh/identity`. Authorized keys are contained in the file `~/.ssh/authorized_keys`. To add a new public key to the authorized keys file, append it with this command:

```
> cat mykey.pub >> ~/.ssh/authorized_keys
```

The SSH keys generated by IBM SmartCloud Enterprise are in a format compatible with Open SSH. In the section "Linux, Apache, MySQL, and PHP" in Chapter 2 you saw how to convert these to PuTTY format using puttygen. PuTTYgen can also convert back to OpenSSH format in case you lose the original key. To do this, go to the Conversions, Export OpenSSH Key menu and save the file. PuTTYgen can also generate keys. You can upload the generated public SSH key to SmartCloud Enterprise if you prefer not to use the keys the cloud generated.

The configuration file for SSH on the Linux systems on the IBM SmartCloud Enterprise is at `/etc/ssh/ssh_config` and `/etc/ssh/sshd_config`. The `AllowedUsers` setting is one setting that you can change. The value of this parameter is a space-separated list of user name patterns—for example:

```
AllowUsers idcuser webadmin
```

To start the SSH server (`sshd`), use this command:

```
# /etc/init.d/sshd start
```

To restart, use this command:

```
# /etc/init.d/sshd restart
```

The `-v` option (verbose) prints extra information and can be useful in debugging problems at the client end. For most of the images in the IBM SmartCloud Enterprise, by default, the SSH server sends informational and error messages to the file `/var/log/secure`.

It is good practice to add a passphrase to your SSH keys to protect them. The problem with this is that scripts and programs that need access to the keys do not have the passphrase. The ssh-agent tool, which is a part of the OpenSSH distribution, is designed to solve this problem. It can automate entry of the passphrase in a similar way so that a stash file can allow the web server to open the X.509 certificate, as described in the section "Example: Setting up HTTPS on the IBM HTTP Server" in this chapter.

A number of tools tunnel over SSH. For example, NX remote desktop technology, discussed in the section "NX Remote Desktop" in Chapter 6, "Cloud Services and Applications," tunnels over SSH. Several file transfer tools also tunnel over SSH.

Secure Copy, also known as SCP, is a protocol and a program to transfer files securely. It is based on BSD remote copy (RCP) command tunneled through SSH. As with SSH, by default, it runs on port 22. Basic use of SCP on Linux is shown here:

```
# scp -i identity_file host:sourcefile destinationfile
```

This uses the private key file `identity_file` to copy the file `sourcefile` from the machine `host` to the file `destinationfile` on the local machine.

OpenSSH also includes an SFTP (secure FTP) client and server. The Windows WinSCP program works by default but can also use SCP. This is probably the most convenient way to transfer files from a Windows client to a Linux virtual machine in the cloud.

Port Forwarding

Port forwarding with SSH involves this process:

1. The address and port of a packet are translated to a new destination.
2. The packet is carried over an SSH connection, where the destination is accessed.

SSH enables a user to tunnel another protocol over an SSH connection. With OpenSSH, this is done with sshd. It can be useful if the protocol being tunneled is not secure or the destination address and port combination is not visible from the origin. The client that uses the tunneled protocol must be capable of specifying a nonstandard port for this to work. The concept is that you establish an SSH session to your server and then specify which port on the client machine to forward connections from. SSH can also forward to other servers. Port forwarding can be important for solving connectivity problems with the cloud, your workstation, and IT resources inside your company. This is especially important when you need flexible but secure access for low-volume administrative tasks, such as moving files to secure locations. Knowing the tools well can avoid you having to write and maintain custom code for these administrative tasks. We look at several examples in this section.

Suppose that an administrative application is running on port 9080. Your company does not allow it to be exposed on the Internet, so the firewall is configured not to allow any Internet access on port 9080. You can access the administrative web application like using SSH, using the scheme shown in Figure 7.23.

The laptop will run an SSH client that forwards HTTP requests on port 9080 to the server, which will carry the data encrypted over SSH across the Internet and through the firewall. The server will then forward the requests to the web server on the virtual machine on port 9080. The ports need not be the same at either end, and the host that you access need not be the one on which the SSH server is running.

To enable SSH forwarding at the SSH server, edit the SSH server configuration file `/etc/ssh/sshd_config`, uncommenting the following line and setting the value to `yes`:

```
AllowTcpForwarding yes
```

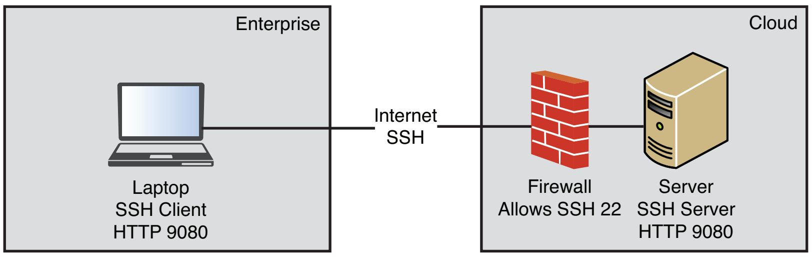


Figure 7.23 Port forwarding from a laptop to the cloud using SSH

Then restart the SSH server. You also need to enable forwarding on the SSH client. Edit the file `ssh_config`, uncomment this line, and set the value to `yes`.

```
Tunnel yes
TunnelDevice any:any
```

If you are using Cygwin, the file is at `{cygwin}\etc\defaults\etc`. Then you start an SSH session with the `-L` option, as shown:

```
ssh -L 9080:host:9080 idcuser@host -i .ssh/key-file -v
```

The `-L` option has the form `-L [bind_address:]port:host:hostport`, where `port` is the local port to be allocated, optionally bound to the `bind_address`; `host` is the host name of the remote computer; and `hostport` is the port on the remote computer. The local and remote ports in this example are both 9080. In verbose mode, you should see some lines like the following when you connect:

```
...
debug1: Local connections to LOCALHOST:9080 forwarded to remote
address 170.224.193.53:9080
debug1: Local forwarding listening on 127.0.0.1 port 9080.
debug1: channel 0: new [port listener]
debug1: channel 1: new [client-session]
...
```

Now open your web browser and point to the URL http://127.0.0.1:9080/admin, where `admin` is the path to your administrative application. The web user interface for your administrative application is shown. The SSH client listens to the local address on port 9080 and forwards to the remote host on the same port. In verbose mode, you will see messages about forwarded requests. All the web traffic will be encrypted because it is tunneled over SSH. You can use the same technique to forward to multiple port:host combinations with a single SSH session. This example uses the OpenSSH client, but it is equally possible with PuTTY.

The `openSSL -f` option requests SSH to go into the background before command execution. The `-Y` option (`-X` on older OpenSSH installations) does X forwarding to enable you to run graphical user interface programs from the remote machine using your local display. This can allow you to run xterms and other programs with X Windows graphical user interfaces. Perhaps a nicer way to do this is to run vnc over SSH to have a full desktop available. We did this from a Windows workstation with Cygwin installed to a Linux virtual machine on the cloud. First, configure both the SSH server and the client to allow X11 forwarding, as you did earlier. Start the VNC server and then start the SSH client with this command:

```
ssh -L 5901:host:5901 idcuser@host -i .ssh/key-file
```

Finally, open the vncviewer and enter `localhost:5901` as the address. You should see a Linux desktop displayed within a few seconds.

SELinux can potentially interfere with SSH port forwarding. Check `/var/log/messages` for any sign of this if you have trouble. SELinux is discussed in the upcoming section "Security-Enhanced Linux."

SSH can support more scenarios in addition to than those described here. For example, OpenSSH can also act as a SOCKS server by using the `-D` option for dynamic port allocation for forwarding. The `-w` option can be used to tunnel.

HTTPS

The HTTPS protocol is HTTP over Secure Sockets Layer (SSL) or its successor, Transport Layer Security (TLS). After establishing a TLS connection with the server, all traffic, including HTTP headers, are encrypted.

HTTP depends on public key cryptography, discussed in the section "Public Key Infrastructure and Certificates" in this chapter. In the most common mode, sometimes referred to as one-way SSL, the identity of the server is verified by a public certificate that is known to the client. This protects against eavesdropping. In a variation sometimes referred to as two-way SSL, the server also verifies the identity of the client with a client certificate.

You must obtain a certificate from a well-known certificate signing authority (CA), such as Equifax or Verisign. This can be costly if you have many servers, so you might consider issuing certificates with your own signing authority. You can use a certificate server such as OpenSSL. However, you will still need a certificate chain with a root certificate signed by a well-known certificate signing authority. For development server, you can use a certificate shipped with the server or sign your own certificates without needing to use the well-known CA.

In developing and hosting an application, your main challenge is obtaining certificates for your server and maintaining them, making sure that they do not expire.

Example: Setting up HTTPS on the IBM HTTP Server

The IBM WebSphere images in the SmartCloud Enterprise catalog come with the IBM HTTP Server (IHS) bundled. IHS is a customized version of the Apache HTTP Server with additions to make administration easier, more secure, and better integrated with WebSphere Application Server (WAS). Using IHS in front of WAS is a best practice, for security and performance reasons. If you browse to the IP of the virtual machine instance after it has been provisioned, you will see the default screen for this (see Figure 7.24).

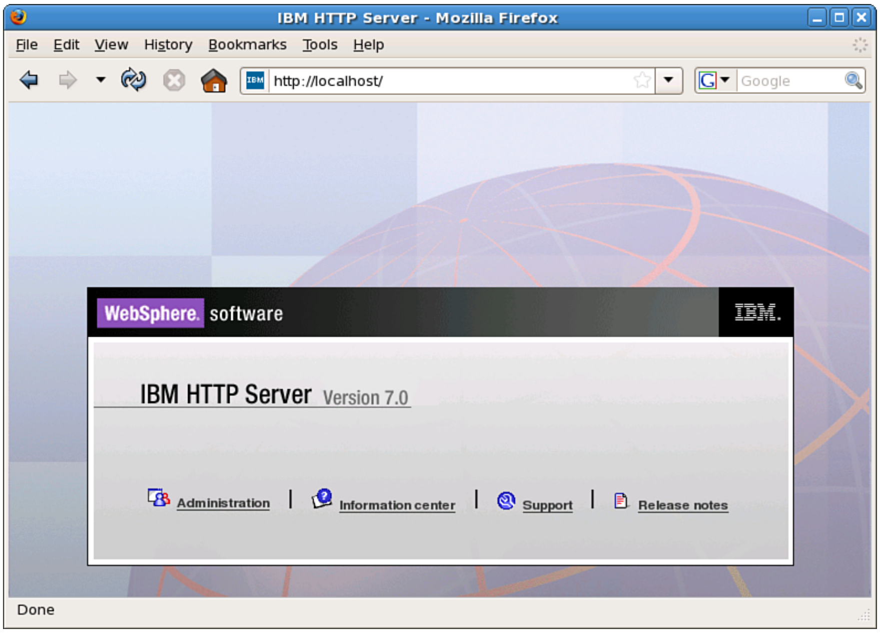
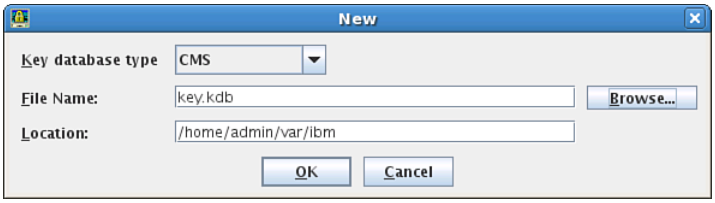

IBM HTTP Server - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/
Google
WebSphere. software
IBM.
IBM HTTP Server Version 7.0
Administration | Information center | Support | Release notes
Done

Figure 7.24 IBM HTTP Server default page

The page has useful links to product documentation. You can use the WebSphere Administration Console to manage IHS. You can use the IKEYMAN utility graphical user interface or command line to create a CMS key database file and self-signed server certificate. To start the graphical user interface, log into VNC as the WAS administrator, create a command shell, and enter this command:

```
# cd /opt/IBM/HTTPServer
# bin/ikeyman -x
```

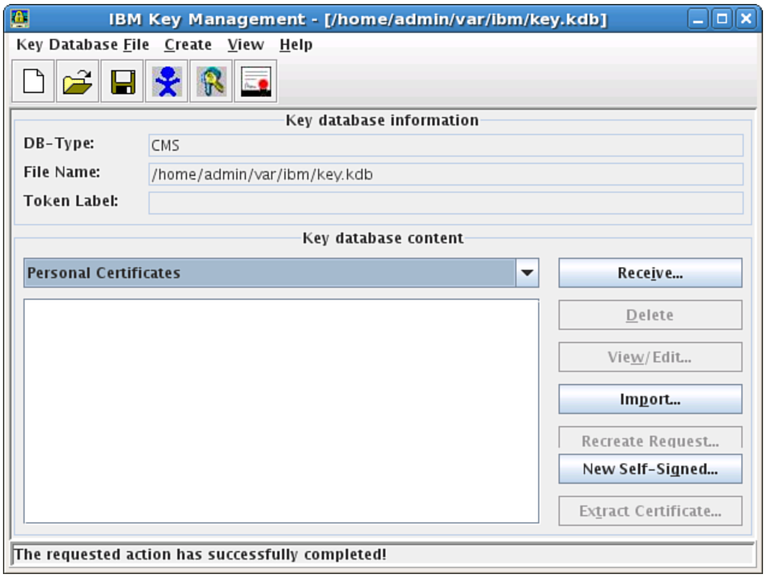
To create a new key database, click Key Database File, New on the menu; choose CMS for the type and enter values for the file name and location, as in Figure 7.25.



Key database type: CMS
File Name: key.kdb
Location: /home/admin/var/ibm

Figure 7.25 Creating a new key store in the iKeyMan key-management tool

After clicking OK and entering a password, choose to stash the password. IHS needs this to be able to read the keystore. The main console displays again, as in Figure 7.26.


IBM Key Management - [/home/admin/var/ibm/key.kdb]

Key database information

DB-Type: CMS

File Name: /home/admin/var/ibm/key.kdb

Token Label:

Key database content

Personal Certificates

The requested action has successfully completed!

Figure 7.26 Main console in iKeyMan key-management tool

To generate a self-signed certificate, select Personal Certificates in the drop-down menu and click the New Self-Signed button. Enter a key label for the certificate and choose arbitrary values for the other fields. Click OK.

Enable SSL by editing the file `/opt/IBM/HTTPServer/conf/httpd.conf`, uncommenting the `mod_ibm_ssl` directive, as shown here:

```
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
Listen 443
<VirtualHost *:443>
      SSLEnable
      SSLProtocolDisable SSLv2
</VirtualHost>
KeyFile /home/admin/var/ibm/key.kdb
SSLDisable
```

Restart IHS from the WAS console under Servers, Server Types, Web Servers. Open a browser and enter the URL https://host into the address bar. You should see a message that the certificate is not trusted. Messages vary between browsers. Add an exception and, in the process, view the certificate, checking that it is the same as the self-signed one you created earlier. You are now ready to do development with a self-signed certificate. When you are ready for production, create a Certificate Signing Request and send it to a well-known CA so that your users will trust it.

Nothing is wrong with the certificate you just created. You can use it for internal communication between servers, such as between IHS and WAS.

Internet Protocol Security (IPSec)

IPSec is a protocol suite that secures each IP packet of a communication session. The protocols include a method for mutual authentication and negotiation of cryptographic keys. IPSec operates at the Internet layer of the IP protocol stack, which means that applications do not need to be specifically aware of IPSec. In contrast, the security protocols HTTPS, SSL, TLS, and SSH operate at higher levels in the stack that do require applications to be aware of their use. One of the main uses of IPSec is in virtual private networks (VPNs).

Operating System and Virtual Machine Security

Operating system and virtual machine security is also an important security consideration in cloud environments. First, you are accessing the virtual machine over the network, very likely over the Internet, for administrative tasks. You need to be assured that you can do this in a secure way. Second, if you develop virtual machine images for others to use, you need to configure it in a way that is secure right from the moment it is created. Additionally, certain risks of operating in a shared virtualization environment exist that you should be aware of; you should also be aware of how to mitigate against these. The sections presented here discuss these topics.

Basic Operating System Tools

The basic operating system virtual machine images on IBM SmartCloud Enterprise are created by experts from Red Hat, Novell, and IBM with solid basic security defaults. One of these is to not allow remote login by root; another is for IBM to not retain any passwords or private keys. That means you need to save and remember your SSH private key to be able to log in and use your virtual machine instances. Of course, you need root access to the operating system for a number of reasons. After you log in with the idcuser account, you can effectively switch to root using the `sudo` command, like this:

```
> sudo /bin/bash
```

The `sudo` command executes a command as another user. In this case, the command you are executing is the bash shell. This demonstrates the enormous power of the `sudo` command. Access to `sudo` should be carefully controlled. The previous command is permitted because the idcuser account has been added to the `/etc/sudoers` file with the following line:

```
idcuser ALL=(ALL)       NOPASSWD: ALL
```

The first part of the line lists the user account name, followed by the list of permissions, followed by the password option. This gives the idcuser access to execute any command without a password. You can tighten this by creating a password for the idcuser account and then removing the `NOPASSWD` option in the `sudoers` file. Of course, once root is executing the bash shell, it can execute any command. You can restrict this by editing the `ALL` option, which requires the idcuser to use `sudo` for specific commands. The other permission options are listed in the comments in the `/etc/sudoers` file.

The initial security defaults are a good start, but you should tighten them for production applications. However, experimenting with the utilities described here can be dangerous. If you make a mistake or tighten security too much, you can lock yourself out. You might want to experiment with a virtual machine instance on the IBM SmartCloud Enterprise that has no critical data on it. If you mess up, just delete the instance and start with another one.

It is a best practice to require users to log in remotely using their own user names and use root sparingly. These steps can be useful in achieving this:

1. Remove the `/etc/securetty` file that lists the TTY devices that root can use to log in.
2. Set `PermitRootLogin no` in the SSH configuration file `/etc/ssh/sshd_config` to prevent root login with SSH.
3. Set the root's shell to `/sbin/nologin` in the `/etc/passwd` file.

If you are sharing an image with others, you should clean up your instance before saving it. Follow these steps:

1. Remove the shell history. On Linux instances, you can use the command `history -c`.
2. Remove the last login information by clearing `/var/log/lastlog`.
3. Remove the idcuser randomized password trigger file `/etc/cloud/idcuser_pw_randomized`.
4. Clear other logs and temporary files.

On Windows, the `SYSPREP` process used for virtual machine instance creation deletes log events and temporary files in the users' directory. Reboot the instance after you make system configuration changes with administration tools.

Security-Enhanced Linux

Security-Enhanced Linux (SELinux) is a feature that supports access control policies through the use of Linux Security Modules in the Linux kernel. It is an important tool for cloud providers to improve security in environments that many users share, and it is also a part of the virtual machines that you will use as a consumer of cloud services. It is closely related to the United States Trusted Computer System Evaluation Criteria (TCSEC, commonly called the Orange Book). SELinux enforces mandatory access control policies that confine system services and user programs to the minimum amount of privilege needed. It has no concept of a root user. The great advantage of SELinux is that is limits damage from attacks on programs that are not correctly designed and configured. SELinux is included with version 2.6 of the Linux kernel. The modules that make up SELinux have also been ported to other UNIX flavors. Red Hat Enterprise Linux and Novell (SUSE) provide commercial support of SELinux.

You can check the status of SELinux with the `sestatus` command—for example:

```
# /usr/sbin/sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted
```

SELinux assigns a context for every user or process that consists of three parts: the role, the user name, and the domain. Files, network ports, and hardware also have a SELinux context, which consists of a name, a role, and a type. Mapping between files and user contexts is called labeling. The `-Z` switch is added to shell commands, allowing the security context to be seen. For example, for files, you can use a variation of the `ls` command:

```
# ls -Z
-rw-r--r--  root root user_u:object_r:user_home_t      myfile.txt
drwxr-xr-x  root root user_u:object_r:user_home_t      myshare
```

The POSIX permissions are shown on the left, the SELinux context strings are shown in the center, and the file names are shown on the right. In SELinux, there are SELinux users as well as Linux users, which are mapped to SELinux users. In the first example in this section, `root` is mapped to the `user_u` SELinux user for file permissions. In the second example, `root` is mapped to the `system_u` SELinux user. For processes, you can use a variation of the `ps` command:

```
# ps -efZ | less
LABEL                        UID   PID  PPID  C STIME TTY TIME     CMD
system_u:system_r:init_t     root  1    0     0 09:19 ?   00:00:00 init [3]
system_u:system_r:kernel_t   root  2    1     0 09:19 ?   00:00:00 [migration/0]
system_u:system_r:kernel_t   root  3    1     0 09:19 ?   00:00:00 [ksoftirqd/0]
...
```

This shows the process label on the left, the standard process information in the center, and the command name on the right.

In SELinux terminology, files, directories, and devices are referred to as objects; processes are referred to as subjects. Linux and most distributed operating systems use discretionary access controls—that is, users control the access to objects they own. This opens an opportunity for malicious code to take advantage of users who give too much permission to the data they own. In SELinux, as in mainframes, a mandatory access control policy is used where the users do not control this access; instead, a system-wide policy controls the access and protects against malicious code taking advantage of user permissions to alter applications or bypass system security. These mandatory access rules are checked after the regular Linux discretionary access rules.

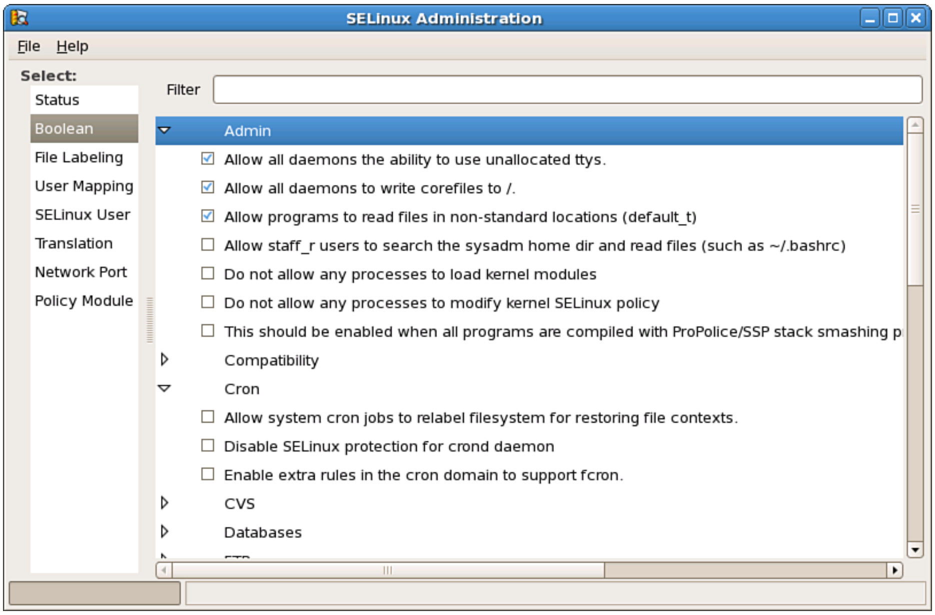
All files and processes are labeled with a type. A type defines a domain for the process, which separates processes by confining them to their own domains. Policies define how processes interact with files and with each other. These policies can prevent an attack that uses a process to access files that have wide access permissions. For example, with only discretionary access control, if an attacker can gain control of a HTTP server, he or she may be able to read files that have world-readable file access. The SELLinux mandatory access controls can prevent this.

The `semanage` command can be used to see the mapping between Linux users and SELinux users—for example:

```
# /usr/sbin/semanage login -l

Login Name          SELinux User        MLS/MCS Range
__default__         user_u              s0
root                root                SystemLow-SystemHigh
```

Relatively few SELinux users exist in the out-of-the-box configuration. Most Linux users are mapped to the SELinux user `user_u`.

Configuration files are kept in the directory `/etc/selinux`; the main configuration file is `/etc/selinux/config`. Graphical tools can help with the administration of SELinux. See Figure 7.27 for a screen shot of the SELinux Administration tool on RHEL.



Admin
- ☑ Allow all daemons the ability to use unallocated ttys.
- ☑ Allow all daemons to write corefiles to /.
- ☑ Allow programs to read files in non-standard locations (default_t)
- ☐ Allow staff_r users to search the sysadm home dir and read files (such as ~/.bashrc)
- ☐ Do not allow any processes to load kernel modules
- ☐ Do not allow any processes to modify kernel SELinux policy
- ☐ This should be enabled when all programs are compiled with ProPolice/SSP stack smashing p

Compatibility

Cron
- ☐ Allow system cron jobs to relabel filesystem for restoring file contexts.
- ☐ Disable SELinux protection for crond daemon
- ☐ Enable extra rules in the cron domain to support fcron.

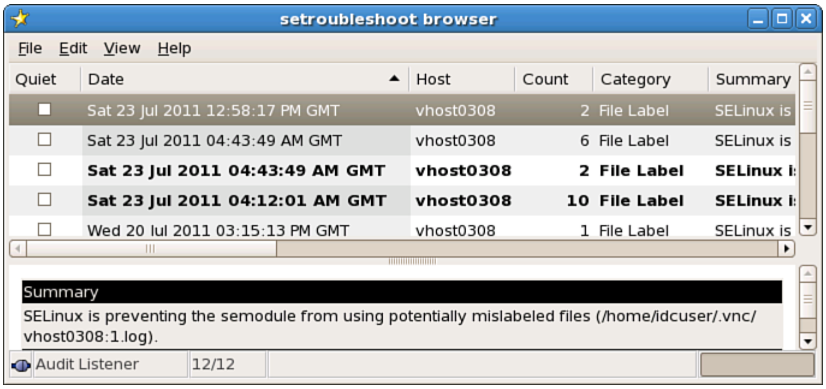
CVS

Databases

Figure 7.27 SELinux Administration tool on RHEL 5.4

Figure 7.27 shows the Boolean settings for SELinux, including settings for many services. You can manage the Boolean settings via the `setsebool` command.

If you have trouble with SELinux, check the `/var/log/messages` file for SELinux-related messages. In addition, you can use the SE Troubleshoot Browser, shown in Figure 7.28.



Quiet	Date	Host	Count	Category	Summary
☐	Sat 23 Jul 2011 12:58:17 PM GMT	vhost0308	2	File Label	SELinux is
☐	Sat 23 Jul 2011 04:43:49 AM GMT	vhost0308	6	File Label	SELinux is
☐	**Sat 23 Jul 2011 04:43:49 AM GMT**	**vhost0308**	**2**	**File Label**	**SELinux i:**
☐	**Sat 23 Jul 2011 04:12:01 AM GMT**	**vhost0308**	**10**	**File Label**	**SELinux i:**
☐	Wed 20 Jul 2011 03:15:13 PM GMT	vhost0308	1	File Label	SELinux is

Summary

SELinux is preventing the semodule from using potentially mislabeled files (/home/idcuser/.vnc/vhost0308:1.log).

Figure 7.28 SE Troubleshoot Browser

If you are logged into the Linux desktop as root, you get a visual notification warning about SE Linux events.

Some software might not run well with SELinux. You can set SELinux to permissive mode by using this command:

```
# setenforce 0
```

The `setenforce` command modifies the mode that SELinux runs in. In permissive mode, SELinux only logs actions; it does not enforce them. To disable SELinux, edit the file `/etc/selinux/config` and set the `SELINUX` line to `disabled`.

```
SELINUX=disabled
```

Then reboot the system. If you decide to re-enable it later, you will have to relabel the system. If you are simply trying to see if SELinux is interfering with a service, set it to permissive mode instead of disabling it.

A common problem with Apache involves labeling some of the files needed for operation of the web server. You might see something like the message "`httpd: bad user name apache`" when trying to start httpd. This is related to SELinux. To fix this, you can use this command:

```
/sbin/fixfiles restore
```

Security of Data at Rest

Data at rest refers to data in a state other than being transported over a network (data in flight). In this context, security most likely is a last line of defense. A number of principles are related to data security:

- **Confidentiality**—Ensure that only authorized people have access to the data
- **Integrity**—Ensure that the data is not tampered with
- **Availability**—Ensure that the data is available when it should be
- **Authorized use**—Ensure that the data is used only for an authorized purpose

Security of data at rest can be expensive and time consuming to provide—and more so to prove. For this reason, data classification is important. Data classification can help you focus on security in areas where it is most important. **Personally identifiable information** (PII) is data that identifies an individual person, such as the combination of a person's name and physical address. **Business confidential information** is information that an enterprise desires to keep confidential for business reasons.

Considerations for security of data at rest that we have not covered in previous sections include the following:

- Erasing data properly
- Encrypting data when needed
- Auditing access and changes to data when needed
- Ensuring security of backups
- Addressing access control

Erasing or shredding data properly is important for sensitive data. When the data that you are managing is not your own, you need to treat it all as potentially sensitive. When you delete a file, most operating systems simply disconnect the inode from the file system. A malicious user with access to the block device could potentially scan the individual blocks and reconstruct deleted files. Shredding files can avoid this and be achieved by zeroing out the file before deleting it.

Data encryption might also be necessary, depending on the nature and location of the data. Java and other programming languages provide libraries to do this encryption; commercial products also can handle the encryption. For example, Pretty Good Privacy (PGP) is an application that encrypts and decrypts data and is commonly used for documents, files, and whole file systems. It follows the RFC 4880 OpenPGP Message Format standard.

Security Events

Security events come in many shapes and forms. We have already mentioned some types in previous sections. You can distinguish event sources and event targets. For example, in a denial-of-service attack, the problem might originate at a server hosted by an ISP (which is the source) and might target a web site (which is the target). If the protocol of the attack is HTTP on port 80, we would say that the web server was the target of he attack. If the event were a port scanning event, we would say that the operating system was the target. These events are usually logged to event data stores, which could be files or databases. It is good practice to monitor these event logs regularly. In fact, it might be necessary for compliance reasons.

The basic security log in Linux is usually located in `/var/log/secure`. It contains entries similar to these:

```
...
Apr  3 20:00:03 vm-10-200-7-154 sudo:  idcuser : TTY=pts/0 ;
PWD=/home/idcuser ; USER=root ; COMMAND=/bin/bash
Apr  3 23:09:16 vm-10-200-7-154 sshd[27090]: Accepted publickey for
idcuser from 192.168.254.1 port 22356 ssh2
...
```

The first entry indicates that the user `idcuser` executed the `sudo` command `/bin/bash`. The second line indicates that the SSH service accepted the public key for user `idcuser` and lists the IP address the login came from.

Because of the large volume of potential security events and the number of possible different sources, a large data center needs a way to automate analysis of these events. IBM Tivoli Security Operations Manager (TSOM) is a product that gives you visibility into and control over events from many event sources, and targets and automates response. TSOM comes with a built-in set of rules to determine which events are security related and assigns a score to each event indicating severity. TSOM can also correlate security events across many different sources and targets. For example, if a worm affects many machines, these events can be automatically correlated in real time. The events display on a dashboard, reports can also be generated, and email alerts can be sent. The event information can be discovered remotely via network protocols or using agents to extract information from log files or other data stores with no remotely accessible protocol. Four different correlation engines exist: statistical, rule-based, susceptibility, and vulnerability. The events go through four stages of processing: aggregation, normalization, contextualization, and correlation. Event sources can be weighted depending on the source, the target, and the event type. For example, an attack on a web server running Common Gateway Interface (CGI) scripts might be weighted heavily because of its high vulnerability. TSOM has adapters for security events for firewalls, routers, operating systems, web servers, databases, network management systems, and other applications.

Security Compliance

Different compliance certifications usually apply based on the industry or business context you are operating within. Some of these include the following:

- Sarbanes-Oxley, for companies listed on U.S. stock exchanges
- Payment Card Industry (PCI), for e-commerce

- HIPPA, for health-related industries
- Basel II, for banking
- Common Criteria
- FIPS, for U.S. government

Most of these compliance regulations are not limited in scope to security, but security is an important part. In this context, a control is a method for ensuring the security and integrity of data. The security techniques discussed earlier in this chapter can form important security controls for compliance purposes. Compliance for banks and other large enterprises can require specialists and whole departments to ensure compliance for the enterprise.

Business Scenario: IoT Data Security Architecture

This section gives a high-level description of our example company, IoT Data, in its approach to security.

IoT Data will store all user identity information in a central OpenLDAP directory server. The schema will be customized to include a tenant ID, identifying which customer each user belongs to. This ID will also be included in all database tables so that we can trace each data record to a customer. Basic authentication over HTTPS will be used for the user interface and REST APIs.

All systems are protected by firewalls. We described the data flows and firewall configuration in the section "Business Scenario: Network Deployment and Firewall Rules," earlier in this chapter. IoT Data also needs development and test environments. These should not be exposed to the Internet, in case a competitor accidentally becomes aware of the URL of the test and development systems. To achieve this, all test and development servers will be located on a VLAN and access provided by VPN. An instance of a CohesiveFT image will be used as the VPN gateway, and the Open VPN client will be used to connect to it.

Except for a limited public area of the main web site, all communication to and between systems uses some form of encryption. During and after login to the user interface, all communication will be over HTTPS. All REST APIs will operate over HTTPS. The certificates for the public web servers will be signed by well-known certificate authorities. To reduce cost, all other certificates will be self-signed.

IBM Rational AppScan will be used to guide application hardening during the development phase. Penetration testing (ethical hacking) will be used to ensure that cross-site scripting, cross-site request forgery, and injection attacks cannot be performed using known techniques.

All database and system log backups will be stored encrypted. All data deleted by customers will be shredded so that it cannot be recovered by a third party.

All operating systems will be Linux. The standard security defaults will mostly be maintained by tightening them somewhat. `sudo` commands will require passwords.

IBM Tivoli Security Operations Manager (TSOM) will be used to monitor all security events and will display a summary in a central location.

CHAPTER 8

Performance, Availability, Monitoring, and Metering

When you have your service developed and available on the cloud, you need to monitor it to avoid problems. In particular, you need to make sure that it is performing well and stays up. One of the main differences in performance, availability, and monitoring in cloud computing is the evolving role of the developer, which currently demands more involvement in the overall system lifecycle. However, other differences also arise compared to traditional systems:

- Virtual nature of all system resources
- Relatively constrained set of choices for compute resources
- Dynamic capability to add more resources
- Greater dependence on the network
- The dynamic nature with which resources can be created and destroyed

We also discuss metering, which is one of the basic principles of cloud computing and operates with cost based on measured use. Just as you pay for the electricity that you use by the kilowatt and your telephone bill is based on the number of minutes used, cloud services are based on resource usage.

Performance and Scalability

One of the great promises of the cloud is that it can allow enterprises to scale out instead of scaling up. In scaling up, if you have too much demand on an application, you can buy a bigger server to host your application. The problem with this is that you need to migrate the data from the original, smaller server to the bigger, more capable server. In scaling out, you can add another server to share the workload. Setting up load balancing and failover is complex and involves a lot of work. For certain common classes of applications, this is a benefit that Platform as a Service (PaaS) can add if your application can migrate to or be built on it.

Compute Capacity

Virtual memory is a concept that was developed to allow each process in a multitasking system to have its own unique memory space. Virtual memory enables a developer to think about a program as if there is only one kind of memory. However, when we get to system performance, we need to consider the different kinds and uses of memory. **Swap** is an area of disk that is used as part of the virtual memory area. Swap extends the physical memory, allowing the virtual memory to be much larger than the physical memory. **Resident memory** is the part of a process space that is resident in physical memory. A **page** is a contiguous block of memory. **Paging**, or **swapping**, is the process of moving a page from physical memory to swap.

Thrashing occurs when excessive swapping takes place between physical memory and swap disk, and is a common cause of performance problems in virtual machines. Especially when using virtualization on-premises, it is tempting to set up more virtual machines than the hardware can support. Thrashing usually is easily diagnosed using performance-monitoring tools, such as top, by checking metrics such as virtual memory, physical memory, memory utilization, and resident memory.

One of the great aspects of cloud computing is that you can add compute capacity dynamically. For Infrastructure as a Service, you can do this using the APIs that we discussed in Chapter 3, "Developing with IBM SmartCloud Enterprise APIs." However, to know when to add compute capacity, you need to use the performance-monitoring tools discussed in the "Monitoring and Metering" section of this chapter. To make use of the additional compute capacity, you will to do some form of load balancing. One of the most common ways to do load balancing is to use a reverse proxy. In the "Proxy Servers" section of Chapter 7, "Security," we discussed the use of proxies for security purposes.

Recently, more application capability has been pushed to execute with JavaScript on users' browsers, and data is transmitted with AJAX. This can improve the perceived responsiveness of an application. Now we need to consider the browser performance.

Network Performance

Network factors greatly influence the performance of a cloud computing application as users see it. Network infrastructure within a cloud data center is beyond the scope of the this book—and probably beyond the scope of your influence as an application developer. However, you can take many steps as an application developer to improve the performance of your application across the network. Primarily, you can (1) know the characteristics of your network and your users' access to it, (2) design your application to stay within these limits, and (3) make optimal use of the global cloud network infrastructure to reduce latency. Monitoring of network performance is discussed in the "Network Monitoring" section of this chapter.

Network bandwidth is a measure of the capacity or maximum throughput of a network to transmit data, measured in bits per second. Table 8.1 lists the typical network bandwidth for different network media.

Table 8.1 Network Bandwidth Rates

Connection Technology	Bandwidth
Dialup Modem	56 kbps
ADSL Lite	1.5 Mbps
T1	1.54 Mbps
Ethernet	10 Mbps
Wireless 802.11b	11 Mbps
T3	44.7 Mbps
Wireless 802.11g	54 Mbps
Fast Ethernet	100 Mbps
OC3 (optical)	155 Mbps
OC12 (optical)	622 Mbps
Gigabit Ethernet	1 Gbps

Keep in mind that for most of the cases in Table 8.1, users share the available bandwidth with other users, so actual bandwidth is less. Also remember that the network path from the cloud data center to a user's browser traverses a number of different network media, each with different bandwidth and shared among a different set of users. However, the numbers give you an idea of the expectations you can have with different network segments.

Network latency is the one-way or round-trip time taken for a network packet to be sent from a source to a destination. Round-trip network latency is most often quoted. The most commonly used method of measuring network latency is ping, which measures round-trip latency. Because ping packets are small, they can be quickly relayed between different network nodes and give an optimistic measure of network latency, compared to the network data packets that applications users will experience.

Many cloud applications are web applications, which means that the user interface elements, such as HTML, JavaScript, and images, must be loaded into a browser over the network along with the data presented to the user. Application developers should be mindful of the size and total amount of these elements because they can greatly affect the time it takes a page to load.

Because of network latency, applications that are "chatty," or require many small network transmissions, can perform poorly over Internet connections that are remote from the cloud data center (for example, users in Asia who are accessing applications running in a data center in the United States). Web applications with many small graphics, JavaScript files, and AJAX calls are prone to this problem because the network latency magnifies the page load time. Loading of the individual page elements might not be as optimized as you hope and varies considerably by browser type.

Users in different areas of the world do not experience the same levels of service. The IBM SmartCloud Enterprise has seven data centers distributed globally to serve customers around the world. The simplest and most obvious way to make use of this capability is to host your application geographically close to where you think most of your users are. If you have users spread throughout the world, another approach is to distribute your application globally across different data centers. The feasibility of doing this depends on the complexity of your application. In the simplest case, your main page could give users the choice of geographic location. This might also be connected with other globalization aspects of your application, such as sales and billing. A more sophisticated approach is to detect the location of the user and forward him or her to the optimum delivery point.

J2EE Application Performance and Scalability

J2EE application servers have a number of features that assist with achieving performance and availability goals. One of the main features is the capability to cluster application servers to distribute load. A **cluster** is a group of application server nodes or individual application servers that work as a group to balance workload. In cloud computing, you have the opportunity to create virtual machines dynamically, and these can act as nodes in the cluster. This section focuses on WebSphere Application Server (WAS).

A **stand-alone server** provides a platform to run an application in an isolated environment. This is efficient for applications supporting small workloads. However, applications that support large workloads where a cluster is needed must have a way of organizing and managing the parts of the cluster. In WebSphere, a **node** is usually a physical server or virtual machine with software for one or more application servers installed. A **cell** is a logical group of servers or clusters. Applications can be installed in a cluster in a single operation, and the cluster can be started and stopped as a single unit.

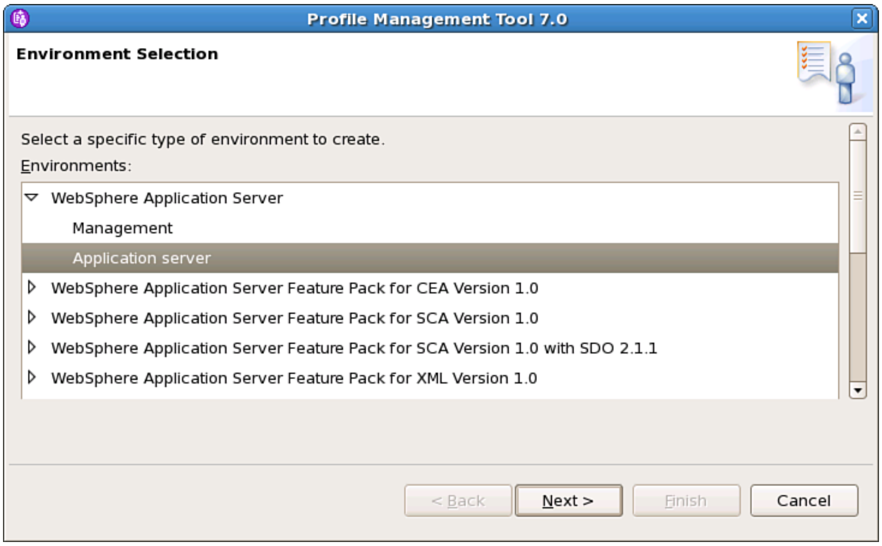
If you connect via SSH to the WAS server created in Chapter 2, "Developing on the Cloud," you can see three directories in the WAS installation directory `/opt/IBM/WebSphere`:

```
[wasadmin]$ ls
activation  AppServer  Profiles
```

The `AppServer` directory contains all the core product files. The `Profiles` directory is the default directory for creating profiles. We discussed provisioning of WAS in the section "WebSphere Application Server" of Chapter 2, where we recommended use of the Development profile in developing a J2EE application. As discussed, a WAS **profile** defines a runtime environment. Creating a standard profile for your needs can help in managing a number of WAS installations, compared with creating a new profile each time you install WAS. Profiles can be managed with the `manageprofiles` command-line tool or with the Profile Management Tool graphical user interface. To create a new profile in WebSphere, first launch the Profile Management Tool with this command:

```
[wasadmin]$ /opt/IBM/WebSphere/AppServer/bin/ProfileManagement/pmt.sh
```

When you start the tool, your existing profiles are listed. To create a new profile, click the **Create** button. You should see something like Figure 8.1.



Select a specific type of environment to create.
Environments:

```
▽ WebSphere Application Server
    Management
    Application server
▷ WebSphere Application Server Feature Pack for CEA Version 1.0
▷ WebSphere Application Server Feature Pack for SCA Version 1.0
▷ WebSphere Application Server Feature Pack for SCA Version 1.0 with SDO 2.1.1
▷ WebSphere Application Server Feature Pack for XML Version 1.0
```

Figure 8.1 Creating a new profile with the WebSphere Application Server Profile Management Tool

Follow these steps:

1. Select Application Server and click Next.
2. Select Typical Profile Creation on the next screen and click Next.
3. On the next screen, type a user name and password and click Next again.
4. Make a note of the settings on the confirmation screen and click Create.

The Advanced Profile Creation option enables you to deploy or not deploy the administration console and sample applications, select performance tuning options, configure additional security settings, configure port settings, and create a web server definition.

You can manage cells, clusters, and nodes in the administrative console. Adding a node to a cell is known as **federation**. A node is a group of managed servers. A **managed node** has been federated into a cell or registered with an **administrative agent** or a **node agent**. An administrative agent can manage multiple application servers on the same machine. In the WebSphere Application Server Network Deployment model, a single administrative console is the central point for all the application servers that it manages. A **deployment manager** manages nodes in a cell. WAS 7.0 introduced the concept of a **job manager** to better manage cells across distant geographic locations. Because administrative calls across the Internet can take a considerable amount of time, synchronization problems could occur with topologies spanning distant locations without the use of a job manager. The job manager queues administrative requests to make sure they are delivered successfully. This flexible administrative model has important applications if you want to deploy your application across multiple cloud data centers to ensure availability in the event of a disaster at one location. Figure 8.2 illustrates this concept.

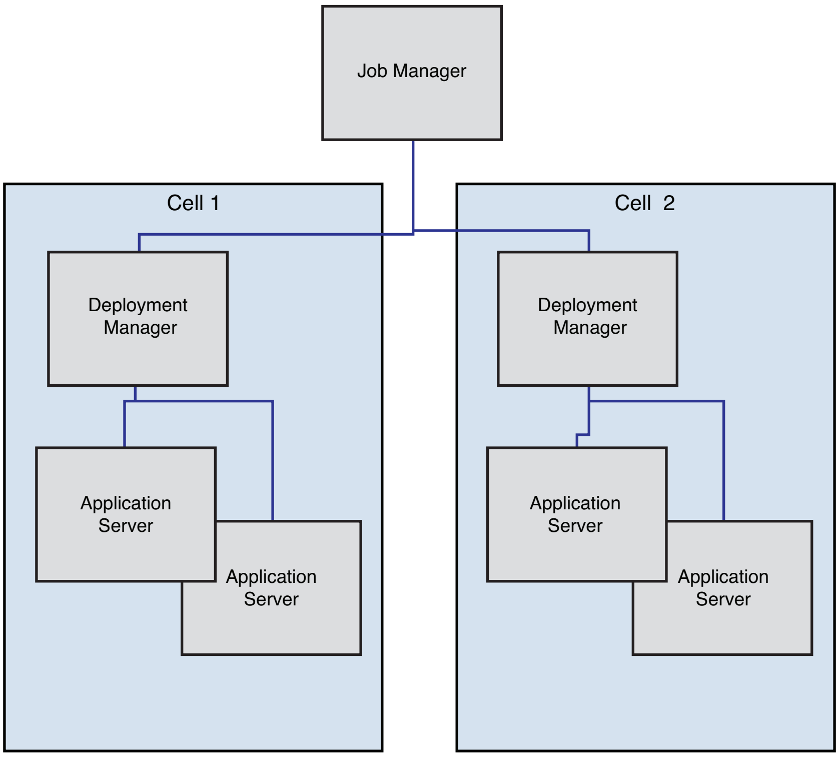


Figure 8.2 WebSphere network deployment management concepts

You can create deployment managers, administrative agents, and job managers with the Profile Management Tool. To create a new Administrative Agent Profile with an administrative console, launch the Profile Management Tool as described. However, instead of choosing Application Server, as before, choose Management. Choose Typical Profile Creation at the first screen and enter a user name and password at the next screen. Finally, confirm the profile information and click Create.

All the steps that you can perform in the administrative console can be scripted with the wsadmin tool and commands, including Java programs with the Java Management eXtensions (JMX) and Jython scripts. You can combine this knowledge of the WebSphere network deployment model with your knowledge of creating and customizing images and what you learned about the IBM SmartCloud Enterprise REST API to automate provisioning of elements of a WebSphere cluster. You can do this with the following steps:

1. Starting with a base WebSphere Application Server image, customize the profile with the Profile Management Tool and the application sever with the administrative console.
2. Save the virtual machine to an instance with the SCE user interface or REST API.
3. Use the SCE REST API to provision new virtual machines.
4. Use the wsadmin tool to join the new virtual machines to the cluster and synchronize application deployment across the nodes, including deploying your J2EE application.

Performance Analysis and Testing

Performance analysis and testing is closely related to application development. Usually, performance analysis is the responsibility of the development group and performance testing is the responsibility of the test group. IBM Rational Software Architect provides graphical application profiling for both local and remote processes. Profiling remote processes is especially useful to analyze the performance of J2EE web applications.

To profile a Java application in RSA, switch to the Profiling and Logging perspective from the Window, Open Perspective menu, as shown in Figure 8.3.

In the Navigator tab, right-click your application and click Profile As from the context menu. You are presented with a configuration dialog, as shown in Figure 8.4.

You can filter the amount of data collected to avoid overloading and slowing the RSA client. Click Edit Options and add a filter to include the package you are testing with a * after it. Click Apply and Profile to profile your application. For the execution time analysis, you should see something like Figure 8.5.

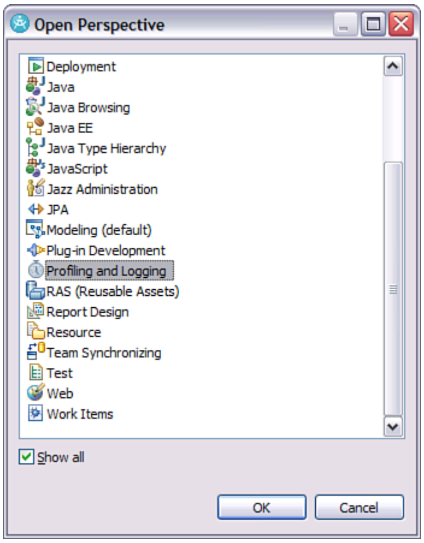
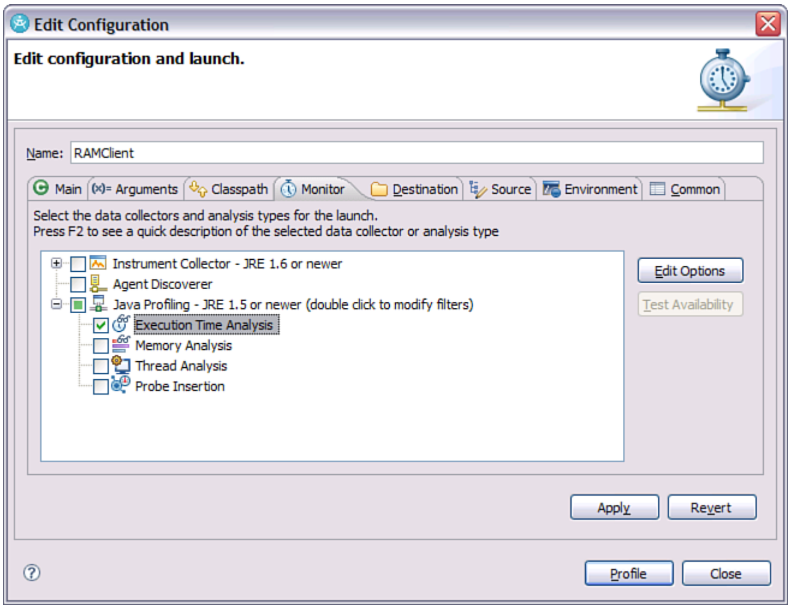


Figure 8.3 Opening the Profiling and Logging perspective in RSA


Edit Configuration

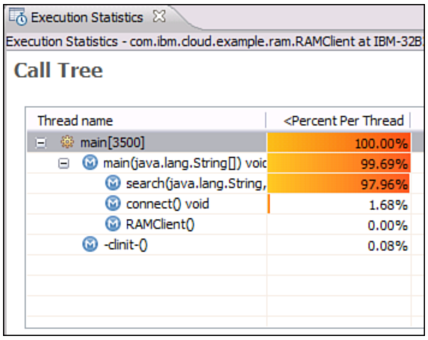
Edit configuration and launch.

Name: RAMClient

Select the data collectors and analysis types for the launch.
Press F2 to see a quick description of the selected data collector or analysis type

- [ ] Instrument Collector - JRE 1.6 or newer
- [ ] Agent Discoverer
- Java Profiling - JRE 1.5 or newer (double click to modify filters)
  - [x] Execution Time Analysis
  - [ ] Memory Analysis
  - [ ] Thread Analysis
  - [ ] Probe Insertion

Figure 8.4 RSA Edit Configuration dialog box


Execution Statistics
Execution Statistics - com.ibm.cloud.example.ram.RAMClient at IBM-32B
Call Tree

Thread name	<Percent Per Thread
main[3500]	100.00%
main(java.lang.String[]) voic	99.69%
search(java.lang.String,	97.96%
connect() void	1.68%
RAMClient()	0.00%
-clinit-()	0.08%

Figure 8.5 Example of RSA profiling execution time analysis

This figure shows that about 98% of the time is spent in the `search()` method.

To profile a J2EE web application running in WebSphere Application Server, start the server from the RSA Servers console in profile mode. If the WAS server is on a remote machine, you must first install the IBM Agent Controller, which you can freely download from IBM.com. Then you must set up environment variables, as explained in the documentation for the Agent Controller.

Performance analysis and tuning of relational databases is a topic by itself. SQL is a declarative language, in the sense that the data to be retrieved or changed is described in the statement, but the algorithm to perform the query or operation is not described. The statement can be implemented by a relational database in many ways, each of which is called an access path or access plan. These can result in considerably different performance associated with different access paths. IBM Data Studio and IBM Optim™ Query Tuner have tools for optimizing access paths, including graphical visualization capability. DB2 also has its own native tools for access path tuning.

The goal of performance testing is to verify that your application performs acceptably under a variety of scenarios and conditions. In contrast with performance analysis, performance testing covers a wider range of test cases. Performance testing usually focuses on a metric that is closely aligned with user experience, such as page load time. In addition, a goal of performance testing might be to ensure that performance does not degrade from release to release. For this reason, it is important to be able to repeat performance tests.

IBM Rational Performance Tester (RPT) is a performance and load-testing tool that can store test scripts for automated execution and create a rich set of reports to summarize performance characteristics of the application. It sends HTTP requests to your web application, simulating the large numbers of users, and tracks page response times under different test cases. RPT enables you to record steps in a browser and then play them back to run the test with multiple users. Data returned from pages can be simulated using data pools, which can be created to match test cases. It also helps identify performance bottlenecks by pinpointing lines of code with slow execution. RPT has built-in support for enterprise systems, including SAP and Seibel. It can help correlate system health with test execution so that you can see how your system is stressed under load. RPT can be combined with ITCAM, discussed in the section "Application Monitoring" in this chapter.

Cloud computing is an ideal environment for performance testing because a large number of resources can be mobilized over a short time period to conduct resource-intensive tests. Another approach is to measure the system with a certain compute capacity and then repeat the test with double (or triple, and so on) the compute capacity. Based on that data, you can extrapolate to see what compute capacity is required to support the predicted system load before making the service available to customers. That assumes that you see a linear trend between compute capacity and the load capable of supporting it, of course. If you do not see that, you need to identify the bottleneck. Some bottlenecks are hard to identify until the full production load is applied. For example, suppose that a 4 CPU system can support 50,000 users, an 8 CPU system can support 100,000 users, but a 16 CPU system can support only 110,000 users. You have identified a problem in scaling your application past 8 CPUs.

Availability

A number of different approaches to availability exist. Two fundamentally different approaches are traditional high availability and design for failure. In the traditional model, the infrastructure takes responsibility for availability. For example, a J2EE application might rely on high availability features from the application server and database. In a design for failure approach, the application and management infrastructure is built to handle failure.

Availability also can be achieved at different levels, including physical, virtual resource, availability zone, region, and cloud. Physical redundancy means replicating all the physical infrastructure for the application to run in, including servers and network equipment. This is a primary technique for high availability in a traditional approach. However, a design for failure approach ignores physical redundancy because hardware is expected to fail.

Virtual resource redundancy enables you to avoid faults in hardware by spreading your resources over different availability zones. Because hardware in clouds is usually located in racks with many CPUs in each node of the rack, this approach can give you some guarantee if a whole node or a whole rack goes down. Spreading resources over different regions, or data centers, can enable you to avoid service disruption if an entire data center becomes unavailable.

Another issue in high availability scenarios where hardware fails is loss of data. Most applications store data in relational databases. To prepare for this, you should be incrementally backing up data. Different databases have different schemes for this. Then if one database fails and you need to switch to another, you will have lost a minimal amount of data. Another option for data storage is a NoSQL database. A NoSQL database trades data consistency for partition tolerance, which can be thought of as the capability to split data between partitions in different regions.

Designing for high availability can be difficult and expensive. Another approach for applications that are not mission critical is to be able re-establish your application quickly in the event of a failure. Regular data and system backup enables you to do this easily in a cloud environment, depending on the complexity of your system topology.

We often hear about availability in terms of a number of nines. Table 8.2 shows how difficult it can be to achieve the number of nines in some claims.

Table 8.2 Availability versus Downtime

Availability	Downtime per Year
90%	36.5 days
99%	3.7 days
99.9%	8.8 hours
99.99%	53 minutes
99.999%	5 minutes

Just managing maintenance and system upgrade windows can make achieving these goals difficult, let alone planning for power and network outage, equipment failure, or disasters. Human errors are far more common than equipment failures. Most important, you need to invest considerable effort in planning and practicing every configuration change to the system, to avoid downtime. This is one area where cloud computing has considerable advantages: It greatly reduces the scope of IT operations that your own organization is responsible for.

Backup, Recovery, and Restore

Backup can be done locally, or it can be done to a remote location. The advantage of backing up to a remote location is that, if the entire location where your application is hosted becomes unavailable, you will still have your data. The elastic nature of cloud storage is helpful in planning for backup of data because you do not need to plan for and procure a fixed set of hardware in advance. Important requirements for backup include these:

- Backup should be nondisruptive—that is, backup should not require you to take your system offline.
- Backup should provide for fast recovery.
- Backup should be schedule driven.
- Backup should be automated.
- Encryption of data is an additional desirable requirement in cloud environments.

Cold backup is the simplest and cheapest method of recovery. In this method, the application is stopped and a full image of the data is taken and saved to a location that cannot immediately make use of it if a failure occurs. This ensures consistency of the data. However, the application is offline during the process. For large sets of data that take a long time to restore and for mission-critical applications, this is unacceptable. Other methods, known as application-consistent backup, can make backups without needing to take the application offline.

Typically, backups involve the following components:

- **Backup server**—This is the administrative point for the backup system
- **Storage node**—This is where the backup data is stored.
- **Agent**—An agent is installed on the application server to manage collection of data and metadata.
- **Application server**—This is the system that owns the data being backed up.

A snapshot is a copy of the original data at one point in time. Snapshots allow for a full restore of the data. In most cases, a snapshot is a copy of an entire storage volume. The disadvantages of snapshots are that making them takes a considerable amount of time, they consume a large amount of storage space, and application performance can be impacted greatly while taking a snapshot.

Incremental backup is one form of application-consistent backup that saves only the increment since the last backup. It saves time and network bandwidth, but restoring data based on incremental backups can be more complex and time consuming. In a differential backup, all the changes since the last full backup are saved. This can make it simpler to restore than with incremental backups because only the full backup and the differential backup are needed, compared with the full backup and all the incremental backups with an incremental backup strategy. One way to save time in restoring from incremental backups is to reconstruct a synthetic full backup offline on a regular basis before a failure occurs.

With either full backup, incremental backup, or differential backup, some data will be lost: the data saved by the application after the last backup and before the failure. Figure 8.6 shows recovery with full or incremental backup.

The data modifications since the last backup will be lost. Continuous data protection is a technique that captures every change in application data; it can be file based, block based, or application based. Figure 8.7 shows this.

Continuous backup can also reduce the time necessary for restore.

Backups can be done at different levels:

- **File level**—If a file changes, the whole file must be backed up. Open files might not be able to be backed up.
- **Block level**—Only the block within a file that changed is backed up. This requires lower-level tools to work below the file level.

- **Client-side backups**—The data is backed up to the local system. This saves network bandwidth, but at additional risk. However, RAID can be used, and a client-side backup strategy can be combined with other methods to periodically save backup data on another system.

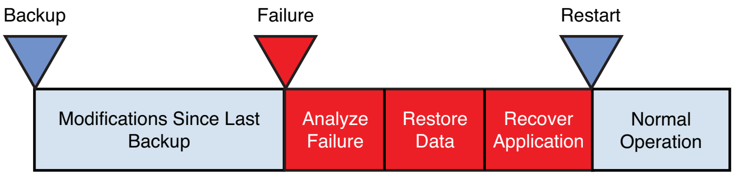


Figure 8.6 Application recovery with traditional backup

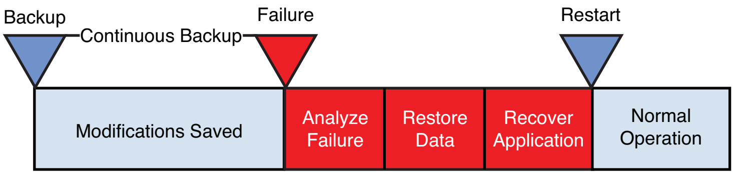


Figure 8.7 Application recovery with continuous backup

Tape is still a common backup media for traditional IT systems because of its low cost and high reliability. However, backing up to disk is becoming more prevalent because of the speed of restoring data and convenience in use over wide area networks. In cloud backup, data is backed up to a network storage device on the cloud. A virtual tape library (VTL) looks and acts like a tape but is actually a file-based system. It can be useful when working with legacy systems that depend on tape backups.

Backup data can consume a lot of space, depending on the size of the data set backed up and the frequency of backup. Data deduplication is a technique to save only unique data elements. It depends on a deduplication capability in the backup system.

It is a common mistake to do backup but not test data restoration. Testing restore and recovery is time consuming but important, to make sure that you can actually recover from a failure. After all, recovery and restoration of service will be an urgent matter when it is really needed. The process of recovery can involve these steps:

1. Diagnosing the problem
2. Deciding what recovery point to use
3. Deciding what point to recover to
4. Restoring data from the backup location to the recovery location

5. Performing standard recovery and startup
6. Testing

Diagnosing the failure is important and affects the action taken. Possible causes of failure include these:

- **Application failure—**The application software may fail due to a bug, lack of robust error handling, an overloaded condition, or some other reason. Usually recovery is possible with a restart.
- **Power failure—**Most applications will recover from a power failure most of the time. A simple restart might be all that is required.
- **Disk failure—**The mechanical nature of hard disks makes them the most likely hardware component to fail. Backup data from another system likely will be needed in this case.
- **Hardware failure other than disk failure—**In a physical IT environment, you might need to perform maintenance or replace the system. In a cloud, you will need to restart the system on another virtual machine instance.
- **Data Center failure.**

A good strategy to minimize downtime is to have an idea of the cause of the problem as soon as it occurs. Monitoring is important to achieve this.

Another strategy for reducing recovery time is to keep backup data locally and replicate offsite. The local data can be quickly accessed for recovery, and the offsite data can be used in the case of an emergency.

Storage Availability

Commercial hard disks fail at the rate of approximately 4% per year [Thorsten, 2008]. So whether you are managing an application in a traditional or a cloud environment, you need to consider the availability of your storage system. One of the advantages of IBM SmartCloud Enterprise is support for storage availability areas. An availability area provides some guarantee that, if data is stored in multiple availability areas, a hardware failure will not result in loss of the data. This is somewhat similar to some **Redundant Array of Independent Disks** (RAID) configurations in traditional IT.

IaaS clouds also enable you to format your own storage volumes, including with the use of RAID configurations. RAID is a storage technology that improves storage availability using redundancy. It distributes data across multiple physical disk drives as a logical unit. The way the data is distributed among the disks determines the RAID level, which implies the combination of performance and fault tolerance. The physical disks are termed a **RAID array** that the operating system accesses as a single logical disk. **Striping** is a technique for arranging blocks of files sequentially across multiple physical disks to allow the data to be read and written concurrently across those disks. Using striping, data can be read and written by the operating system at a greater rate than supported by the physical devices. **Parity** data can be added to disks so that, if any one disk fails, the lost data can be reconstructed. Disk **mirroring** involves cloning the data on the disk to a separate physical disk in real time. In this way, the mirrored disk is always identical to the original disk and, if a disk fails, no data is lost.

Table 8.3 summarizes the RAID levels.

Table 8.3 RAID Levels

Level	Description
RAID 0	Has striping but no parity or mirroring. Highest performance and lowest fault tolerance. If any one disk fails, data is lost.
RAID 1	Mirroring with no striping or parity. Highest fault tolerance and lowest performance. Only two disks are needed for this level.
RAID 2	Bit-level striping and Hamming-code parity. Minimum of three disks needed. Can recover from a failure of one disk.
RAID 3	Byte-level striping and parity. Minimum of three disks needed. Can recover from a failure of one disk.
RAID 4	Block-level striping and parity. Minimum of three disks needed. Can recover from a failure of one disk.
RAID 5	Block-level striping and distributed parity. Minimum of three disks needed. Can recover from a failure of one disk.
RAID 6	Block-level striping and double-distributed parity. Minimum of four disks needed. Can recover from a failure of two disks.

RAID 0 and 1 are the two extremes, and RAID 2 through 6 achieve intermediate levels by adding disks and using more complex systems.

Availability of Relational Databases

Basically, no difference exists between physical and cloud in terms of availability of a single machine. However, there are large differences in local storage between cloud-based virtual machines and servers that are typically used to host production databases. Cloud-based virtual machines generally have a standard amount and type of local storage. Because local storage is important for operating relational databases, production database servers generally have a greater amount of storage, and it tends to be higher performance than in other systems. Some clouds have no local storage. For these reasons, and for a standard high availability solution, it can be beneficial to look for a platform as a service solution for a relational database.

Relational databases can be backed up fully or incrementally. In general, the database can remain online when being backed up, but it will operate in backup mode. This is important to minimize downtime for 24×7 operations. In this mode, data is received by the database but is cached after the backup operation finishes. After the backup, all the changes in the cache are applied to the data files.

IBM DB2 has a tools catalog that is used for scheduling jobs, including backups. Also, by default, DB2 is set up to send notifications of events, such as health alerts, to administrators.

Virtual IP Addresses

A virtual IP address is an IP address that is assigned to multiple servers but is active on only one server at any one time. The IBM SmartCloud Enterprise infrastructure cloud has the capability to support this. The capability provides flexibility for high availability for applications deployed in the cloud.

In addition to a regular static IP address, a VM can dynamically assume one or several additional virtual IP addresses. Virtual IP addresses are not active when the VM is started. They are activated by entering this command in the VM:

```
# /sbin/ifup eth1
```

`eth1` is the Ethernet interface associated with the virtual IP. Be careful when you activate the virtual IP: If the same IP addresses may be associated with another server and it is up, you will end up with an IP address conflict. To deactivate a virtual IP, use this command:

```
# /sbin/ifdown eth1
```

An example of the application of these capabilities is high availability for a web site using a reverse proxy server. Two similar web servers are provided to end users as reverse proxy servers. One of them acts as the primary proxy server; the other is a warm standby server. Figure 8.8 shows the network configuration of these servers.

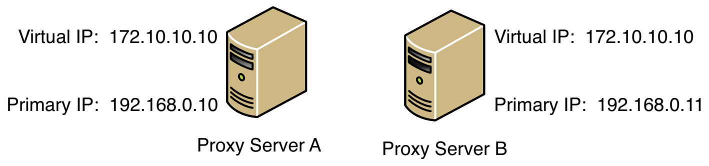


Figure 8.8 Network configuration for reverse proxy servers

192.168.0.x is a private LAN that is not visible to the user. 172.10.10.10 is the address users can access. After the virtual IP in proxy server A gets activated, it acts as the main proxy server; the virtual IP address on server B is not activated, as shown in Figure 8.9.

When server A is down or the response time of that server is slow, you can deactivate the virtual IP address of server A and activate the one in server B. The roles of these two servers then switch. Thus, you can achieve high availability of the service. The Linux kernel has the capability to operate in a high availability mode in this manner so that you do not have to build your own. The software package nginx also provides high availability using this feature. These modules operate a heartbeat to monitor for availability of the servers. Note that, with the IBM SmartCloud Enterprise, you need to operate with unicast; broadcast and multicast are not supported.

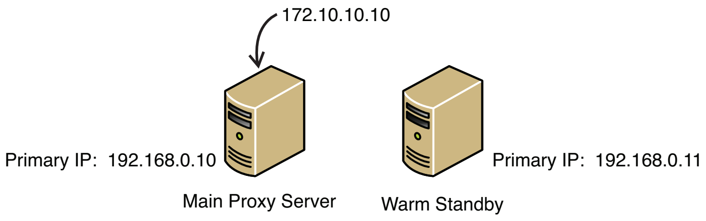


Figure 8.9 Virtual IP address assigned to the main proxy server

Monitoring and Metering

Monitoring in the cloud has several implications that are different from traditional computing:

- The dynamic nature of cloud computing means that monitoring is more important. You prefer not to pay for resources that you are not using.
- If you need to scale to cloud levels, you need to automate.
- Monitoring often forms the basic for metering—that is, measuring how your customers use resources so that you can charge them.

The last point is an interesting one and is fundamental to cloud computing: Cost should be based on use. If you are providing a service to customers over the Internet, how will you charge them for it? Several possibilities exist: per unit time of service, per unit of data, per transaction, per user, one-time charges, or based on premium services. In IaaS clouds, customers are usually charged per hour of virtual machine time, per gigabyte of storage per unit of time, or per IP address per unit time. In traditional computing, software costs can have complex license models. Often these are based on power units or seats—that is, the more compute power your servers have, the more the license costs are. For example, license costs might be different for number of CPUs and type of CPU (32 or 64 bit, Intel or RISC). A per-seat license is basically the number of users of your application. It could be the number of users with a unique login, number of users served, or number of floating licenses. Whatever metering model you choose for your service, the implications for your business could be profound.

Many tools are available for monitoring. We divide them into four categories:

- Operating system monitoring
- Network monitoring
- Application monitoring
- Comprehensive monitoring solutions

Many of the monitoring methods in this section can be the basis for metering. You can use operating system monitoring tools to measure power units of computing capacity per unit time, storage used, network bandwidth used, and so on. Often Software as a Service is likely to be metered at a higher level, such as per transaction or per seat. In those cases, metering will be more closely aligned to application monitoring. Some problems specific to cloud computing that must be solved are how to tie resource usage to specific customers and how to keep a permanent record of metering data. Because of the dynamic and distributed nature of cloud resources, you should keep metering data in a central location, such as a database. Keeping track of log files will be too difficult and prone to error. We discuss an example of a metering strategy in the section "Business Scenario: IoT Data Performance, Availability, Monitoring, and Metering Plan," later in this chapter. You can also divide these areas into performance monitoring and availability monitoring. Operating system, network, and application monitoring are each needed and are discussed in the following sections. However, they do not form a complete monitoring strategy for these reasons:

- You cannot watch a console 24 hours a day.
- You still might not be able to tell if something is wrong just by watching the output.
- You might have too many systems to monitor using these tools.

The next section looks at tools for addressing these problems.

Operating System Monitoring

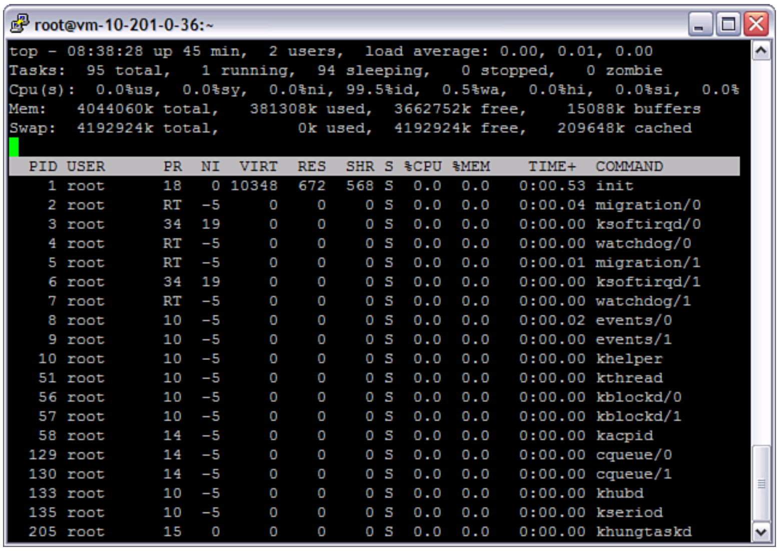
Tools for operating system monitoring fall into distinct categories, including fundamental tools, availability monitoring, and performance monitoring. Fundamental tools, such as syslog, top, vmstat, mpstat, and the `/proc` filesystem, are useful by themselves and also form the basis for other tools. Nagios is a popular availability-monitoring tool. Munin is a popular performance-monitoring tool. IBM Tivoli Monitoring also has extensive capabilities for operating system monitoring, but we defer discussion of that to the upcoming section "Comprehensive Monitoring Solutions."

Operating system monitoring is fundamental to all monitoring. One of the most basic tools on Linux is `syslog`, which is a library that many services use to log to files in the `/var/log` directory tree. Log entries contain a description of the application that generated the message, a severity level, a time stamp, and the message. Log configuration can be done by editing the file `/etc/syslog.conf`. A common default setting is for all messages of syslog level *info* or higher to be logged to the file `/var/log/messages`.

You can write your own messages to the `syslog` files using the `syslog` library in C programs or by using the `logger` command in shell scripts. For example:

```
$ /bin/logger A bad thing happened
$ tail messages
...
Apr  4 10:13:59 vm-10-200-7-154 idcuser: A bad thing happened
```

The `top` command displays Linux tasks and a system summary. It is invaluable, providing a real-time view at an operating system level. The types and order of information displayed are all user configurable. Figure 8.10 shows sample output from the `top` command on an IBM SmartCloud Enterprise RHEL system.



```
top - 08:38:28 up 45 min,  2 users,  load average: 0.00, 0.01, 0.00
Tasks:  95 total,   1 running,  94 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.5%id,  0.5%wa,  0.0%hi,  0.0%si,  0.0%
Mem:   4044060k total,   381308k used,  3662752k free,    15088k buffers
Swap:  4192924k total,        0k used,  4192924k free,   209648k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      18   0 10348  672  568 S  0.0  0.0   0:00.53 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.04 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.01 migration/1
    6 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/1
    7 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
    8 root      10  -5     0    0    0 S  0.0  0.0   0:00.02 events/0
    9 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 events/1
   10 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   51 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
   56 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   57 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/1
   58 root      14  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
  129 root      14  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue/0
  130 root      14  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue/1
  133 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
  135 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
  205 root      15   0     0    0    0 S  0.0  0.0   0:00.00 khungtaskd
```

Figure 8.10 Linux top screen shot from a SmartCloud Enterprise RHEL system

The summary output shown indicates that the CPUs are 99.5% idle; there is 4 GB of memory, of which 381 MB is used; and 1 process is running and 94 processes are sleeping. Table 8.4 describes the fields shown in Figure 8.10, from left to right.

Table 8.4 Description of top Task Detail Output

Abbreviation	Name	Description
PID	Process ID	The unique process.
PPID	Parent process ID	The unique parent process.
RUSER	Real user name	The real owner of the process.
UID	Effective user ID	The user ID of the effective owner of the process.
USER	User name	The user name of the process owner.
GROUP	Effective group name	The effective group of the owner of the process.
TTY	Controlling terminal	The controlling terminal of the process.
PR	Priority	The priority of the task.
NI	Nice value	A negative value is a higher priority, a positive value is a lower priority.
P	Last used CPU	CPU affinity. In a true SMP environment, this changes frequently.

Table 8.4 Description of top Task Detail Output (continued)

Abbreviation	Name	Description
%CPU	CPU usage	Divided by total number of CPUs.
Time	CPU time	Total CPU time since the process started.
%MEM	Memory usage	Share of available physical memory.
VIRT	Virtual memory	The amount of virtual memory used by the task.
SWAP	Swapped size	The amount of virtual memory swapped.
RES	Resident size	The nonswapped physical memory.
CODE	Code size	The physical memory dedicated to executable code.
DATA	Data size	The physical memory dedicated to other-than-executable code.
SHR	Shared memory size	The amount of shared memory used by a task.
nFLT	Page fault count	Attempts to read or write to a page that is not currently in its address space.
nDRT	Dirty pages count	Number of pages modified since the last write to disk.
S	Process status	One of the states uninterruptible sleep, running, interruptible sleep, stopped, paging, dead, or defunct.
Command	Command	The command that started the process.

When you enter top from the command line, the cursor is positioned in an area that enables you to enter interactive commands. Type `h` for help and `q` to quit. You can start top in batch mode with the `-b` flag, to keep a record of system performance or to send to another program. You can monitor the results by user with the `-U` flag and by process with the `-p` flag.

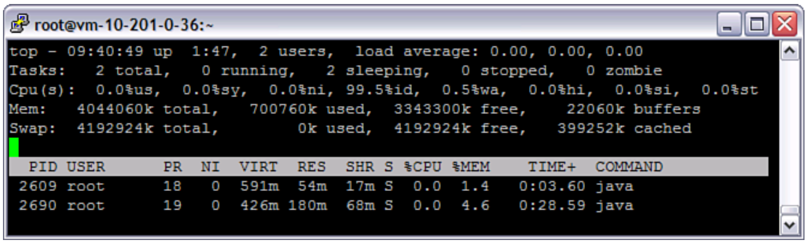
As an example, suppose that you had a problem with a Java process that is not shown on the top output. However, the top output shows that you have 94 sleeping processes and 1 running process. These are not all displayed on the output by default; you need to find the process ID of the Java process and use the `-p` flag. You can do this with these commands:

```
# ps -ef | grep java
root      2609  2595  0 08:30 pts/0    00:00:03
/opt/IBM/WebSphere/AppServer/java/bin/java
root      2690  2609  0 08:40 pts/0    00:00:28
/opt/IBM/WebSphere/AppServer/java/bin/java
```

Now that you know the process IDs, you can start up top so that only those processes are shown:

```
# top -p2609 -p2690
```

This results in the top output shown in Figure 8.11.



```
top - 09:40:49 up  1:47,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:   2 total,   0 running,   2 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.5%id,  0.5%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4044060k total,   700760k used,  3343300k free,    22060k buffers
Swap:  4192924k total,        0k used,  4192924k free,   399252k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2609 root      18   0  591m  54m  17m S  0.0  1.4   0:03.60 java
 2690 root      19   0  426m 180m  68m S  0.0  4.6   0:28.59 java
```

Figure 8.11 top output focusing on Java processes

Although the Java processes are slow, the CPU and memory utilization is low. However, the RES column shows that there is very little of the processes in resident memory: 54 MB out of 591 MB and 68 MB out of 436 MB. Maybe the problem is swap configuration. On a more healthy system, the resident memory for the same Java process is greater than 90%.

Virtual memory statistics can be reported with the command `vmstat`, including process memory use, paging, block I/O, traps, CPU, and disk statistics for the overall system. Table 8.5 lists the individual fields.

Table 8.5 Metrics Reported by vmstat

Abbreviation	Name	Description
r	Processes waiting for runtime	Number of processes waiting for runtime
b	Processes sleeping	Number of processes waiting in uninterruptible sleep
swpd	Virtual memory used	Amount of virtual memory used
free	Idle memory	Quantity of idle memory
buff	Memory used as buffers	Quantity of memory used as buffers
cache	Memory use as cache	Quantity of memory used as a cashe
inact	Inactive memory	Quantity of inactive memory
active	Active memory	Quantity of active memory
si	Swap in	Amount of memory swapped in from disk

Table 8.5 Metrics Reported by vmstat (continued)

Abbreviation	Name	Description
so	Swap out	Amount of memory swapped out to disk
bi	Blocks in	Blocks read in from a block device
bo	Blocks out	Blocks written out to a block device
in	Interrupts	Number of interrupts per second
cs	Context switches	Number of context switches per second
us	User time	Time spent in user mode
sy	Kernel time	Time spent in kernel mode
id	Idle time	Time spent idle
wa	Waiting for I/O	Time spent waiting for IO
st	Stolen time	Time stolen from a virtual machine
total	Total disk reads or writes	Total number of successful reads or writes
merged	Grouped reads or writes	Resulting in a single I/O operation
sectors	Sectors read or written	Number of sectors read or written to successfully
ms	Milliseconds spent	Reading or writing
cur	I/O in progress	Number of I/O operations currently in progress
s	Seconds for I/O	Seconds spent in I/O

The `mpstat` command reports processor-related statistics and reports activities of each available processor and global averages. Input parameters to the command include the time interval to report for.

In Linux, the `/proc` file system stores a large amount of constantly updated information about the current state of the system.

IBM Tivoli Monitoring (ITM) provides detailed operating system monitoring. It is discussed in the upcoming section "Comprehensive Monitoring Solutions."

Nagios is an open source monitoring platform that enables you to monitor systems remotely for availability, CPU load, memory, disk use, users logged in, and processes. It includes a core, plug-ins, and additional front ends. To install it, download the tarballs from the Nagios web site and follow the installation instructions. The installation includes setting up a user to run Nagios and running `Make` scripts.

Munin is a performance-monitoring tool. It provides detailed graphs of system performance over a web user interface. To install Munin, download the tarball and run the `Make` scripts.

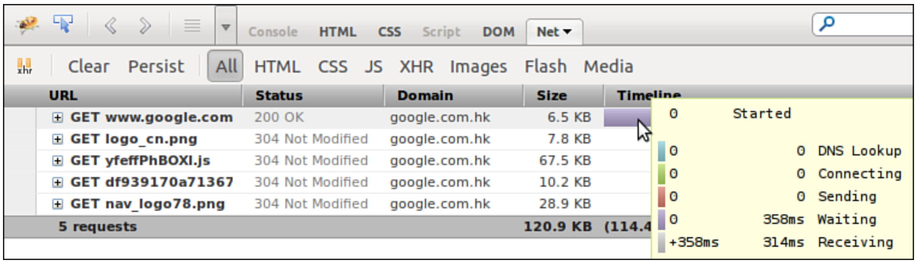
Network Monitoring

The Linux `netstat` command prints network configuration, connection activity, and usage statistics. With no arguments, `netstat` gives a list of active connections. Example partial output is shown here:

```
# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address        Foreign Address   State
tcp        0      0 localhost:ldap       localhost:50640   ESTABLISHED
tcp        0      0 vhost0308:corbaloc   www-900:24407     ESTABLISHED
tcp        0      0 vhost0308:9403       www-900:61028     ESTABLISHED
tcp        0      0 localhost:50640      localhost:ldap    ESTABLISHED
tcp        1      0 localhost6:54535     localhost6:46931 CLOSE_WAIT
...
```

We have deleted the last parts of the fully qualified domain names to make the output more readable. `Recv-Q is` the number of bytes not copied by the program using the socket. `Send-Q is` the number of bytes not acknowledged by the receiving host. The state field can be useful to identify connections in a bad state. Although not shown, the program using the socket can also be listed.

So far, we have discussed tools useful for monitoring server performance. Viewing the network performance as experienced by your users of your web site is essential. The Firefox Firebug plug-in can display page-loading times, as shown in Figure 8.12.



URL	Status	Domain	Size	Timeline
GET www.google.com	200 OK	google.com.hk	6.5 KB	
GET logo_cn.png	304 Not Modified	google.com.hk	7.8 KB	
GET yfeffPhBOXI.js	304 Not Modified	google.com.hk	67.5 KB	
GET df939170a71367	304 Not Modified	google.com.hk	10.2 KB	
GET nav_logo78.png	304 Not Modified	google.com.hk	28.9 KB	
5 requests			120.9 KB	(114.4

```
0              Started
0           0  DNS Lookup
0           0  Connecting
0           0  Sending
0       358ms  Waiting
+358ms  314ms  Receiving
```

Figure 8.12 Page element loading times in Firebug

Application Monitoring

Application monitoring and application performance analysis are closely related areas. They overlap but differ in several respects:

- Monitoring is primarily the responsibility of an operations team; performance analysis is primarily the responsibility of a development team.

- Monitoring needs to identify points when problems occur; analysis needs to identify bottlenecks in the application structure.
- Performance analysis includes profiling, which requires deep insight into the structure of an application.

However, monitoring and analysis can overlap for these reasons:

- The results of monitoring need to be actionable. Developers with limited control over a production system might not have the capability to reproduce problems after they occur or to do a deep analysis on a production system. The very tools that enable profiling can have a large impact on the performance of the system.
- Analysis tools can be capable of capturing output over time that can be used as input into monitoring applications.

Logging is a fundamental tool for application monitoring and maintenance. In the Java world, the two most popular logging frameworks are the Java Logging API and Apache Log4j. In fact, Log4j can be a Java Logging provider. An example of the use of the Java Logging API is shown here for a program that connects to the IBM SmartCloud asset repository:

```
package com.ibm.cloud.example.ram;

import java.util.logging.Level;
import java.util.logging.Logger;

import com.ibm.ram.client.RAMSession;

public class RAMClient {

      private static final String USER = "a@b.com";
      private static final String PASSWORD = "****";
      private static final String URL = "https://www-
147.ibm.com/cloud/enterprise/ram.ws";
      private RAMSession session;
      private static Logger logger =
Logger.getLogger("com.ibm.cloud.example.ram");

      public void connect() {
            logger.fine("Creating a connection");
            try {
                  session = new RAMSession(URL, USER, PASSWORD);
            } catch(Exception e) {
                  logger.log(Level.SEVERE, "Error connecting", e);
            }
            logger.log(Level.INFO, "session = " + session);
      }
```

```
        public static void main(String[] args) {
              RAMClient client = new RAMClient();
              client.connect();
        }
  }
```

The program creates a `logger` object as a private field with the name `com.ibm.cloud.example.ram`. In the `connect` method, you log three statements. The first is a debug statement that tells you that you are creating a connection. The second statement is a severe error message that you encountered an exception when attempting to connect to the asset catalog. The third statement is an informational statement that prints the value of the session field. Initially, you can use these statements for debugging. Later, during the development and test phases of the project, you retain only the informational messages. You want to capture these log statements in a file so that your test and operations teams can send them to you for analysis. To do that, you use the configuration file `myLogging.properties`, shown here:

```
handlers=java.util.logging.FileHandler
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormat
ter
java.util.logging.FileHandler.count=10
java.util.logging.FileHandler.pattern=myApplication%g.log
com.ibm.cloud.example.ram.level=FINEST
```

The `handlers` property is set to `FileHandler` so that the log output will be written to a log file. A `SimpleFormatter` is used to give human-readable output. The `java.util.logging.FileHandler.count` property keeps ten rolling log files. The pattern property specifies the file names to be used. During the debug phase, you presumably set the log level to `FINEST`. Later, in operations, you will set it to `SEVERE` log–only errors. The possible log levels are `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`. To invoke the program with this log configuration file, use the Java system property, invoking the program as shown here:

```
java com.ibm.cloud.example.ram.RAMClient -
Djava.util.logging.config.file=bin/myLogging.properties
```

You add the configuration file under the `bin` directory. Example output to the log file is shown next:

```
2011-7-21 17:20:59 com.ibm.cloud.example.ram.RAMClient connect
FINE: Creating a connection
2011-7-21 17:21:00 com.ibm.cloud.example.ram.RAMClient connect
INFO: session = com.ibm.ram.client.RAMSession@69846984
```

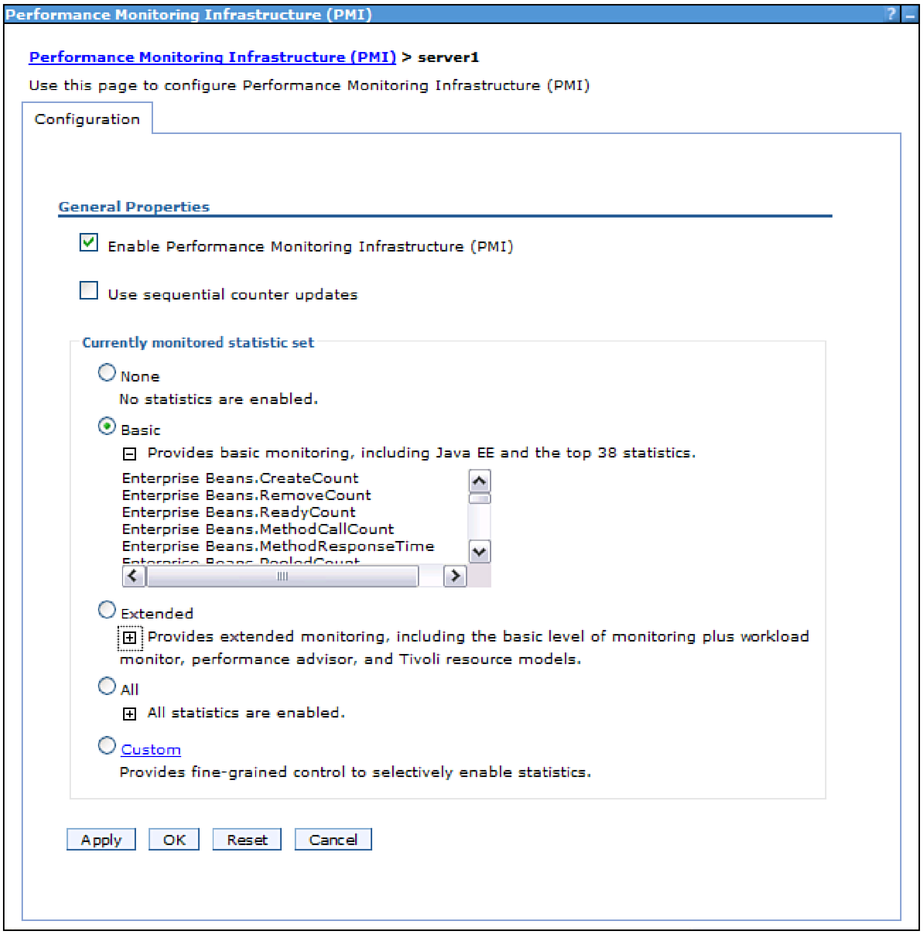
The log shows such useful fields as the time stamp, class name, log level, and message. You can configure these fields as well, if needed.

To make use of logs in a monitoring context, you can write log-monitoring agents. These agents parse logs in real time (that is, they tail the logs) and look for clues of problems, such as

Exception messages. For example, in tailing the previous log, you could look for lines starting with the string `SEVERE`. If we encounter such an error, you could create a support ticket and send an email.

The Java Standard Edition platform includes monitoring and management capabilities via the `java.lang.management` package APIs and the Java Management eXtension (JMX). Several open source and commercial tools build on these platforms.

WebSphere Application Server has built-in monitoring capabilities. These are available under the Monitoring and Tuning, Performance Monitoring Infrastructure menu on the administrative menu. To begin doing this, enable the Performance Monitoring Infrastructure, shown in Figure 8.13.


Performance Monitoring Infrastructure (PMI)

Performance Monitoring Infrastructure (PMI) > server1

Use this page to configure Performance Monitoring Infrastructure (PMI)

Configuration

General Properties

[✓] Enable Performance Monitoring Infrastructure (PMI)

[ ] Use sequential counter updates

Currently monitored statistic set

○ None
No statistics are enabled.

● Basic
⊟ Provides basic monitoring, including Java EE and the top 38 statistics.

```
Enterprise Beans.CreateCount
Enterprise Beans.RemoveCount
Enterprise Beans.ReadyCount
Enterprise Beans.MethodCallCount
Enterprise Beans.MethodResponseTime
```

○ Extended
⊞ Provides extended monitoring, including the basic level of monitoring plus workload monitor, performance advisor, and Tivoli resource models.

○ All
⊞ All statistics are enabled.

○ Custom
Provides fine-grained control to selectively enable statistics.

Figure 8.13 Enabling Performance Monitoring Infrastructure in the WAS administrative console

When Performance Monitoring Infrastructure is enabled, you can collect metrics and view performance summary statistics and graphs.

IBM Tivoli Composite Application Manager (ITCAM) is a component of ITM for monitoring composite applications, including J2EE, SAP, and other enterprise platforms. It is an end-to-end transaction management monitoring solution that simulates end user behavior. The Response Time Tracking component monitors the response time from web applications. ITCAM also integrates with Rational Performance Tester to record and playback scripts.

Comprehensive Monitoring Solutions

By "comprehensive monitoring solution," we mean a solution that can monitor different aspects of performance and availability, set up alerts and notifications, cover many systems and many different types of systems, and use a central dashboard to view the overall status. ITM Tivoli Monitoring (ITM) is a monitoring solution that includes these capabilities and is available in the IBM SmartCloud Enterprise image catalog. It consists of several components:

- The central portal Tivoli Enterprise Portal (TEP), with dashboard, endpoint configuration, summary views, and alert configuration
- Endpoint connectors, both agent based and agentless
- Tivoli Data Warehouse, for collection of data and reporting
- Agent Builder

Figure 8.14 shows the Tivoli Enterprise Portal.

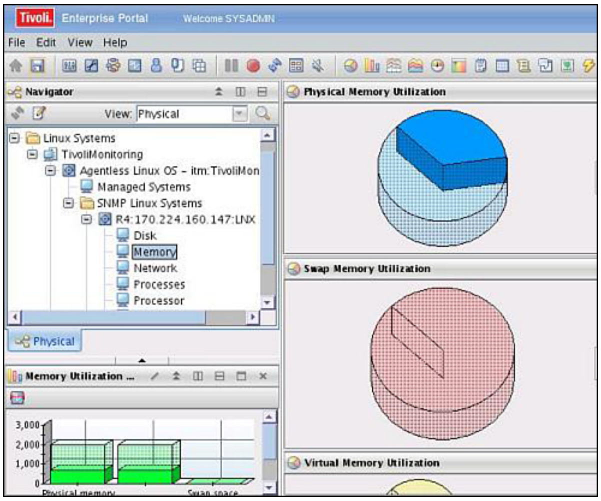


Figure 8.14 Tivoli Enterprise Portal

Figure 8.14 shows an agentless Linux endpoint. TEP connects to the Linux system using SNMP to retrieve monitoring data.

The difference between an agent-based and an agentless adapter is that some resources support remote monitoring APIs, which allows the use of agentless endpoint interfaces. In general, an agentless adapter is more convenient to manage because no software must be installed on the remote systems. However, this is not always possible. Agent-based adapters can be installed on the monitored systems, allowing the collection of more detailed information that is perhaps not available remotely. In addition, agent-based adapters can operate either with the central monitoring server or autonomously.

Custom agents, such as those that listen to logs, can be built using a wizard with the ITM Agent Builder.

IBM Systems Director™ is an availability-monitoring and management product. Systems Director can do the following:

- Automate startup of single system and composite systems
- Simplify and identify a single point for the management of many servers, both physical and virtual
- Help understand and manage the health of related systems

Systems Director also integrates with ITM.

Business Scenario: IoT Data Performance, Availability, Monitoring, and Metering Plan

IoT Data is a storage-centric application. Its main concern in performance and scalability is being able to scale to large amounts of devices, files, and total data. The plan to test this is to run a series of scale and performance tests for a period of two to three weeks before going live. IoT Data will start up a large number of virtual machines and storage volumes. It will use the SmartCloud Enterprise REST APIs to create the storage volumes and virtual machines to support this. In addition, custom scripts will be created to generate a large number of files, to stress the storage-management system.

The capability to handle a large user base with acceptable response times is also a concern. The performance test team will test with user populations of 100, 200, 400, 800, 1,600, and so on to ensure that the performance scales linearly and to predict the capital expenses to support the IoT business model.

Table 8.6 IoT Data System Inventory

System	No. of CPU Cores	Data Center	Software
ihs1	2	1	IBM HTTP Server
ihs1	2	1	IBM HTTP Server
was1	4	1	WebSphere Application Server
was1	4	1	WebSphere Application Server
db21	8	1	DB2, Tivoli Directory Server
db22	4	2	DB2, Tivoli Directory Server
hdfs1, hdfs2, etc	4	1, 2	Hadoop, storage volumes attached
tsom	4	1	Tivoli Security Operations Manager
itm	4	1	IBM Tivoli Monitoring

Two data centers are used, Data Center 1 and Data Center 2. For performance, some systems are clustered; for high availability, critical systems are replicated at different data centers. For the noncritical systems, virtual machine images at Data Center 2 are ready for a cold start. The IHS servers are clustered using DNS spraying. The WebSphere servers are clustered, and load balancing is done with the proxy. High availability is ensured for data using the DB2 HADR replication system, which replicates data every two hours. The actual customer data is stored in Hadoop cluster with the Hadoop Distributed File System (HDFS). TSOM is used to check security events in the system logs, web servers, directory server, and database. ITM is used to monitor performance and availability. If a system becomes unavailable, an alert will be sent to a system administrator. Application-level monitoring is done by a custom agent that monitors the application logs on the WebSphere servers.

IoT Data provides a data-centric service. It is natural to charge customers by the amount of data used—say, based on gigabytes per month. We can measure the amount of data used with an operating system–level monitoring agent. We need one that can log usage data to a central database. We will keep track of the customer associated with each file by keeping a record of the customer-to-file name mapping when customers upload files. Every month, before we send customers a bill, we will run a job that transforms the data into metering data that our billing system can use. We will write this as a stand-alone Java program that is invoked by calendar date and time using the Java scheduling service.

This page intentionally left blank

Chapter 9

Operations and Maintenance on the Cloud

We have discussed the various aspects of putting together a cloud application and now can discuss how to manage an application as part of a business. Not only do you want to run on the cloud for small systems, but you also want to be able to scale up to large systems with a minimum of system administration. That means you need to think about automating everything you do. You can automate some activities through image customization. For example, you can set up the right firewall rules before the image is saved so that you do not have to repeat it with every server you add. However, some activities, such as requesting and installing certificates for HTTPS, have to be done after image customization. This chapter also addresses maintenance of cloud applications. You might have offloaded a lot of system administration to the cloud provider, but now you have become more dependent on the cloud. One of the characteristics of current clouds is that they are rapidly evolving. What if they change something that affects you? How can you future-proof yourself and your enterprise against changes?

Business Support Systems

The business support systems (BSS) include customer management, subscriber management, metering, billing, cost management, and other back-office business processes. To scale your business, you need to automate as much of this as possible. You saw some of the use cases for BSS in the section describing the Distributed Management Task Force Open Cloud Standards Incubator work in Chapter 1, "Infrastructure as a Service Cloud Concepts." Here is a consolidated list:

- Opening a customer account. The consumer of the service opens an account with the service provider, including agreeing to terms and conditions.
- Closing a customer account. The consumer closes the customer account with the service provider.

- Terminating a customer account. The service provider terminates the customer account.
- Updating a service contract. The customer adds new offerings to the service agreement, upgrades, or agrees to modified service.
- Adding a new offering. The service provider adds a new offering to the service catalog. This includes setting the price for the offering and the unit of measure for the offering. For example, will the offering be charged per hour, per month, per user, per transaction, or with a one-time charge?
- Retiring an offering. The service provider removes a service. It may be that customers already using the service can continue to use it for a period of time, but no new customers can register for the service.
- Creating a user account. The service provider or customer administrator adds a user account.
- Modifying a user account. The service provider or customer administrator adds a user account.
- Deleting a user account. The service provider or customer administrator adds a user account.
- Billing a customer. The service provider bills the customer. Either a credit card is billed or an invoice is sent. Usually, this is done on a regular basis for a fixed billing period, such as at the end of each month. A whole set of activities enable this.
- Customer viewing a billing statement. Customers should be able to see what they are paying for.
- Customer viewing a usage report. Customers should be able to check the usage for the past billing period.
- Modifying a customer charge. A charge may be deducted because of an inability to provide the service, or a special service might be provided to a customer.

You might consider other use cases for marketing or other business reasons, such as free trials, performance of a premium service, and so on.

Developing software for all of these use cases would be a lot of work. Depending on the sophistication of your service, you might or might not need all these use cases. When managing customer accounts, you might need your own customer database, and you might also need to interact with other systems, such as customer relationship management systems. One option is to manage customers and other business services with online systems such as Salesforce.com. Salesforce has application programming interfaces that can automate manual work and a user interface to monitor customer management and other business activities.

You can consider several options for billing models:

- Develop an image and charge via the cloud provider. In this model, you use the cloud provider's BSS, which then sends you payments. You need to be able to register your customers with the cloud provider and adopt the cloud provider's metering and billing model. For the IBM SmartCloud Enterprise, this might work well if your service can be packaged into an image. In this case, you use IBM's BSS and have no need to write your own. If you can fit your service into an image, this solution has the lowest up-front development cost.

- Use a monthly subscription. This is perhaps the simplest billing model, but your challenge is to provide the different BSS functions to support it yourself. Metering is not necessary because it does not depend on actual usage. However, it might not be appropriate in scenarios where customer usage determines your own cost of operations.
- Base billing on user seats. This model is only slightly more complex than a monthly subscription and accounts for differences between large and small customers, based on the number of users allowed to use the service. It is appropriate for services in which your cost is largely determined by the number of users supported.
- Base billing on usage. In this model, you need to track customer usage and bill each customer accordingly. This is often the most reasonable from a customer's point of view. However, a considerable amount of development work is needed to support this.

You can do billing with payment gateways that financial institutions provide, such as PayPal. Again, PayPal has application programming interfaces to eliminate manual work.

When integrating with external systems, you can build out cloud applications to call out to them, or you can write an interface and have those systems call into you. The latter is somewhat more convenient for evolving over time. You can build a standard interface beginning with minimal information for the initial beta program and free trials, and build up to integration with one or more commercial systems to process transactions from paying customers. You can use simple systems such as Salesforce and provide a standard interface for integration with more complex systems. This is preferable to baking all the business logic into your infrastructure application yourself. As time goes on, you might not find it convenient to drag out your application code and add yet another piece of business logic.

Maintaining Compatibility with Future Versions of Software

After releasing and supporting a cloud service in production for some time, you will find that the cloud and some of the components that you depend on will have changed. You are somewhat more at risk in this respect, compared with traditional IT deployments, because you have given some control of your environment to the cloud. The goal of this section is to help you write robust client code that can stand up against changes in the cloud environment and APIs.

An Evolving API

An important goal in providing an API is that it be stable over a long period. Incremental changes in minor releases must be compatible. In some cases, major releases of an API are added that result in a major revision of the API. At that point, two API versions should be supported.

A key question for cloud providers is, how many versions of the API should be maintained and how should versions be designated? Cloud computing is evolving rapidly, driving frequent changes and additions to APIs. One strategy is to update the base URL for the API version. However, with the API being updated every two to three months, changing the base URL would result in too many versions being concurrently supported and would cause user confusion. The strategy adopted by IBM SmartCloud is to keep the base URL fixed and make minor updates to the API strictly compatible. Changing the base URL would also likely disrupt many API users.

Another risk is that every time an API is changed, something might accidentally break either because of a bug or because a client wrote code in a very brittle way, such as not handling white space properly. To mitigate against this possibility, client code should be written defensively and without dependencies on response formatting, such as white space and labels. In addition, cloud-maintenance notices should be monitored and the availability of the cloud application checked after maintenance windows.

Java

Java does not have explicit versioning built into the platform. This has led to initiatives such as Open Service Gateway (OSGi), which models a versioning and component system with dependencies. A simple way to indicate dependencies can be maintained in Java libraries using the JAR manifest file. An example from the IBM SmartCloud is shown here:

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven
Built-By: root
Build-Jdk: 1.6.0
API-Version: 1.4
Build-ID: CC20110325-0523
```

The manifest file indicates that the version is 1.4.

Compatibility of changes to a Java APIs is relatively easy to define due to the type-checked nature and solid design of the language. The prime directive should be this: Do not break clients written in conformance to APIs. Moving down a level, several types of compatibility are important:

- Contract compatibility ensures that method signatures are compatible.
- Binary compatibility means that the methods will be loaded without linkage errors.
- Semantic compatibility means that the meaning of the API does not change.

- Runtime compatibility means that the API will continue to run with an old client. Even though the previous three types of compatibility might be achieved, the API might fail with an old client if it was not implemented in a robust way.

Several implications and rules flow directly from the need to be binary compatible:

- New objects and methods may be added.
- No parameters may be added or removed from methods. If you find that you need to do this, add a new method instead.
- Checked exceptions may not be added to or deleted from a method signature.
- Unchecked exceptions may be added to or deleted from a method signature, but they should be semantically compatible.
- Return types must be the same.
- New fields, methods, or constructors added to an existing class.
  - New fields, methods, or constructors may be added to an existing interface, provided that the interface is not expected to be implemented by the client.
  - The last parameter `T` of a method signature may be changed to `variable arity T`.... However, this should be avoided because the meaning might be unclear.
  - Semantics of APIs (as well as syntax) must remain the same.

Some rules also apply to semantics:

- Don't strengthen preconditions (for example, forbidding `null` for input parameters).
- Don't weaken post conditions (for example, allowing `null` as a return parameter).

Some rules can be used by API implementers to help consumers of those APIs, including using the `@since` Javadoc tag to indicate the version in which methods and classes were added (for example, `@since` 1.2).

REST

At the time of writing, no standards exist for REST API versioning. However, debate on this subject abounds in Internet forums and blogs. Consider several rules for compatibility:

- The order of parameters does not matter in HTTP requests.
- The order of elements matters in JavaScript arrays (confusing because, in JavaScript, an array is a hashtable).
- Parameters can be added to future versions of REST methods and returned JSON arrays.
- Parameters will not be removed from future versions of REST methods and return JSON arrays.
- New methods may be added with new URLs.

- The base URL will not change.
- Parameters may be added to methods provided they are optional.

Guidance to consumers is that they should not rely on white space or string indexes in parsing results of REST calls.

Several suggestions for REST APIs have arisen to indicate versioning of the API and even to allow content negotiation. One of these is media type. Generally two lines of thought arise regarding use of media types in REST APIs:

- Standard media types should be used.
- Custom media types should be used to create a strict API contract.

Roy Fielding's thesis proposes that standard media types be used. A custom media type would be something like `application/i.did.it.myway+xml`. One argument for standard types is that they aid in the use of tools. An opposing argument for custom types is that they avoid the need for out-of-band knowledge about the content of the return data. The risk exists that certain firewall rules might not like customer media types. Many enterprise firewalls are set up to strip out Java applets and certain other media types.

XML

XML is a strongly typed language if an XML Schema is used, and many of the same rules that apply to Java also apply to XML. Consider some rules for backward compatibility for XML documents returned from REST APIs by a service provider:

- New elements may be introduced.
- New attributes may be added.
- The order of elements within other elements matters.
- The order of attributes does not matter.
- No elements or attributes should be deleted.

Some of these rules are different for the body of a REST request made to a cloud service:

- Optional new elements may be introduced.
- Optional new attributes may be added.
- The order of elements within other elements matters.
- The order of attributes does not matter.

JSON

Many REST APIs, such as the IBM SmartCloud Enterprise API, allow either XML or JSON to be used. XML and JSON have different levels of type checking, with JSON having no type checking. However, a similar set of rules to maintain backward compatibility is suggested.

Command Line

No standard relates to compatibility of command-line clients. However, some commonsense requirements apply:

- Order of parameters does not matter.
- Parameters should not be deleted in future versions.
- New parameters may be added in new versions.
- Parameters may be added to methods, provided they are optional.

Data

Data in cloud services can be thought of as part of a compatibility contract. Consider a lightweight application that launches a virtual machine instance with a particular compute size. The compute size typically has an identifier specific to its combination of CPU count, memory, and disk size. Now, if the cloud provider changed that identifier, the program would break. In this case, we conclude that the compute size identifier is part of the API contract. However, now consider a virtual machine image for a beta software product. Several months after the beta finishes, the image provider wants to withdraw the image from the catalog. In this case, we conclude that the image ID should not be part of the API contract.

We expect some data items to be part of an API contract:

- Compute size identifiers
- Identifiers for IP address offerings, storage volume offerings, and related resource offerings
- Data center identifiers

On the other hand, we expect other data items to not be part of an API contract:

- Image identifiers
- Text for labels, names, and descriptions

Business Scenario: IoT Data Operations and Maintenance Plan

IoT Data will use a third-party SaaS service for billing. It will collect metering data from monitoring agents, as discussed in the last chapter, and combine that with price data using SQL statements to generate billing data, which is then exported to the third-party billing service. The billing service will send an invoice the customers, settle the payments, and send the revenue to IoT Data's bank account. The goal in developing this solution is to eliminate manual work and minimize custom code.

The solution consists of shell scripts that execute commands over SSH and are driven by cron jobs. This sequence takes place:

1. On the last day of every month, the billing export cron job is triggered.
2. The job calls a script that logs on to the database system using SSH.
3. The script then executes a DB2 stored procedure to combine the metering and price data to populate the billing table.
4. The script exports the billing data from the database.
5. The exported file is copied with SCP to a secure location for safekeeping.
6. The third-party billing REST service is invoked with cURL commands, sending a POST request with the billing data.
7. The result is written to an audit trail.

Further Reading

IBM SmartCloud Enterprise

For IBM Cloud API references for the Smart Business IBM SmartCloud Enterprise, see the REST and command line reference manuals and the Javadoc available at www.ibm.com/cloud/enterprise. You need a user account to access these. The IBM DeverloperWorks article "Get Started with the IBM Smart Business Development and Test on the IBM Cloud" [Rekesh, et al., 2010] is an excellent guide in getting started guide and offers great tips on tools for working on the cloud. The developerWorks® article "Convert IBM Cloud Image Parameters into Java Using JAXB" [Vernier, 2011] contains a useful description of parsing the `parameters.xml` file. The documents "Creating and Customizing Images" [Goodman, 2009] and *External Image Provider's Guide* [IBM, 2011] provide descriptions of customizing images with integration with parameters. The article "IBM SmartCloud Enterprise Tip: Deploy a Complex Topology" [Vernier, 2011] describes deploying and managing systems with multiple virtual machines.

Cloud Background and Architecture

For more on cloud architecture and background behind the use of clouds, see "Cloud Computing Reference Architecture" [IBM, 2011] and the DMTF white papers "Interoperable Clouds" [DMTF, 2009], "Use Cases and Interactions for Managing Clouds" [DMTF, 2010], and "Architecture for Managing Clouds" [DMTF 2010]. The paper "Cloud Computing Synopsis and Recommendations: Recommendations of the National Institute of Standards and Technology" [Badger, 2011] describes the different roles, responsibilities, service level agreements, and risks for subscribers of different cloud models. For more on cloud workloads, see the paper "MAD-MAC: Multiple Attribute Decision Methodology for Adoption of Clouds" [Saripalli and Pingali, 2011]. The paper "Introduction and Architecture Overview: IBM Cloud Computing Reference Architecture 2.0" [Behrendt, et al., 2011] gives an overview of IBM's perspective of cloud computing.

Virtualization

For more on virtualization concepts, see the article "Virtual Linux: An Overview of Virtualization Methods, Architectures, and Implementations" [Jones, 2006] and the book *Running Xen: A Hands-on Guide to the Art of Virtualization* [Matthews, et al., 2008]. The IBM Information Center for Linux, the KVM and QEMU wikis, and the VMWare Technical Resource Center [VMWare, 2011] are good references on virtualization in general, as well as the individual products and projects in specific.

For more on KVM, see the article "Discover the Linux Kernel Virtual Machine" [Jones, 2007]. The article "kvm: The Linux Virtual Machine Monitor" [Kivity, et al., 2007] goes into more depth on the virtualization methods that KVM uses. Also see the Open Virtualization Alliance web site, which promotes use of KVM.

For more on the Open Virtualization Format, see the Distributed Management Task Force Open Virtualization Format home page [DMTF, 2011] for links to the latest standards and white papers. Snapshots and other QCOW image file features are discussed in "The QCOW2 Image Format" [McLoughlin, 2008].

For more on libvirt, see libvirt.org and the article "Anatomy of the libvirt Virtualization Library" [Jones, 2010].

REST and Related Programming APIs

For more on REST development with Java, see the book *RESTful Java with JAX-RS* [Burke, 2009]. See the article "Java Architecture for XML Binding (JAXB)" [Ort and Mehta, 2003] for details about JAXB. See the *Apache Wink User Guide* [Apache Wink Team, 2011] for a step-by-step guide to programming with JAX_RS using Apache Wink. See the Haxx web site to download cURL and for detailed information on using it. Also see related information on the use of cURL in PHP in the PHP manual and on the Zend web site.

For more on JavaScript, especially basic JavaScript functions and HTML DOM, see the Mozilla Developer Network. The *Geko DOM Reference* is especially valuable. Also see the Prototype JavaScript framework and the Dojo Toolkit open source projects. The page "Introduction to Ajax" describes AJAX principles and how to use the Prototype framework to execute AJAX calls.

Operating Systems

For more on Linux features discussed in this book, see the IBM Information Center for Linux [IBM, 2011], the Fedora Wiki [Red Hat, 2011], Red Hat product documentation [Red Hat, 2011], and the openSUSE project [Novell, 2011]. For more on UNIX, see the "FreeBSD Books and Articles Online" web page. For more on software installation and management on SUSE, see the article "Package Management with Zypper" [Frazier, 2009]. For more on interoperability between Linux and Windows, see the Cygwin web site.

Middleware and Development Tools

For more on the IBM software discussed in this book, refer to the IBM DB2 9.7 Database for Linux, UNIX, and Windows Information Center and the WebSphere Application Server Version 7.0 Information Center. For the new features in WebSphere Application Server 7.0, see the articles "System Administration for WebSphere Application Server V7: Part 1: An Overview of Administrative Enhancements" [Apte and Chiu, 2008] and "System Administration for WebSphere Application Server V7: Part 2: New Administrative Topologies [Brady, et al., 2009]. For an overview of using relational databases in a cloud environment, see the article "Database in the Cloud" [Zhang, 2011].

For more on build and deployment automation, see the Apache Maven web site. For more on deployment topologies, see Rational Software Architect 8.0 Help [IBM, 2011] and the article "Create a Deployment Topology Diagram in IBM Rational Software Architect" [Bell, 2009].

For more on basic email configuration, see the article "sendmail: Introduction and Configuration" [Seneca, 2001].

Software Installation, Management, and Deployment Automation

For more on OSGi, see the OSGi Alliance web site. For more on the use of OSGi in Eclipse, see the Eclipse Equinox web site and the article "Understanding How Eclipse Plug-ins Work with OSGi" [Delap, 2006]. The article "Hello, OSGi, Part 1: Bundles for Beginners" [Patil, 2008] gives an example of creating a simple OSGi plug-in. The article "Explore Eclipse's OSGi Console" [Aniszczyk, 2007] explains the use of the OSGi console in Eclipse.

For more on topology modeling, see the articles "Anatomy of a Topology Model in Rational Software Architect Version 7.5: Part 1: Deployment Modeling" [Makin, 2008] and "Extending the Topology Editor with Custom Technology Domains" [Zwanziger, et al., 2010]. For more on automation, see the article "IBM Image Construction and Composition Tool" [Kalantar, et al., 2010].

Remote Displays and Desktops

For more information on remote displays and desktops, see the Xfree86 Resources page, the X Org wiki, and the NoMachine, FreeNX Project, RealVNC, and Cygwin/X web sites.

Security

For more on cloud security, see the white paper "Cloud Security Guidance: IBM Recommendations for the Implementation of Cloud Security" [Buecker, et al., 2009] and also the Cloud Security Alliance web site. For more on Linux security, see the *Security Guide for Red Hat Enterprise Linux 4* [Red Hat, 2008]. For more on SELinux, see the *Fedora 13 Security-Enhanced Linux User Guide* [McAllister, et al., 2010] and the SELinux project wiki [SELinux Project Team]. See the Common Criteria Portal for more about Common Criteria.

For more on OAuth, see the *OAuth 2.0 Authorization Protocol* Draft RFC [Hammer-Lahav, 2011]. You can experiment with OAuth at the Google OAuth Playground. The "Google Internet Identity Research" page has links to a number of Google and external projects, especially those related to social networking.

For more on LDAP, see the *OpenLDAP Software 2.4 Administrator's Guide* [OpenLDAP Project, 2011] and the Red Hat Enterprise Linux documentation. The Identity Commons web site has links to various identity initiatives and workgrounds.

Security of the web layer against cross-site scripting (CSS), SQL injection, and cross-site request forgery (CSRF) is mostly in separate bodies of literature than in general security. For more on CSRF, see the article "Robust Defenses for Cross-Site Request Forgery" [Bath, et al., 2008].

For more on VPN concepts and methods, see the articles "Deliver Cloud Network Control to the User" [Koop, 2010] and "Extend Your Corporate Network with the IBM Cloud" [Rokosz, 2011]. For more on VPN technologies and products, see the OpenVPN, CohesiveFT, and Red Hat Enterprise Linux documentation web pages.

For more on SSH, see the OpenSSH documentation. For more on port forwarding and tunneling, see the articles "SSH Port Forwarding" [Hatch, 2011], "SSH Port Forwarding" [Frields, 2007], and "Tunneling with SSH" [Shewbert, 2006].

Monitoring, Performance, Availability, and Metering

For more on application performance testing and monitoring, see the papers "Load Testing Web Applications Using IBM Rational Performance Tester: Part 1: Overview of Functions, Features, and Reports" [Lee and Tham, 2007] and "Response Time Monitoring Using RPT and ITCAM: Best Practices and Techniques" [Jordan and Vykunta, 2007]. For more on database performance tuning, see the article "Tuning SQL with Optim Query Tuner, Part 1: Understanding Access Paths" [Fuh, Ren, and Zeidenstein, 2011].

For availability, see the blog post, "The AWS Outage: The Cloud's Shining Moment" [Reese, 2011] and the article "High Availability Apps in the IBM Cloud" [Rekesh and Robertson, 2011]. For more on cloud storage and backup, see the materials on the SNIA site, especially the presentations "Backup, Recovery, and Restore" [Baig, 2011] and "Introduction to Data Protection: Backup to Tape, Disk, and Beyond" [Fishman, 2011].

For more on monitoring in the cloud with ITM, see the article “Monitor Services in the Cloud” [Amies, Sanchez, Vernier, and Dong, 2011]. For more on Java monitoring, see the *Java SE Monitoring and Management Guide* [Sun Microsystems, 2006] and the *Java Management Extensions (JMX) Technology Tutorial* [Sun Microsystems, 2006].

For more on metering, see the article “Craft a Cloud Performance Metrics Policy” [Myerson, 2011].

Version Compatibility

For more on language compatibility in general and for XML, see the World Wide Web Consortium Editorial Drafts *Extending and Versioning Languages: Terminology* [Orchard, 2007] and *Extending and Versioning Languages: XML Languages* [Orchard, 2007], and the draft standard *Guide to Versioning XML Languages Using New XML Schema 1.1 Features* [Orchard, 2007]. For more on Java version compatibility, see Chapter 13 in *Binary Compatibility* of the *Java Language Specification* [Sun Microsystems, 2005] and the Eclipse wiki page. See the Eclipse document “Evolving Java-Based APIs” [Rivières, 2007] for more on Java API versioning. For more on the debate about REST versioning, see the blog post “Media Types, Plumbing, and Democracy” [Allamaraju, 2009].

Business Support Systems

For more on metering and billing, see the article “Cloud Metering and Billing” [Meiers, 2011].

References

Allamaraju, S., 2009. “Media Types, Plumbing and Democracy,” at www.subbu.org/blog/2009/12/media-types-and-plumbing.

Amies, A., J. Sanchez, D. Vernier, and X. D. Dong. February 16, 2011. “Monitor Services in the Cloud,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-monitorcloudservices/.

Aniszczyk, C., 2007. “Explore Eclipse’s OSGi Console: Use and Extend the Console That Drives Eclipse,” IBM developerWorks, at www.ibm.com/developerworks/opensource/library/os-ecl-osgiconsole/.

Apache Foundation, 2010. *Apache OpenJPA 2.1 User’s Guide,* at http://openjpa.apache.org/builds/latest/docs/manual/manual.html.

Apache Foundation, 2010. FileUpload project, at http://commons.apache.org/fileupload/.

Apache Hadoop Team. Hadoop web site, at http://hadoop.apache.org/.

Apache Maven Team, 2011. Maven web site, at http://maven.apache.org/.

Apache Wink Team, 2011. *Apache Wink User Guide*, at http://incubator.apache.org/wink/1.1.2/Apache_Wink_User_Guide.pdf.

Apte, A., and B. Chiu, 2008. “System Administration for WebSphere Application Server V7: Part 1: An Overview of Administrative Enhancements,” IBM developerWorks, at www.ibm.com/developerworks/websphere/techjournal/0811_apte/0811_apte.html.

Badger, L., et al., 2011. “Cloud Computing Synopsis and Recommendations: Recommendations of the National Institute of Standards and Technology,” draft, NIST, at http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf.

Baig, A., 2011. “Backup, Recovery, and Restore,” Storage Networking Information Association, at www.snia.org/education/tutorials/2011/spring.

Balfe, R., 2007. “Leave Eclipse Plug-in Headaches Behind with OSGi,” IBM developerWorks, at www.ibm.com/developerworks/opensource/library/os-ecl-dynext/.

Bath, A., C. Jackson, and J. C. Mitchell, 2008. “Robust Defenses for Cross-Site Request Forgery,” Association for Computing Machinery Conference on Computer and Communications Security 2008, at http://seclab.stanford.edu/websec/csrf/csrf.pdf.

Behrendt, M., et al., 2011. “Introduction and Architecture Overview: IBM Cloud Computing Reference Architecture 2.0,” IBM submission to the Open Group at www.opengroup.org/cloudcomputing/.../CCRA.IBMSubmission.02282011.doc.

Bell, D., 2009. “Create a Deployment Topology Diagram in IBM Rational Software Architect,” at www.ibm.com/developerworks/rational/library/09/creatingphysicaltopologydiagramrsa/.

Brady, J., et al., 2009. “System Administration for WebSphere Application Server V7: Part 2: New Administrative Topologies,” IBM developerWorks, at www.ibm.com/developerworks/websphere/techjournal/0901_cundiff/0901_cundiff.html.

Buecker, A., et al., 2009. “Cloud Security Guidance: IBM Recommendations for the Implementation of Cloud Security,” IBM red paper, at www.redbooks.ibm.com/abstracts/redp4614.html.

Burke, W. J., 2009. *RESTful Java with JAX-RS*. O’Reilly.

Cloud Security Alliance. http://cloudsecurityalliance.org.

CohesiveFT, 2011. *VPN-Cubed 2.0* product page, at www.cohesiveft.com/vpncubed/.

Coleman, N., and M. Borret, 2010. “Cloud Security: Who Do You trust?” IBM white paper, at www-03.ibm.com/security/cloud-security.html.

Common Criteria, 2011. “Common Criteria Portal,” at www.commoncriteriaportal.org/.

Criveti, M., 2011. “IBM SmartCloud Enterprise Tip: Configure the Linux Logical Volume Manager,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudtip-lvmconfig/.

Crockford, D. “JSON in Java,” at www.json.org/java/index.html.

Cygwin Project Team, 2011. Cygwin web site, at cygwin.com/index.html.

Cygwin Project Team, 2011. “Cygwin/X Documentation,” at http://xfree86.cygwin.com/docs/.

Deering, S., and R. Hidden, 1998. "Request for Comments: 2460 Internet Protocol, Version 6 (Ipv6)," Internet Engineering Task Force (IETF), at http://tools.ietf.org/html/rfc2460.

Delap, S., 2006. "Understanding How Eclipse Plug-ins Work with OSGi," IBM developerWorks, at www.ibm.com/developerworks/opensource/library/os-ecl-osgi/.

Distributed Management Task Force, 2011. Open Virtualization Format home page, at www.dmtf.org/standards/ovf.

Distributed Management Task Force, 2010. "Architecture for Managing Clouds," a white paper from the Open Cloud Standards Incubator, document number DSP-IS0102, DTMF, at http//:dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf.

Distributed Management Task Force, 2010. *Open Virtualization Format Specification*, version 1.1, at www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf.

Distributed Management Task Force, 2010. "Use Cases and Interactions for Managing Clouds," a white paper from the Open Cloud Standards Incubator, document number DSP-IS0103, at http://dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf.

Distributed Management Task Force, 2009. "Interoperable Clouds," a white paper from the Open Cloud Standards Incubator, document number IDSP-IS0101, DTMF, at www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf.

Distributed Management Task Force, 2009. "Open Virtualization Format," white paper, at www.dmtf.org/sites/default/files/standards/documents/DSP2017_1.0.0.pdf.

Dojo Project Team. Dojo Toolkit, at http://dojotoolkit.org/.

Eclipse Foundation. J2EE Standard Tools Project web site, at www.eclipse.org/webtools/jst/main.php.

Eclipse Foundation, 2011. Equinox web site, at www.eclipse.org/equinox/.

Eucalyptus Systems, Inc., 2011. Eucalyptus web site, at http://open.eucalyptus.com.

Facebook, 2011. "Authentication," at http://developers.facebook.com/docs/authentication/.

Fielding R., J. Gettys, et al., 1999. "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, The Internet Society, at www.w3.org/Protocols/rfc2616/rfc2616.html.

Fielding, R., and R. Taylor, 2002. "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology* 2, no. 2 (May 2002), at www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf.

Fishman, M., 2011. "Introduction to Data Protection: Backup to Tape, Disk, and Beyond," Storage Network Industry Association, at www.snia.org/education/tutorials/2011/spring.

Franks, J., et al., 1999. “RFC 2617: HTTP Authentication: Basic and Digest Access Authentication,” The Internet Society, at http://tools.ietf.org/html/rfc2617.

Frazier, M., 2009. “Package Management with Zypper,” *Linux Journal*, at www.linuxjournal.com/content/package-management-zypper.

FreeBSD Project Team. “FreeBSD Books and Articles Online,” at www.freebsd.org/docs/books.html.

FreeNX Project. FreeNX Project web site, at http://freenx.berlios.de/.

Frields, P., 2007. “SSH Port Forwarding,” *Red Hat Magazine,* at http://magazine.redhat.com/2007/11/06/ssh-port-forwarding/.

Fuh, G., K. Ren, and K. Zeidenstein, 2011. “Tuning SQL with Optim Query Tuner, Part 1: Understanding Access Paths,” IBM developerWorks, at www.ibm.com/developerworks/data/library/techarticle/dm-1006optimquerytuner1/index.html.

Glassfish Community. *JAXB Reference Implementation*, Glassfish, at http://jaxb.java.net.

Goodman, B. D., 2009. “Creating and Customizing Images,” IBM, at https://www.ibm.com/cloud/enterprise.

Google, 2011. “Authentication and Authorization for Google APIs,” at http://code.google.com/apis/accounts/docs/GettingStarted.html.

Google, 2011. “google-gson,” at http://code.google.com/p/google-gson/.

Google. “Chrome Web Store,” at http://chrome.google.com/webstore.

Google. “Internet Identity Research Page,” at http://sites.google.com/site/oauthgoog/Home.

Google. “OAuth Playground,” at http://googlecodesamples.com/oauth_playground/index.php.

Hadley, M., and P. Sandoz, 2009. “JAX-RS: Java API for RESTful Web Services,” JSR 311, Java Community Process, Oracle Corporation, at http://jcp.org/en/jsr/detail?id=311.

Hammer-Lahav, E., 2011. *OAuth 2.0 Authorization Protocol*, IETF Internet Draft, at http://datatracker.ietf.org/doc/draft-ietf-oauth-v2.

Hammer-Lahav, E., 2010. “RFC 5847: The OAuth 1.0 Protocol,” at http://tools.ietf.org/html/rfc5849.

Hammer-Lahav, E., 2010. “OAuth Google Code Project,” at http://code.google.com/p/oauth/.

Hammer-Lahav, E., 2009. “The Authoritative Guide to OAuth 1.0,” at http://hueniverse.com/oauth/guide/.

Hatch, B., 2011. “SSH Port Forwarding,” Symantec, at www.symantec.com/connect/articles/ssh-port-forwarding.

Haxx. cURL web site, at http://curl.haxx.se/.

Hudacek, B., 2011. “Secure Multi-user Access to IBM Cloud Instances with VNC and SSH,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudaccessvncssh/index.html.

IANA, 2001. “MIME Media Types,” at www.iana.org/assignments/media-types/index.html.

IBM, 2011. “Cloud Computing Reference Architecture,” The Open Group, at www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc.

IBM, 2011. *External Image Provider’s Guide*, at www.ibm.com/cloud/enterprise.

IBM, 2011. “IBM DB2 Database for Linux, UNIX, and Windows Information Center,” at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp.

IBM, 2011. *IBM SmartCloud Enterprise: REST API Reference*, Version 1.4.1, at http://ibm.com/cloud/enterprise.

IBM, 2011. *IBM SmartCloud Enterprise: User Guide*, Version 1.4.1, at http://ibm.com/cloud/enterprise.

IBM, 2011. “Information Center for Linux,” at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp.

IBM, 2011. “Open Services for Lifecycle Collaboration,” at http://open-services.net.

IBM, 2011. “Rational Software Architect 8.0 Help,” at http://publib.boulder.ibm.com/infocenter/rsahelp/v8/index.jsp.

IBM, 2011. “Rational Team Concert Downloads,” at http://jazz.net/downloads/rational-team-concert/.

IBM, 2011. “WebSphere Application Server Version 7.0 Information Center,” at http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp.

IBM, 2010. *IBM IaaS Resource Model & REST APIs*. Document version 1.0, submission to the DMTF July 2, 2010.

IBM, 2010. *IBM LotusLive: Partner Platform Guide*.

IBM. “Linux Java Technology,” at www.ibm.com/developerworks/java/jdk/linux/.

Identity Commons, 2011. www.idcommons.net/.

Jersey Project Team. “Glassfish Jersey, JAX-RS Reference Implementation,” at http://jersey.java.net/.

Jones, M. T., 2010. "Anatomy of the libvirt Virtualization Library," IBM developerWorks, at www.ibm.com/developerworks/linux/library/l-libvirt/.

Jones, M. T., 2007. "Discover the Linux Kernel Virtual Machine," IBM developerWorks, at www.ibm.com/developerworks/linux/library/l-linux-kvm/.

Jones, M. T., 2006. "Virtual Linux: An Overview of Virtualization Methods, Architectures, and Implementations," IBM developerWorks, at www.ibm.com/developerworks/library/l-linuxvirt/.

Jordan, L., and R. Vykunta, 2007. "Response Time Monitoring Using RPT and ITCAM: Best Practices and Techniques," IBM, at www-304.ibm.com/software/brandcatalog/ismlibrary/details?catalog.label=1TW10CP19.

Kalantar, M., et al., 2010. "IBM Image Construction and Composition Tool," IBM alphaWorks, at www.alphaworks.ibm.com/tech/iicct.

Kivity, A., et al., 2007. "kvm: the Linux Virtual Machine Monitor," *Proceedings of the Linux Symposium,* at www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf.

Koop, R., 2010. "Deliver Cloud Network Control to the User," IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudvirtualnetwork/.

KVM Project Team. KVM web site, at www.linux-kvm.org.

Lee, F. Y., and A. Tham, 2007. "Load Testing Web Applications Using IBM Rational Performance Tester: Part 1. Overview of Functions, Features, and Reports," IBM developerWorks, at www.ibm.com/developerworks/rational/library/07/1211_lee-tham1/index.html.

Libvirt Project Team. Libvirt web site, at http://libvirt.org/index.html.

Makin, N., 2008. "Anatomy of a Topology Model in Rational Software Architect Version 7.5: Part 1: Deployment Modeling." IBM developerWorks, at www.ibm.com/developerworks/rational/library/08/1202_makin/.

Matthews, J., et al, 2008. *Running Xen: A Hands-on Guide to the Art of Virtualization*, Prentice-Hall.

McAllister, M., et al., 2010. *Fedora 13 Security-Enhanced Linux User Guide*, Edition 1.5, Red Hat, at http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/.

McLoughlin, M., 2008. "The QCOW2 Image Format," at http://people.gnome.org/~markmc/qcow-image-format.html.

Meiers, J., 2011. "Cloud Metering and Billing," IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudmetering/index.html.

Mell, P., and T. Grance, 2009. "The NIST Definition of Cloud Computing," U.S. National Institute of Standards and Technology, at http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

Mozilla Foundation. *Geko DOM Reference*, at http://developer.mozilla.org/en/Gecko_DOM_Reference.

Mozilla Foundation. JavaScript home page, at http://developer.mozilla.org/en/JavaScript.

Mozilla Foundation. "Mozilla Developer Network," at http://developer.mozilla.org.

Mozilla Foundation. "Prism Wiki," at http://wiki.mozilla.org/Prism.

Mozilla Foundation, 2010. "Same Origin Policy for JavaScript," at http://developer.mozilla.org/en/Same_origin_policy_for_JavaScript.

Munin, 2011. Munin Monitoring web site, at http://munin-monitoring.org/.

Myerson, J. M., 2011. "Craft a Cloud Performance Metrics Policy," IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudperformmetrics/index.html.

Nagios, 2011. Nagios web site, at http://nagios.org/.

National Institute of Standards and Technology. Cloud Computing home page, at http://csrc.nist.gov/groups/SNS/cloud-computing/.

Nebel, E., and L. Masinter. 1995. *Form-based File Upload in HTML*, Internet Engineering Task for RFC 1867, at www.ietf.org/rfc/rfc1867.txt.

NoMachine web site, at www.nomachine.com.

Novell, 2011. OpenSUSE Project, at http://en.opensuse.org.

Noyes, K., 2010. "Rackspace's Risky Open Cloud Bet," LinuxInsider, at www.linuxinsider.com/story/70442.html.

OAuth Community. OAuth web site, at http://oauth.net.

Open Virtualization Alliance web site, at www.openvirtualizationalliance.org/.

Open Web Application Security Project. "Top Ten Risks," at www.owasp.org/index.php/Top_10_2010-A5.

OpenID Foundation. OpenID web site, at http://openid.net/.

OpenLDAP Project, 2011. *OpenLDAP Software 2.4 Administrator's Guide*, at www.openldap.org/doc/admin24/index.html.

OpenSSH Project Team, *OpenSSH Documentation*, at www.openssh.com/manual.html.

OpenVPN. Documentation page, at http://openvpn.net/index.php/open-source/documentation.html.

Oracle. *Java SE 6 Documentation,* At http://download.oracle.com/javase/6/docs/index.html.

Orchard, D. (ed.), 2007. *Extending and Versioning Languages: Terminology*, World Wide Web Consortium Editorial Draft, at www.w3.org/2001/tag/doc/versioning-xml.

Orchard, D. (ed.), 2007. *Extending and Versioning Languages: XML Languages*, World Wide Web Consortium Editorial Draft, at www.w3.org/2001/tag/doc/versioning-xml.

Orchard, D. (ed.), 2007. *Guide to Versioning XML Languages Using New XML Schema 1.1 Features,* World Wide Web Consortium Editorial Draft, at www.w3.org/TR/xmlschema-guide2versioning/.

Orchard, D., 2006. "A Theory of Compatible Versions," at www.xml.com/pub/a/2006/12/20/a-theory-of-compatible-versions.html.

Ort, E., and B. Mehta, 2003. "Java Architecture for XML Binding (JAXB)," Oracle Corporation, at www.oracle.com/technetwork/articles/javase/index-140168.html.

OSGi Alliance web site, at www.osgi.org.

Patil, S., 2008. "Hello, OSGi, Part 1: Bundles for Beginners," JavaWorld.com, at www.javaworld.com/javaworld/jw-03-2008/jw-03-osgi1.html.

PHP Documentation Group, 2011. *PHP Manual*, online at http://php.net/manual/en/index.php.

Prototype Team, 2007. "Introduction to Ajax," at www.prototypejs.org/learn/introduction-to-ajax.

Prototype Team. Prototype JavaScript web site, at www.prototypejs.org/.

QEMU Project Team. QEMU wiki, at http://wiki.qemu.org.

Rackspace, 2011. OpenStack wiki, at http://wiki.openstack.org/StartingPage.

RealVNC, 2011. RealVNC web site, at www.realvnc.com.

Red Hat, 2011. Fedora wiki, at http://fedoraproject.org/wiki/Fedora_Project_Wiki.

Red Hat, 2011. *Red Hat Product Documentation*, at http://docs.redhat.com.

Red Hat, 2008. *Security Guide for Red Hat Enterprise Linux 4*, at docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/4/html/Security_Guide/index.html.

Reese, G., 2011. "The AWS Outage: The Cloud's Shining Moment," O'Reilly, at http://broadcast.oreilly.com/2011/04/the-aws-outage-the-clouds-shining-moment.html.

Rekesh, D., and A. Robertson, 2011. “High Availability Apps in the IBM Cloud,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-highavailabilitycloud/.

Rekesh, D., B. Snitzer, and H. Shaikh, 2010. “Get Started with the IBM Smart Business Development and Test on the IBM Cloud,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudstart.html.

Rightscale, 2011. “Cloud Foundry Architecture and Auto-Scaling,” at http://blog.rightscale.com/2011/04/14/cloud-foundry-architecture-and-auto-scaling/.

Rivières, J., 2007. “Evolving Java-Based APIs,” Eclipse wiki, at http://wiki.eclipse.org/index.php/Evolving_Java-based_APIs.

Rokosz, V., 2011. “Extend Your Corporate Network with the IBM Cloud,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-extendnetworkcloud/index.html.

Salesforce. *AppExchange*, at http://appexchange.salesforce.com/home.

Saripalli, P., and G. Pingali, 2011. “MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds,” IEEE 4th International Conference on Cloud Computing, 316-323.

SELinux Project Team. SELinux project wiki, at http://selinuxproject.org/page/Main_Page.

Seneca, E. J., 2001. “sendmail: Introduction and Configuration,” *Linux Journal,* at www.linuxjournal.com/article/5507.

Shah, A., 2008. “DEEP VIRTUE: Kernel-based Virtualization with KVM,” at www.linux-magazine.com/Issues/2008/86/DEEP-VIRTUE.

Shewbert, J., 2006. “Tunneling with SSH.” IBM developerWorks, at www.ibm.com/developerworks/aix/library/au-tunnelingssh/index.html.

Sun Microsystems, 2006. *Java Management Extensions (JMX) Technology Tutorial.* Online.

Sun Microsystems, 2006. *Java SE Monitoring and Management Guide*. Online.

Sun Microsystems, 2005. *Java Language Specification*, 3rd Edition, at http://java.sun.com/docs/books/jls/third_edition/html/binaryComp.html.

Thome, J., 2002. “Using cURL and libcurl with PHP,” Zend Developer Zone, online at http://devzone.zend.com/article/1081.

Thorsten, 2008. “Amazon’s Elastic Block Store Explained,” at http://blog.rightscale.com/2008/08/20/amazon-ebs-explained/.

Twitter 2011. Twitter API wiki, at http://apiwiki.twitter.com.

Twitter 2011. “Twitter Developers,” at http://dev.twitter.com/.

Vernier, D., 2011. “Convert IBM Cloud Image Parameters into Java Using JAXB,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-parameterizejaxb/index.html.

Vernier, D., 2011. “IBM SmartCloud Enterprise Tip: Deploy a Complex Topology,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-clouddeployutility/index.html.

Vernier, D., 2011. “IBM SmartCloud Enterprise Tip: Integrate Your Authentication Policy Using a Proxy,” IBM developerWorks, at www.ibm.com/developerworks/cloud/library/cl-cloudtip-authproxy/index.html.

Weeden, S., A. L. Blair, and S. Chen, 2008. “Developing a Custom Java Module: Tivoli Federated Identity Manager 6.2,” IBM DeveloperWorks, at www.ibm.com/developerworks/tivoli/tutorials/tz-tfimjava/.

White, T., 2011. *Hadoop: The Definitive Guide*, O’Reilly, 2nd Edition.

VMWare, 2011. “Cloud Foundry,” at http://cloudfoundry.org/.

VMWare, 2011. “Technical Resource Center,” at www.vmware.com/technical-resources/.

Xen Project Team. Xen wiki, at http://wiki.xensource.com.

Xfree86. “Resources,” at www.xfree86.org/sos/resources.html.

X.Org Foundation web site, at www.x.org/wiki/.

Zhang, J., 2011. “Database in the Cloud,” IBM developerWorks, at www.ibm.com/developerworks/data/library/dmmag/DMMag_2011_Issue2/cloudDBaaS/index.html

Zwanziger, A., et al., 2010. “Extending the Topology Editor with Custom Technology Domains,” IBM developerWorks, at www.ibm.com/developerworks/rational/library/10/extendingthetopologyeditorwithcustomtechnologydomains/index.html.

Index