SEI SERIES IN SOFTWARE ENGINEERING

DevOps

A Software Architect's Perspective

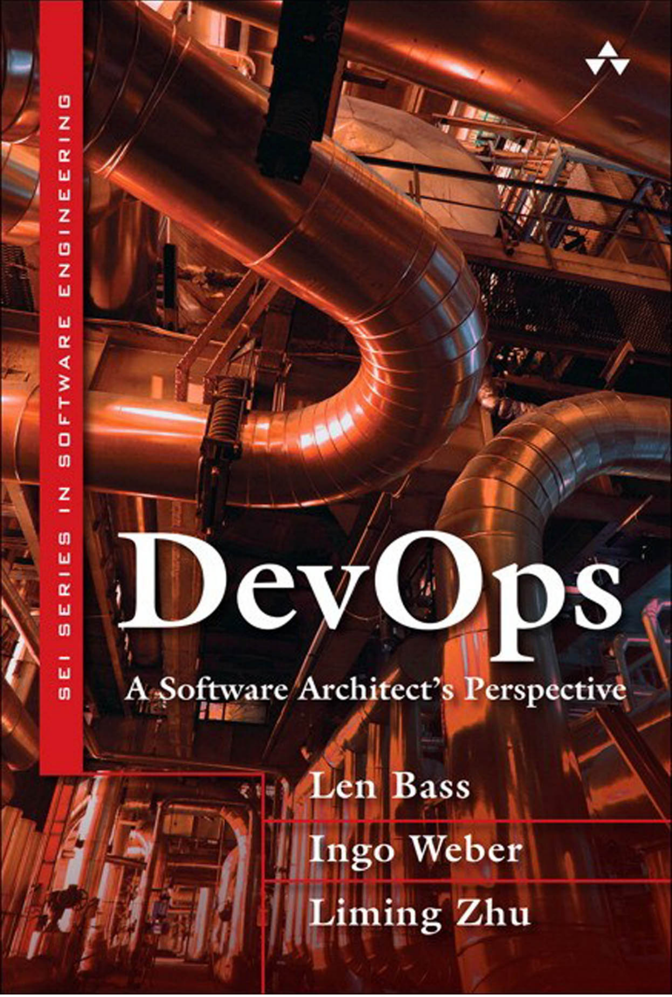
Len Bass

Ingo Weber

Liming Zhu

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



DevOps

The SEI Series in Software Engineering

Software Engineering Institute of Carnegie Mellon University and Addison-Wesley

Software Engineering Institute | Carnegie Mellon University

The CERT Guide to Insider Threats
How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)
Dawn Cappelli
Andrew Moore
Randall Trzeciak
CMMI for Development
Guidelines for Process Integration and Product Improvement
Third Edition
Mary Beth Chrissis
Mike Konrad
Sandy Shrum
TSP
Leading a Development Team
Watts S. Humphrey
Software Architecture in Practice
Third Edition
Len Bass
Paul Clements
Rick Kazman
Model-Based Engineering with AADL
An Introduction to the SAE Architecture Analysis & Design Language
Peter H. Feiler
David P. Gluch

Visit **informit.com/sei** for a complete list of available publications.

The SEI Series in Software Engineering is a collaborative undertaking of the Carnegie Mellon Software Engineering Institute (SEI) and Addison-Wesley to develop and publish books on software engineering and related topics. The common goal of the SEI and Addison-Wesley is to provide the most current information on these topics in a form that is easily usable by practitioners and students.

Titles in the series describe frameworks, tools, methods, and technologies designed to help organizations, teams, and individuals improve their technical or management capabilities. Some books describe processes and practices for developing higher-quality software, acquiring programs for complex systems, or delivering services more effectively. Other books focus on software and system architecture and product-line development. Still others, from the SEI's CERT Program, describe technologies and practices needed to manage software and network security risk. These and all titles in the series address critical problems in software engineering for which practical solutions are available.

Make sure to connect with us!
informit.com/socialconnect

Addison Wesley | informIT the trusted technology learning source | Safari

ALWAYS LEARNING PEARSON

DevOps

A Software Architect's Perspective

Len Bass
Ingo Weber
Liming Zhu

Addison-Wesley

New York • Boston • Indianapolis • San Francisco
Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Software Engineering Institute | Carnegie Mellon

The SEI Series in Software Engineering

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

CMM, CMMI, Capability Maturity Model, Capability Maturity Modeling, Carnegie Mellon, CERT, and CERT Coordination Center are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ATAM; Architecture Tradeoff Analysis Method; CMM Integration; COTS Usage-Risk Evaluation; CURE; EPIC; Evolutionary Process for Integrating COTS Based Systems; Framework for Software Product Line Practice; IDEAL; Interim Profile; OAR; OCTAVE; Operationally Critical Threat, Asset, and Vulnerability Evaluation; Options Analysis for Reengineering; Personal Software Process; PLTP; Product Line Technical Probe; PSP; SCAMPI; SCAMPI Lead Appraiser; SCAMPI Lead Assessor; SCE; SEI; SEPG; Team Software Process; and TSP are service marks of Carnegie Mellon University.

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

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

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

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Bass, Len.
DevOps : a software architect's perspective / Len Bass, Ingo Weber, Liming Zhu.—First [edition].
pages cm.—(The SEI series in software engineering)
Includes bibliographical references and index.
ISBN 978-0-13-404984-7 (hardcover : alk. paper)
1. Software architecture. 2. Computer software—Development.
3. Operating systems (Computers) I. Weber, Ingo M. II. Zhu, Liming, 1975- III. Title.
QA76.76.D47B377 2015
005.1′2—dc23

2015007093

Copyright © 2015 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 200 Old Tappan Road, Old Tappan, New Jersey 07675, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-13-404984-7
ISBN-10: 0-13-404984-5

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.
First printing, May 2015

Contents

Preface *xi*

Previewing the Book *xiii*

Acknowledgments *xv*

Legend *xvii*

PART ONE BACKGROUND 1

CHAPTER 1 What Is DevOps? 3

1.1 Introduction 3
1.2 Why DevOps? 7
1.3 DevOps Perspective 11
1.4 DevOps and Agile 12
1.5 Team Structure 13
1.6 Coordination 17
1.7 Barriers 20
1.8 Summary 23
1.9 For Further Reading 24

CHAPTER 2 The Cloud as a Platform 27

2.1 Introduction 27
2.2 Features of the Cloud 29
2.3 DevOps Consequences of the Unique Cloud Features 41
2.4 Summary 44
2.5 For Further Reading 45

CHAPTER 3 **Operations** **47**

3.1 Introduction 47
3.2 Operations Services 47
3.3 Service Operation Functions 57
3.4 Continual Service Improvement 58
3.5 Operations and DevOps 59
3.6 Summary 61
3.7 For Further Reading 61

PART TWO **THE DEPLOYMENT PIPELINE** **63**

CHAPTER 4 **Overall Architecture** **65**

4.1 Do DevOps Practices Require Architectural Change? 65
4.2 Overall Architecture Structure 66
4.3 Quality Discussion of Microservice Architecture 72
4.4 Amazon's Rules for Teams 75
4.5 Microservice Adoption for Existing Systems 76
4.6 Summary 77
4.7 For Further Reading 78

CHAPTER 5 **Building and Testing** **79**

5.1 Introduction 79
5.2 Moving a System Through the Deployment Pipeline 81
5.3 Crosscutting Aspects 84
5.4 Development and Pre-commit Testing 86
5.5 Build and Integration Testing 91
5.6 UAT/Staging/Performance Testing 95
5.7 Production 96
5.8 Incidents 98
5.9 Summary 98
5.10 For Further Reading 99

CHAPTER 6 **Deployment** **101**

6.1 Introduction 101
6.2 Strategies for Managing a Deployment 102

6.3 Logical Consistency 105
6.4 Packaging 111
6.5 Deploying to Multiple Environments 114
6.6 Partial Deployment 117
6.7 Rollback 118
6.8 Tools 121
6.9 Summary 121
6.10 For Further Reading 122

PART THREE CROSSCUTTING CONCERNS 125

CHAPTER 7 Monitoring 127

7.1 Introduction 127
7.2 What to Monitor 129
7.3 How to Monitor 134
7.4 When to Change the Monitoring Configuration 139
7.5 Interpreting Monitoring Data 139
7.6 Challenges 143
7.7 Tools 147
7.8 Diagnosing an Anomaly from Monitoring Data—the Case of Platformer.com 148
7.9 Summary 152
7.10 For Further Reading 153

CHAPTER 8 Security and Security Audits 155

8.1 What Is Security? 156
8.2 Threats 157
8.3 Resources to Be Protected 159
8.4 Security Roles and Activities 162
8.5 Identity Management 165
8.6 Access Control 169
8.7 Detection, Auditing, and Denial of Service 172
8.8 Development 173
8.9 Auditors 174
8.10 Application Design Considerations 175
8.11 Deployment Pipeline Design Considerations 176

8.12 Summary 177
8.13 For Further Reading 178

CHAPTER 9 Other Ilities 181

9.1 Introduction 181
9.2 Repeatability 183
9.3 Performance 186
9.4 Reliability 188
9.5 Recoverability 190
9.6 Interoperability 191
9.7 Testability 192
9.8 Modifiability 194
9.9 Summary 195
9.10 For Further Reading 196

CHAPTER 10 Business Considerations 197

10.1 Introduction 197
10.2 Business Case 197
10.3 Measurements and Compliance to DevOps Practices 206
10.4 Points of Interaction Between Dev and Ops 209
10.5 Summary 211
10.6 For Further Reading 211

PART FOUR CASE STUDIES 213

CHAPTER 11 Supporting Multiple Datacenters 215

11.1 Introduction 215
11.2 Current State 216
11.3 Business Logic and Web Tiers 216
11.4 Database Tier 220
11.5 Other Infrastructure Tools 223
11.6 Datacenter Switch 225
11.7 Testing 232
11.8 Summary 233
11.9 For Further Reading 234

CHAPTER 12 Implementing a Continuous Deployment Pipeline for Enterprises 237

12.1 Introduction 237
12.2 Organizational Context 238
12.3 The Continuous Deployment Pipeline 240
12.4 Baking Security into the Foundations of the CD Pipeline 257
12.5 Advanced Concepts 259
12.6 Summary 261
12.7 For Further Reading 262

CHAPTER 13 Migrating to Microservices 263

13.1 Introduction to Atlassian 263
13.2 Building a Platform for Deploying Microservices 265
13.3 BlobStore: A Microservice Example 268
13.4 Development Process 273
13.5 Evolving BlobStore 279
13.6 Summary 284
13.7 For Further Reading 284

PART FIVE MOVING INTO THE FUTURE 285

CHAPTER 14 Operations as a Process 287

14.1 Introduction 287
14.2 Motivation and Overview 288
14.3 Offline Activities 289
14.4 Online Activities 294
14.5 Error Diagnosis 296
14.6 Monitoring 296
14.7 Summary 298
14.8 For Further Reading 298

CHAPTER 15 The Future of DevOps 299

15.1 Introduction 299
15.2 Organizational Issues 300
15.3 Process Issues 302

15.4 Technology Issues 305
15.5 What About Error Reporting and Repair? 309
15.6 Final Words 310
15.7 For Further Reading 310

References 311

About the Authors 315

Index 317

Preface

We have been investigating problems in operations for several years and have, naturally, been tracking the DevOps movement. It is moving up the Gartner Hype Curve and has a solid business reason for existing. We were able to find treatments from the IT manager's perspective (e.g., the novel *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*) and from the project manager's perspective (e.g., *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*). In addition, there is a raft of material about cultural change and what it means to tear down barriers between organizational units.

What frustrated us is that there is very little material from the software architect's perspective. Treating operations personnel as first-class stakeholders and listening to their requirements is certainly important. Using tools to support operations and project management is also important. Yet, we had the strong feeling that there was more to it than stakeholder management and the use of tools.

Indeed there is, and that is the gap that this book intends to fill. DevOps presents a fascinating interplay between design, process, tooling, and organizational structure. We try to answer two primary questions: What technical decisions do I, as a software architect, have to make to achieve the DevOps goals? What impact do the other actors in the DevOps space have on me?

The answers are that achieving DevOps goals can involve fundamental changes in the architecture of your systems and in the roles and responsibilities required to get your systems into production and support them once they are there.

Just as software architects must understand the business context and goals for the systems they design and construct, understanding DevOps requires understanding organizational and business contexts, as well as technical and operational contexts. We explore all of these.

The primary audience for this book is practicing software architects who have been or expect to be asked, "Should this project or organization adopt DevOps practices?" Instead of being asked, the architect may be told. As with all books, we expect additional categories of readers. Students who are interested in learning more about the practice of software architecture should find interesting material here. Researchers who wish to investigate DevOps topics can find important background material. Our primary focus, however, is on practicing architects.

This page intentionally left blank

Previewing the Book

We begin the book by discussing the background for DevOps. Part One begins by delving into the goals of DevOps and the problems it is intended to solve. We touch on organizational and cultural issues, as well as the relationship of DevOps practices to agile methodologies.

In Chapter 2, we explore the cloud. DevOps practices have grown in tandem with the growth of the cloud as a platform. The two, in theory, are separable, but in practice virtualization and the cloud are important enablers for DevOps practices.

In our final background chapter, Chapter 3, we explore operations through the prism of the Information Technology Infrastructure Library (ITIL). ITIL is a system of organization of the most important functions of an operations group. Not all of operations are included in DevOps practices but understanding something of the responsibilities of an operations group provides important context, especially when it comes to understanding roles and responsibilities.

Part Two describes the deployment pipeline. We begin this part by exploring the microservice architectural style in Chapter 4. It is not mandatory that systems be architected in this style in order to apply DevOps practices but the microservice architectural style is designed to solve many of the problems that motivated DevOps.

In Chapter 5, we hurry through the building and testing processes and tool chains. It is important to understand these but they are not our focus. We touch on the different environments used to get a system into production and the different sorts of tests run on these environments. Since many of the tools used in DevOps are used in the building and testing processes, we provide context for understanding these tools and how to control them.

We conclude Part Two by discussing deployment. One of the goals of DevOps is to speed up deployments. A technique used to achieve this goal is to allow each development team to independently deploy their code when it is ready. Independent deployment introduces many issues of consistency. We discuss different deployment models, managing distinct versions of a system that are simultaneously in production, rolling back in the case of errors, and other topics having to do with actually placing your system in production.

Part Two presents a functional perspective on deployment practices. Yet, just as with any other system, it is frequently the quality perspectives that control the design and the acceptance of the system. In Part Three, we focus on crosscutting concerns. This begins with our discussion of monitoring and live testing in Chapter 7. Modern software testing practices do not end when a system is placed into production. First, systems are monitored extensively to detect problems, and secondly, testing continues in a variety of forms after a system has been placed into production.

Another crosscutting concern is security, which we cover in Chapter 8. We present the different types of security controls that exist in an environment, spanning those that are organization wide and those that are specific system wide. We discuss the different roles associated with achieving security and how these roles are evaluated in the case of a security audit.

Security is not the only quality of interest, and in Chapter 9 we discuss other qualities that are relevant to the practices associated with DevOps. We cover topics such as performance, reliability, and modifiability of the deployment pipeline.

Finally, in Part Three we discuss business considerations in Chapter 10. Practices as broad as DevOps cannot be adopted without buy-in from management. A business plan is a typical means of acquiring this buy-in; thus, we present the elements of a business plan for DevOps adoption and discuss how the argument, rollout, and measurement should proceed.

In Part Four we present three case studies. Organizations that have implemented DevOps practices tell us some of their tricks. Chapter 11 discusses how to maintain two datacenters for the purpose of business continuity; Chapter 12 presents the specifics of a continuous deployment pipeline; and Chapter 13 describes how one organization is migrating to a microservice architecture.

We close by speculating about the future in Part Five. Chapter 14 describes our research and how it is based on viewing operations as a series of processes, and Chapter 15 gives our prediction for how the next three to five years are going to evolve in terms of DevOps.

Acknowledgments

Books like this require a lot of assistance. We would like to thank Chris Williams, John Painter, Daniel Hand, and Sidney Shek for their contributions to the case studies, as well as Adnene Guabtni, Kanchana Wickremasinghe, Min Fu, and Xiwei Xu for helping us with some of the chapters.

Manuel Pais helped us arrange case studies. Philippe Kruchten, Eoin Woods, Gregory Hartman, Sidney Shek, Michael Lorant, Wouter Geurts, and Eltjo Poort commented on or contributed to various aspects of the book.

We would like to thank Jean-Michel Lemieux, Greg Warden, Robin Fernandes, Jerome Touffe-Blin, Felipe Cuozzo, Pramod Korathota, Nick Wright, Vitaly Osipov, Brad Baker, and Jim Watts for their comments on Chapter 13.

Addison-Wesley did their usual professional and efficient job in the production process, and this book has benefited from their expertise.

Finally, we would like to thank NICTA and NICTA management. NICTA is funded by the Australian government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. Without their generous support, this book would not have been written.

This page intentionally left blank

Legend

We use four distinct legends for the figures. We have an architectural notation that identifies the key architectural concepts that we use; we use Business Process Model and Notation (BPMN) to describe some processes, Porter's Value Notation to describe a few others, and UML sequence diagrams for interleaving sequences of activities. We do not show the UML sequence diagram notation here but the notation that we use from these other sources is:

Architecture

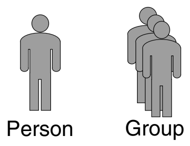


FIGURE P.1 People, both individual and groups

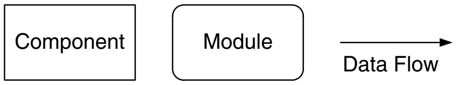


FIGURE P.2 Components (runtime entities), modules (code-time collections of entities), and data flow

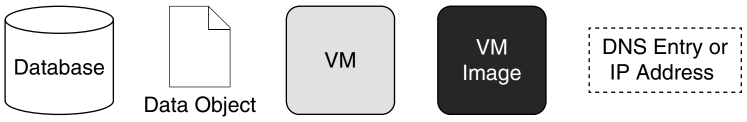


FIGURE P.3 Specialized entities

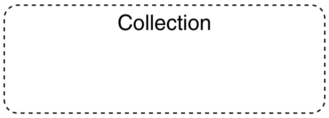


FIGURE P.4 Collections of entities

BPMN

We use Business Process Model and Notation (BPMN) for describing events and activities [OMG 11].

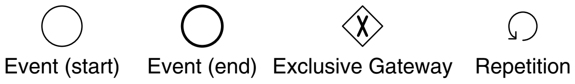


FIGURE P.5 Event indications

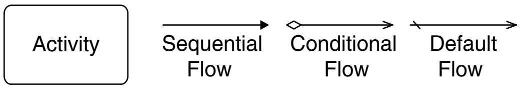


FIGURE P.6 Activities and sequences of activities

Porter's Value Chain

This notation is used to describe processes (which, in turn, have activities modelled in BPMN).

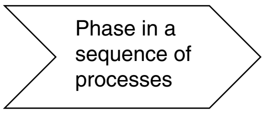


FIGURE P.7 Entry in a value chain

This page intentionally left blank

4

Overall Architecture

A distributed system is one in which the failure of a computer you didn't even know existed can render you own computer unusable.
—Leslie Lamport

In this chapter we begin to see the structural implications of the DevOps practices. These practices have implications with respect to both the overall structure of the system and techniques that should be used in the system's elements. DevOps achieves its goals partially by replacing explicit coordination with implicit and often less coordination, and we will see how the architecture of the system being developed acts as the implicit coordination mechanism. We begin by discussing whether DevOps practices necessarily imply architectural change.

4.1 Do DevOps Practices Require Architectural Change?

You may have a large investment in your current systems and your current architecture. If you must re-architect your systems in order to take advantage of DevOps, a legitimate question is "Is it worth it?" In this section we see that some DevOps practices are independent of architecture, whereas in order to get the full benefit of others, architectural refactoring may be necessary.

Recall from Chapter 1 that there are five categories of DevOps practices.

1. Treat Ops as first-class citizens from the point of view of requirements. Adding requirements to a system from Ops may require some architectural modification. In particular, the Ops requirements are likely to be in the area of logging, monitoring, and information to support incident handling. These requirements will be like other requirements for modifications to a

system: possibly requiring some minor modifications to the architecture but, typically, not drastic modifications.

2. Make Dev more responsible for relevant incident handling. By itself, this change is just a process change and should require no architectural modifications. However, just as with the previous category, once Dev becomes aware of the requirements for incident handling, some architectural modifications may result.
3. Enforce deployment process used by all, including Dev and Ops personnel. In general, when a process becomes enforced, some individuals may be required to change their normal operating procedures and, possibly, the structure of the systems on which they work. One point where a deployment process could be enforced is in the initiation phase of each system. Each system, when it is initialized, verifies its pedigree. That is, it arrived at execution through a series of steps, each of which can be checked to have occurred. Furthermore, the systems on which it depends (e.g., operating systems or middleware) also have verifiable pedigrees.
4. Use continuous deployment. Continuous deployment is the practice that leads to the most far-reaching architectural modifications. On the one hand, an organization can introduce continuous deployment practices with no major architectural changes. See, for example, our case study in Chapter 12. On the other hand, organizations that have adopted continuous deployment practices frequently begin moving to a microservice-based architecture. See, for example, our case study in Chapter 13. We explore the reasons for the adoption of a microservice architecture in the remainder of this chapter
5. Develop infrastructure code with the same set of practices as application code. These practices will not affect the application code but may affect the architecture of the infrastructure code.

4.2 Overall Architecture Structure

Before delving into the details of the overall structure, let us clarify how we use certain terminology. The terms *module* and *component* are frequently overloaded and used in different fashions in different writings. For us, a module is a code unit with coherent functionality. A component is an executable unit. A compiler or interpreter turns modules into binaries, and a builder turns the binaries into components. The development team thus directly develops modules. Components are results of the modules developed by development teams, and so it is possible to speak of a team developing a component, but it should be clear that the development of a component is an indirect activity of a development team.

As we described in Chapter 1, development teams using DevOps processes are usually small and should have limited inter-team coordination. Small teams imply that each team has a limited scope in terms of the components they develop. When a team deploys a component, it cannot go into production unless the component is compatible with other components with which it interacts. This compatibility can be ensured explicitly through multi-team coordination, or it can be ensured implicitly through the definition of the architecture.

An organization can introduce continuous deployment without major architectural modifications. For example, the case study in Chapter 12 is fundamentally architecture-agnostic. Dramatically reducing the time required to place a component into production, however, requires architectural support:

- Deploying without the necessity of explicit coordination with other teams reduces the time required to place a component into production.
- Allowing for different versions of the same service to be simultaneously in production leads to different team members deploying without coordination with other members of their team.
- Rolling back a deployment in the event of errors allows for various forms of live testing.

Microservice architecture is an architectural style that satisfies these requirements. This style is used in practice by organizations that have adopted or inspired many DevOps practices. Although project requirements may cause deviations to this style, it remains a good general basis for projects that are adopting DevOps practices.

A microservice architecture consists of a collection of services where each service provides a small amount of functionality and the total functionality of the system is derived from composing multiple services. In Chapter 6, we also see that a microservice architecture, with some modifications, gives each team the ability to deploy their service independently from other teams, to have multiple versions of a service in production simultaneously, and to roll back to a prior version relatively easily.

Figure 4.1 describes the situation that results from using a microservice architecture. A user interacts with a single consumer-facing service. This service, in turn, utilizes a collection of other services. We use the terminology *service* to refer to a component that provides a service and *client* to refer to a component that requests a service. A single component can be a client in one interaction and a service in another. In a system such as LinkedIn, the service depth may reach as much as 70 for a single user request.

Having an architecture composed of small services is a response to having small teams. Now we look at the aspects of an architecture that can be specified globally as a response to the requirement that inter-team coordination be minimized. We discuss three categories of design decisions that can be made globally as a portion of the architecture design, thus removing the need for inter-team

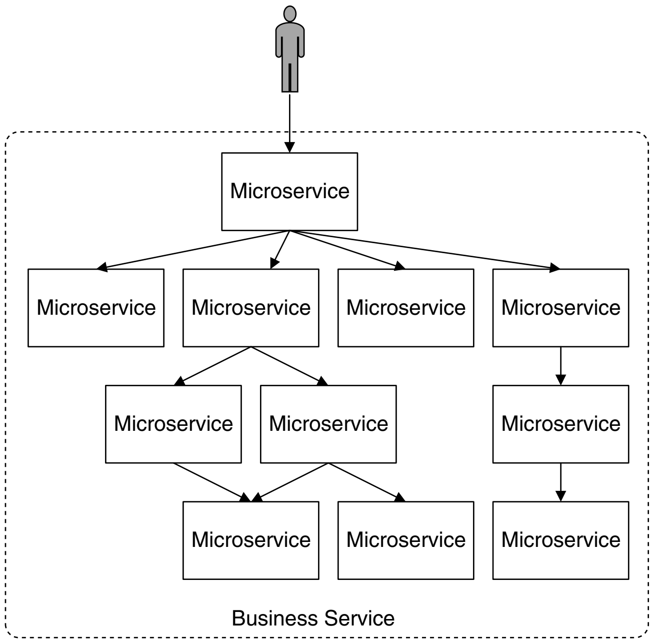


FIGURE 4.1 User interacting with a single service that, in turn, utilizes multiple other services [Notation: Architecture]

coordination with respect to these decisions. The three categories are: the coordination model, management of resources, and mapping among architectural elements.

Coordination Model

If two services interact, the two development teams responsible for those services must coordinate in some fashion. Two details of the coordination model that can be included in the overall architecture are: how a client discovers a service that it wishes to use, and how the individual services communicate.

Figure 4.2 gives an overview of the interaction between a service and its client. The service registers with a registry. The registration includes a name for the service as well as information on how to invoke it, for example, an endpoint location as a URL or an IP address. A client can retrieve the information about the service from the registry and invoke the service using this information. If the registry provides IP addresses, it acts as a local DNS server—local, because typically, the registry is not open to the general Internet but is within the environment of the application. Netflix Eureka is an example of a cloud service registry that acts as a DNS server. The registry serves as a catalogue of available services, and

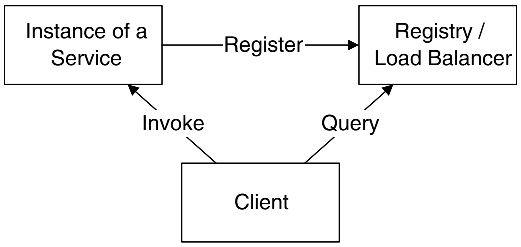


FIGURE 4.2 An instance of a service registers itself with the registry, the client queries the registry for the address of the service and invokes the service. [Notation: Architecture]

can further be used to track aspects such as versioning, ownership, service level agreements (SLAs), etc., for the set of services in an organization. We discuss extensions to the registry further in Chapter 6.

There will typically be multiple instances of a service, both to support a load too heavy for a single instance and to guard against failure. The registry can rotate among the instances registered to balance the load. That is, the registry acts as a load balancer as well as a registry. Finally, consider the possibility that an instance of a service may fail. In this case, the registry should not direct the client to the failed instance. By requiring the service to periodically renew its registration or proactively checking the health of the service, a guard against failure is put in place. If the service fails to renew its registration within the specified period, it is removed from the registry. Multiple instances of the service typically exist, and so the failure of one instance does not remove the service. The above-mentioned Netflix Eureka is an example for a registry offering load balancing. Eureka supports the requirement that services periodically renew their registration.

The protocol used for communication between the client and the service can be any remote communication protocol, for example, HTTP, RPC, SOAP, etc. The service can provide a RESTful interface or not. The remote communication protocol should be the only means for communication among the services. The details of the interface provided by the service still require cross-team coordination. When we discuss the example of Amazon later, we will see one method of providing this coordination. We will also see an explicit requirement for restricting communication among services to the remote communication protocol.

Management of Resources

Two types of resource management decisions can be made globally and incorporated in the architecture—provisioning/deprovisioning VMs and managing variation in demand.

Provisioning and Deprovisioning VMs

New VMs can be created in response to client demand or to failure. When the demand subsides, instances should be deprovisioned. If the instances are stateless (i.e., they do not retain any information between requests), a new instance can be placed into service as soon as it is provisioned. Similarly, if no state is kept in an instance, deprovisioning becomes relatively painless: After a cool-down period where the instance receives no new requests and responds to existing ones, the instance can be deprovisioned. The cool-down period should therefore be long enough for an instance to respond to all requests it received (i.e., the backlog). If you deprovision an instance due to reduced demand, the backlog should be fairly small—in any other case this action needs to be considered carefully. An additional advantage of a stateless service is that messages can be routed to any instance of that service, which facilitates load sharing among the instances.

This leads to a global decision to maintain state external to a service instance. As discussed in Chapter 2, large amounts of application state can be maintained in persistent storage, small amounts of application state can be maintained by tools such as ZooKeeper, and client state should not be maintained on the provider's side anyway.

Determining which component controls the provisioning and deprovisioning of a new instance for a service is another important aspect. Three possibilities exist for the controlling component.

1. A service itself can be responsible for (de)provisioning additional instances. A service can know its own queue lengths and its own performance in response to requests. It can compare these metrics to thresholds and (de)provision an instance itself if the threshold is crossed. Assuming that the distribution of requests is fair, in some sense, across all instances of the service, one particular instance (e.g., the oldest one) of the service can make the decision when to provision or deprovision instances. Thus, the service is allowed to expand or shrink capacity to meet demand.
2. A client or a component in the client chain can be responsible for (de)provisioning instances of a service. For instance, the client, based on the demands on it, may be aware that it will shortly be making demands on the service that exceed a given threshold and provisions new instances of the service.
3. An external component monitors the performance of service instances (e.g., their CPU load) and (de)provisions an instance when the load reaches a given threshold. Amazon's autoscaling groups provide this capability, in collaboration with the CloudWatch monitoring system.

Managing Demand

The number of instances of an individual service that exist should reflect the demand on the service from client requests. We just discussed several different methods for provisioning and deprovisioning instances, and these methods make different assumptions about how demand is managed.

- One method for managing demand is to monitor performance. Other decisions to be made include determining how to implement monitoring (e.g., whether done internally by running a monitoring agent inside each service instance or externally by a specialized component). That is, when demand grows that needs to be detected, a new instance can be provisioned. It takes time to provision a new instance, so it is important that the indicators are timely and even predictive to accommodate for that time. We discuss more details about monitoring in Chapter 7.
- Another possible technique is to use SLAs to control the number of instances. Each instance of the service guarantees through its SLAs that it is able to handle a certain number of requests with a specified latency. The clients of that service then know how many requests they can send and still receive a response within the specified latency. This technique has several constraints. First, it is likely that the requirements that a client imposes on your service will depend on the requirements imposed on the client, so there is a cascading effect up through the demand chain. This cascading will cause uncertainty in both the specification and the realization of the SLAs. A second constraint of the SLA technique is that each instance of your service may know how many requests it can handle, but the client has multiple available instances of your service. Thus, the provisioning component has to know how many instances currently exist of your service.

Mapping Among Architectural Elements

The final type of coordination decision that can be specified in the architecture is the mapping among architectural elements. We discuss two different types of mappings—work assignments and allocation. Both of these are decisions that are made globally.

- *Work assignments*. A single team may work on multiple modules, but having multiple development teams work on the same module requires a great deal of coordination among those development teams. Since coordination takes time, an easier structure is to package the work of a single team into modules and develop interfaces among the modules to allow modules developed by different teams to interoperate. In fact, the original definition of a module by David Parnas in the 1970s was as a work assignment of a team. Although not required, it is reasonable that each component (i.e., microservice) is the responsibility of a single development team. That is, the set of modules that, when linked, constitute a component are the output of a single development team. This does not preclude a single development team from being responsible for multiple components but it means that any coordination involving a component is settled within a single development team, and that any coordination involving multiple development teams goes across components. Given the set of constraints on the architecture we are describing, cross-team coordination requirements are limited.

- *Allocation*. Each component (i.e., microservice) will exist as an independent deployable unit. This allows each component to be allocated to a single (virtual) machine or container, or it allows multiple components to be allocated to a single (virtual) machine. The redeployment or upgrade of one microservice will not affect any other microservices. We explore this choice in Chapter 6.

4.3 Quality Discussion of Microservice Architecture

We have described an architectural style—microservice architecture—that reduces the necessity for inter-team coordination by making global architectural choices. The style provides some support for the qualities of dependability (stateless services) and modifiability (small services), but there are additional practices that a team should use to improve both dependability and modifiability of their services.

Dependability

Three sources for dependability problems are: the small amount of inter-team coordination, correctness of environment, and the possibility that an instance of a service can fail.

Small Amount of Inter-team Coordination

The limited amount of inter-team coordination may cause misunderstandings between the team developing a client and the team developing a service in terms of the semantics of an interface. In particular, unexpected input to a service or unexpected output from a service can happen. There are several options. First, a team should practice defensive programming and not assume that the input or the results of a service invocation are correct. Checking values for reasonableness will help detect errors early. Providing a rich collection of exceptions will enable faster determination of the cause of an error. Second, integration and end-to-end testing with all or most microservices should be done judiciously. It can be expensive to run these tests frequently due to the involvement of a potentially large number of microservices and realistic external resources. A testing practice called Consumer Driven Contract (CDC) can be used to alleviate the problem.

That is, the test cases for testing a microservice are decided and even co-owned by all the *consumers* of that microservice. Any changes to the CDC test cases need to be agreed on by both the consumers and the developers of the microservice. Running the CDC test cases, as a form of integration testing, is less expensive than running end-to-end test cases. If CDC is practiced properly, confidence in the microservice can be high without running many end-to-end test cases.

CDC serves as a method of coordination and has implications on how user stories of a microservice should be made up and evolve over time. Consumers and microservice developers collectively make up and own the user stories. CDC definition becomes a function of the allocation of functionality to the microservice, is managed by the service owner as a portion of the coordination that defines the next iteration, and, consequently, does not delay the progress of the current iteration.

Correctness of Environment

A service will operate in multiple different environments during the passage from unit test to post-production. Each environment is provisioned and maintained through code and a collection of configuration parameters. Errors in code and configuration parameters are quite common. Inconsistent configuration parameters are also possible. Due to a degree of uncertainty in cloud-based infrastructure, even executing the correct code and configuration may lead to an incorrect environment. Thus, the initialization portion of a service should test its current environment to determine whether it is as expected. It should also test the configuration parameters to detect, as far as possible, unexpected inconsistencies from different environments. If the behavior of the service depends on its environment (e.g., certain actions are performed during unit test but not during production), then the initialization should determine the environment and provide the settings for turning on or off the behavior. An important trend in DevOps is to manage all the code and parameters for setting up an environment just as you manage your application code, with proper version control and testing. This is an example of "infrastructure-as-code" as defined in Chapter 1 and discussed in more detail in Chapter 5. The testing of infrastructure code is a particularly challenging issue. We discuss the issues in Chapters 7 and 9.

Failure of an Instance

Failure is always a possibility for instances. An instance is deployed onto a physical machine, either directly or through the use of virtualization, and in large datacenters, the failure of a physical machine is common. The standard method through which a client detects the failure of an instance of a service is through the timeout of a request. Once a timeout has occurred, the client can issue a request again and, depending on the routing mechanism used, assume it is routed to a different instance of the service. In the case of multiple timeouts, the service is assumed to have failed and an alternative means of achieving the desired goal can be attempted.

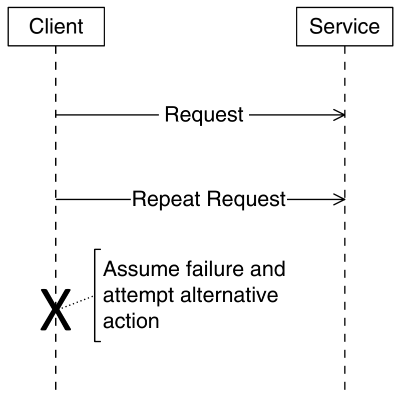


FIGURE 4.3 Time line in recognizing failure of a dependent service [Notation: UML Sequence Diagram]

Figure 4.3 shows a time line for a client attempting to access a failed service. The client makes a request to the service, and it times out. The client repeats the request, and it times out again. At this point, recognizing the failure has taken twice the timeout interval. Having a short timeout interval (failing fast) will enable a more rapid response to the client of the client requesting the service. A short timeout interval may, however, introduce false positives in that the service instance may just be slow for some reason. The result may be that both initial requests for service actually deliver the service, just not in a timely fashion. Another result may be that the alternative action is performed as well. Services should be designed so that multiple invocations of the same service will not introduce an error. *Idempotent* is the term for a service that can be repeatedly invoked with the same input and always produces the same output—namely, no error is generated.

Another point highlighted in Figure 4.3 is that the service has an alternative action. That is, the client has an alternative action in case the service fails. Figure 4.3 does not show what happens if there is no alternative action. In this case, the service reports failure to its client together with context information—namely, no response from the particular underlying service. We explore the topic of reporting errors in more depth in Chapter 7.

Modifiability

Making a service modifiable comes down to making likely changes easy and reducing the ripple effects of those changes. In both cases, a method for making the service more modifiable is to encapsulate either the affected portions of a likely change or the interactions that might cause ripple effects of a change.

Identifying Likely Changes

Some likely changes that come from the development process, rather than the service being provided, are:

- *The environments within which a service executes*. A module goes through unit tests in one environment, integration tests in another, acceptance tests in a third, and is in production in a fourth.
- *The state of other services with which your service interacts*. If other services are in the process of development, then the interfaces and semantics of those services are likely to change relatively quickly. Since you may not know the state of the external service, a safe practice is to treat, as much as possible, all communication with external services as likely to change.
- *The version of third-party software and libraries that are used by your service*. Third-party software and libraries can change arbitrarily, sometimes in ways that are disruptive for your service. In one case we heard, an external system removed an essential interface during the time the deployment process was ongoing. Using the same VM image in different environments will protect against those changes that are contained within the VM but not against external system changes.

Reducing Ripple Effects

Once likely changes have been discovered, you should prevent these types of changes from rippling through your service. This is typically done by introducing modules whose sole purpose is to localize and isolate changes to the environment, to other services, or to third-party software or libraries. The remainder of your service interacts with these changeable entities through the newly introduced modules with stable interfaces.

Any interaction with other services, for example, is mediated by the special module. Changes to the other services are reflected in the mediating module and buffered from rippling to the remainder of your service. Semantic changes to other services may, in fact, ripple, but the mediating module can absorb some of the impact, thereby reducing this ripple effect.

4.4 Amazon's Rules for Teams

As we mentioned in Chapter 1, Amazon has a rule that no team should be larger than can be fed with two pizzas; in the early years of this century they adopted an internal microservice architecture. Associated with the adoption was a list of rules to follow about how to use the services:

- "All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-service/team communication allowed: no direct linking, no direct reads of another team's datastore, no shared-memory model, no backdoors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they [other services] use.
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world."

Each team produces some number of services. Every service is totally encapsulated except for its public interface. If another team wishes to use a service, it must discover the interface. The documentation for the interface must include enough semantic information to enable the user of a service to determine appropriate definitions for items such as "customer" or "address." These concepts can sometimes have differing meanings within different portions of an organization. The semantic information about an interface can be kept in the registry/load balancer that we described earlier, assuming that the semantic information is machine interpretable.

By making every service potentially externally available, whether or not to offer a service globally or keep it local becomes a business decision, not a technical one. External services can be hidden behind an application programming interface (API) bound through a library, and so this requirement is not prejudging the technology used for the interface.

A consequence of these rules is that Amazon has an extensive collection of services. A web page from their sales business makes use of over 150 services. Scalability is managed by each service individually and is included in its SLA in the form of a guaranteed response time given a particular load. The contract covers what the service promises against certain demand levels. The SLA binds both the client side and the service side. If the client's demand exceeds the load promised in the SLA, then slow response times become the client's problem, not the service's.

4.5 Microservice Adoption for Existing Systems

Although microservices reflect the small, independent team philosophy of DevOps, most organizations have large mission-critical systems that are not architected that way. These organizations need to decide whether to migrate their architectures to microservice architectures, and which ones to migrate. We discuss this migration somewhat in Chapter 10. Some of the things an architect thinking of adopting a microservice architecture should ensure are the following:

- Operational concerns are considered during requirements specification.
- The overarching structure of the system being developed should be a collection of small, independent services.
- Each service should be distrustful of both clients and other required services.
- Team roles have been defined and are understood.
- Services are required to be registered with a local registry/load balancer.
- Services must renew their registration periodically.
- Services must provide SLAs for their clients.
- Services should aim to be stateless and be treated as transient.
- If a service has to maintain state, it should be maintained in external persistent storage.
- Services have alternatives in case a service they depend on fails.
- Services have defensive checks to intercept erroneous input from clients and output from other services.
- Uses of external services, environmental information, and third-party software and libraries are localized (i.e., they require passage through a module specific to that external service, environment information, or external software or library).

However, adopting a microservice architecture will introduce new challenges. When an application is composed of a large number of network-connected microservices, there can be latency and other performance issues. Authentication and authorization between services need to be carefully designed so that they do not add intolerable overhead. Monitoring, debugging, and distributed tracing tools may need to be modified to suit microservices. As mentioned earlier, end-to-end testing will be expensive. Rarely can you rebuild your application from scratch without legacy components or existing data.

Migrating from your current architecture to a microservice architecture incrementally without data loss and interruption is a challenge. You may need to build interim solutions during this migration. We discuss these challenges and some solutions in the Atlassian case study in Chapter 13, wherein Atlassian describes the initial steps of their journey to a microservice architecture. An architect should have a checklist of things to consider when performing a migration.

4.6 Summary

The DevOps goal of minimizing coordination among various teams can be achieved by using a microservice architectural style where the coordination mechanism, the resource management decisions, and the mapping of architectural elements are all specified by the architecture and hence require minimal inter-team coordination.

A collection of practices for development can be added to the microservice architectural style to achieve dependability and modifiability, such as identifying and isolating areas of likely change.

Adopting a microservice architectural style introduces additional challenges in monitoring, debugging, performance management, and testing. Migrating from an existing architecture to a microservice architectural style requires careful planning and commitment.

4.7 For Further Reading

For more information about software architecture, we recommend the following books:

- *Documenting Software Architectures, 2nd Edition* [Clements 10]
- *Software Architecture in Practice, 3rd Edition* [Bass 13]

Service description, cataloguing, and management are discussed in detail in the *Handbook of Service Description* [Barros 12]. This book describes services that are externally visible, not microservices, but much of the discussion is relevant to microservices as well.

The microservice architectural style is described in the book *Building Microservices: Designing Fine-Grained Systems* [Newman 15].

Many organizations are already practicing a version of the microservice architectural development and DevOps, and sharing their valuable experiences.

- You can read more about the Amazon example here: http://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/ and http://www.zdnet.com/blog/storage/soa-done-right-the-amazon-strategy/152
- Netflix points out some challenges in using microservice architecture at scale [Tonse 14].

The Netflix implementation of Eureka—their open source internal load balancer/registry—can be found at https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance

Consumer Driven Contracts (CDCs) are discussed in Martin Fowler's blog "Consumer-Driven Contracts: A Service Evolution Pattern," [Fowler 06].

About the Authors

Len Bass is a senior principal researcher at National ICT Australia Ltd. (NICTA). He joined NICTA in 2011 after 25 years at the Software Engineering Institute (SEI) at Carnegie Mellon University. He is the coauthor of two award-winning books in software architecture—*Software Architecture in Practice, Third Edition* (Addison-Wesley 2013) and *Documenting Software Architectures: Views and Beyond*, *Second Edition* (Addison-Wesley 2011)—as well as several other books and numerous papers in computer science and software engineering on a wide range of topics. Len has more than 50 years' experience in software development and research, which has resulted in papers on operating systems, database management systems, user interface software, software architecture, product line systems, and computer operations. He has worked or consulted in multiple domains, including scientific analysis, embedded systems, and information and financial systems.

Ingo Weber is a senior researcher in the Software Systems Research Group at NICTA in Sydney, Australia, as well as an adjunct senior lecturer at CSE at the University of New South Wales (UNSW). Prior to NICTA, Ingo held positions at UNSW and at SAP Research Karlsruhe, Germany. His research interests include cloud computing, DevOps, business process management, and artificial intelligence (AI). He has published over 60 peer-reviewed papers, and served as a reviewer or program committee member for many prestigious scientific journals and conferences. Ingo holds a Ph.D. and a Diploma from the University of Karlsruhe, and an MSc from the University of Massachusetts at Amherst.

Liming Zhu is a research group leader and principal researcher at NICTA. He holds conjoint positions at the University of New South Wales (UNSW) and the University of Sydney. Liming has published over 80 peer-reviewed papers. He formerly worked in several technology lead positions in the software industry before obtaining a Ph.D. in software engineering from UNSW. He is a committee member of the Standards Australia IT-015 (system and software engineering), contributing to ISO/SC7. Liming's research interests include software architecture and dependable systems.

This page intentionally left blank

Index

A/B testing
in deployment, 118
future of DevOps and, 307
introduction to, 96
user interactions and, 132–133
Access control
authentication for. *See* Authentication
boundary definition for, 170
encryption for, 171
introduction to, 169–170
isolation for, 170–171
outdated data in, 171
ownership/responsibility for, 172
prevention of access in, 170–172
in Sourced Group case study, 258
version updating in, 171–172
vulnerability patches in, 171–172
Access control lists (ACLs), 168
Address Resolution Protocol (ARP), 227
Adoption of DevOps
barriers to, generally, 20
department type and, 22
existing systems and, 76–77
organizational culture/type and, 20–22
personnel issues in, 23
silo mentality *vs.*, 22–23
tool support and, 23
Advertising-based charging models, 303
Agent-based/agentless monitoring, 136–137
Agile, 12–13
Akamai CDN, 228, 230
Alarms
automating configuration of, 144
in CloudWatch, 250
false, 128
in intrusion detection, 134–135
in monitoring, generally, 135, 141–142
Alerts
automating configuration of, 144
in monitoring configurations, 139
in monitoring, generally, 141–142
ALIASes, 250–252
All-encompassing processes, 302
Allocation
in capacity planning, 132
CDC definition for, 73
Janitor Monkey and, 43
in microservice architecture, 72, 183
of responsibilities, 19
of shards, 224
of virtual machines, 104
Amazon
AMIs of. *See* Amazon Machine Images (AMIs)
CloudFormation by. *See* CloudFormation (CF)
cross-team coordination by, 69
on datacenter failures, 35
DynamoDB by. *See* DynamoDB
EC2 by. *See* Elastic Compute Cloud (EC2)
future of DevOps and, 299
Glacier by, 139
long tails in, 37
RDS replicas and, 116
repeatability at, 184
shared microservices and, 283
Simple Queue Service by, 258
team rules of, 14, 75–76
version control and, 186
web services by. *See* Amazon Web Services (AWS)
Amazon Machine Images (AMIs). *See also* Virtual machines (VMs)
defined, 30
in deployment pipelines, 92
operations process and, 291
in Sourced Group case study, 245, 248–249
Amazon Web Services (AWS)
Asgard on, 289–295
in Atlassian case study, 264–265
autoscaling in. *See* Autoscaling groups (ASGs)
BlobStore and, 268–271
CDP tooling and, 241
charging models and, 304
Cloud, 175
CloudFormation. *See* CloudFormation (CF)
CloudTrail, 292–295
CodeDeploy, 261
Direct Connect, 260, 264
introduction to, 237
load balancing in. *See* Elastic Load Balancing (ELB)
native services in, 261
Ambler, Scott, 14
AMIs. *See* Amazon Machine Images (AMIs)
Apache Maven, 82
Apdex (Application Performance Index), 148
Application Performance Index (Apdex), 148
Application programming interfaces (APIs)
IaaS and, 30
immutable, 269
in microservice architecture, 76

Application Response Measurement (ARM), 137
Applications
  adding, 217–218
  code migration in, 278–279
  degradation of, 84
  design of, 175–176
  Gartner Hype Cycle for, 3
  logic of, 217
  management of complex, 255–256
APRA (Australian Prudential Regulatory Authority), 239
Architecture
  allocation in, 72
  Amazon teams and, 75–76
  in BlobStore example, 269–272
  coordination model of, 68–69
  demand management in, 70–71
  further reading on, 78
  mapping in, 20, 71–72
  microservice. *See* Microservice architecture
  of Platformer.com, 149
  requiring changes in, 65–66
  resource management in, 69–71
  structure of, 66–72
  summary of, 77–78
  VM provisioning/deprovisioning in, 70
  work assignments in, 71–72
ARM (Application Response Measurement), 137
ARP (Address Resolution Protocol), 227
Asgard, 289–295
ASGs. *See* Autoscaling groups (ASGs)
Asynchronous coordination, 18
Atlassian
  Bamboo by. *See* Bamboo
  case study of. *See* Atlassian case study
  Cloud by, 263
  JIRA by, 241–242
  Stash by, 241, 249
Atlassian case study
  architecture in, 269–272
  BlobStore example in, 268–273, 279–283
  build and deployment pipeline in, 276–278
  consumer applications in, 278–279
  development process in, 273–279
  further reading on, 284
  "ilities" solutions in, 272–273
  introduction to, 263–264
  microservice deployment in, 265–268
  path to production in, 278–279
  safety/performance in, 269–272
  summary of, 284
  support in, 274–276
Audits for security. *See* Security audits
Australia
  DNS servers in, 31
  intrusion detection in, 133
  Platformer.com in. *See* Platformer.com example
  Sourced Group in. *See* Sourced Group case study
Australian Prudential Regulatory Authority (APRA), 239
Authentication
  hardware for, 166
  introduction to, 165–166
  password controls in, 167–168
  role-based, 166–167
  software for, 166–167
Authorization, 155–156, 168–169
Automation
  of acceptance tests, 95
  of datacenter switches, 230–232
  of error detection/recovery, 304, 308
  introduction to, 11–12
Autoscaled VPCs (virtual private clouds). *See also* Virtual private clouds (VPCs), 260
Autoscaling
  in capacity planning, 51
  groups. *See* Autoscaling groups (ASGs)
  of microservice instances, 266–267
  monitoring and, 57
Autoscaling groups (ASGs)
  CloudFormation and, 257
  CloudTrail and, 292–293
  CloudWatch and, 70, 250
  in deployment, 245
  in operations process, 289
  rolling upgrades and, 294
Availability
  in Atlassian case study, 273
  in CAP, 39
  in CIA, 156–161
Availability zones (AZs), 265
AWS. *See* Amazon Web Services (AWS)

Background of DevOps
  benefits of, 7–11
  the cloud and. *See* Cloud platform
  introduction to, 1
  operations in. *See* Operations (Ops)
  overview of. *See* DevOps overview
Backups, 52–53
Backward/forward compatibility, 107–111
Baking VM images
  AMIs and, 245, 248–249
  defined, 30–31
  lightly *vs.* heavily, 92–93, 112–113
Bamboo
  by Atlassian, 263
  build and deployment pipeline in, 276
  as continuous deployment system, 241–246
  rollbacks in, 254
  "Tasks for AWS" plug-in in, 249
  teardown in, 255
  "update stack if already exists" flag in, 251
Barriers to adoption. *See* Adoption of DevOps
Benefits of DevOps
  in coordination, 7–10
  generally, 7

operations staff and, 10–11
in release process, 7–9
Beta releases, 96
BGP (Border Gateway Protocol), 257
Big flip deployment, 103
Binding time decisions, 20
BizOps, 301
BlobStore example
architecture in, 269–272
build and deployment pipeline in, 276–278
consumer applications in, 278–279
development process in, generally, 273–274
evolving BlobStore in, 279–283
"ilities" solutions in, 272–273
path to production in, 278–279
safety/performance in, 269–272
support in, 274–276
Blue/green deployment, 102–104, 306
BookRenter. *See* Rafter case study
Border Gateway Protocol (BGP), 257
Bottom-up *vs.* top-down monitoring, 145–146
Boundary definition, 170
Branching, 86–88, 276–278
"Breaking the build," 94
Broad network access, 27
BrowserStack, 161
Build process
in Atlassian case study, 265–268
in BlobStore example, 276–278
for deployment pipelines. *See* Deployment pipelines
integration testing environments and, 83
packaging in, 92–93
scripts for, 91–92
in Sourced Group case study, 245, 248–249
status in, 93–94
Business cases. *See also* Business considerations
costs in, 199
external stakeholders in, 201
internal stakeholders in, 200–201
introduction to, 197–198
organizational risks in, 201–202
problem solving issues in, 198–199
risk mitigation in, 201–203
rollout plans in, 203–205
stakeholder impact in, 200–201
success criteria in, 205
technical risks in, 202–203
Business considerations
business cases in. *See* Business cases
compliance, 207–208
continuity. *See* Business continuity
costs, 199
Dev/Ops interactions, 209–210
external stakeholders in, 201
further reading on, 211
in future of DevOps, 301
incident handling, 210
internal stakeholders, 200–201
introduction to, 197
licenses, 209–210
logic in. *See* Business logic
measuring success, 206–207
organizational risks, 201–202
problem solving issues, 198–199
risk mitigation, 201–203
rollout plans, 203–205
stakeholder impact, 200–201
stakeholder satisfaction, 208
success criteria, 205
summary of, 211
technical risks, 202–203
Business continuity
in deployment, 115–117
in DevOps, generally, 59–60
in operations services, 51–53, 59–60
Business logic
adding applications and, 217–218
application logic in, 217
infrastructure discovery in, 219–220
infrastructure in, generally, 217
in Rafter case study, generally, 216

CA Technologies example, 9
Canaries, defined, 15
Canary deployment, 307
Canary testing
in deployment, 117–118
introduction to, 96
monitoring configurations and, 139, 144
in production environments, 193
CAP (Consistency, Availability, Partition Tolerance), 39
Capabilities, 168
Capability maturity models (CMMs), 184
Capacity planning
in DevOps, generally, 59
in monitoring, 131–132
in operations services, 51, 59
Case studies
of continuous deployment pipelines. *See* Sourced Group case study
of migration to microservices. *See* Atlassian case study
overview of, 213
of support for multiple datacenters. *See* Rafter case study
Cases, business. *See* Business cases
CD. *See* Continuous deployment (CD)
CDC (Consumer Driven Contract), 72–73
CDN (content delivery network), 228, 230
CDPs. *See* Continuous deployment pipelines (CDPs)
CF. *See* CloudFormation (CF)
cfn-init, 249
Change management, 171–172
Chaos Monkey, 97
Charging models, 303–304

Chef
  console applications and, 232
  declarative nature of, 194
  in deployment, 121
  infrastructure testing and, 233
  monitoring resources with, 137
  at Rafter, 217–220, 226, 228
  testability and, 192–193
  tool specification in, 12
  traceability in, 85
  uncontrolled switches and, 229–230
CI. *See* Continuous integration (CI)
CIA (confidentiality, integrity, and availability), 156–161
Clients, defined, 67
Closed-loop controls, 57
Cloud platform
  consistency in, 39
  data considerations in, 43–44
  DevOps and, generally, 41–44
  distributed environments in, 34–40
  Domain Name System in, 31–33
  elasticity in, 40
  environments in, 41–43
  failure in, 34–38
  features of, generally, 29
  further reading on, 45–46
  HDFS in, 43–44
  introduction to, 27–29
  IP addresses in, generally, 31
  long tails in, 37–38
  monitoring in, 145–146
  NoSQL databases in, 39–40
  operational considerations in, 44
  Platform as a Service in, 33–34
  summary of, 44–45
  time in, 34–35
  virtualization in, generally, 29–30
  VM creation in, 30, 43
  VM failures in, 36–37
  VM loading in, 30–31
  waste reduction and, 188
CloudFormation (CF)
  in Atlassian case study, 265
  CFEngine in, 184
  in deployment, 121
  in release stage, 251
  in Sourced Group case study, generally, 243–245
  in teardown stage, 254–255
  templates for, 249, 257–258
  tool specification in, 12
CloudWatch
  in Atlassian case study, 265
  autoscaling groups and, 70
  for monitoring, 148
  in Sourced Group case study, 250
Clustrix
  datacenter switches and, 227–228
  switch prerequisites in, 231
  transactional data in, 220–221
  uncontrolled switches and, 229
CMDBs (configuration management databases). *See* Configuration management databases (CMDBs)
CMMs (capability maturity models), 184
CNAMEs, 251–252
Collation, 138
Collection, 137–139
Commit IDs, 85
Compatibility, 111
Compilation of source code, 91
Compliance issues, 207–208
Components, defined, 66
Computational resource protection. *See also* Security, 160–161
Concurrent Versions System (CVS). *See also* Version control, 86–88
Confidentiality, integrity, and availability (CIA), 156–161
Configuration management databases (CMDBs), 186, 195, 308
Configurations
  of alarms, 144
  of alerts, 144
  of deployment pipelines, 89–90
  launch. *See* Launch configurations (LCs)
  of monitoring systems, 139
Conformance checking, 294
Conformity Monkey, 161
Consistency, Availability, Partition Tolerance (CAP), 39
Consistency, in the cloud, 39
Construction phase, 13
Consumer applications, 278–279
Consumer Driven Contract (CDC), 72–73
Consumption-based charging models, 303
Containers
  Atlassian and, 266
  lightweight, 92
  virtual machines *vs.*, 188
Content delivery network (CDN), 228, 230
Context for collation, 138
Continual improvement, 58–59
Continuity. *See* Business continuity
Continuous change, 143–145
Continuous delivery, defined, 80
Continuous deployment (CD)
  Bamboo for, 254–255
  data models and, 192
  definition of, 7, 80
  engineering teams, 238
  feature toggling in, 202
  onboarding groups, 238
  Rafter case study of. *See* Rafter case study
Continuous deployment pipelines (CDPs). *See also* Deployment pipelines
  business considerations and, 200
  in future of DevOps, 306–309

in Sourced Group case study, 240–243
stakeholders and, 200
testing of, 233
Continuous integration (CI)
data models and, 192
defined, 80
in deployment pipelines, 93–94
servers, 91–94
Controlled switch steps, 225–229
Controls, open-loop *vs.* closed-loop, 57
Cookbooks. *See also* Chef, 194
Coordination
in architecture, 68–69
as benefit of DevOps, 7–10, 17
of code pieces, 19
cross-team, 19–20
definition of, 17–18
forms of, 18
in microservice architecture, 72–73
model of, 19, 68–69
of teams, 18–19
Costs, 199
Couchbase, 222–223
CPU utilization, 150–152
Crises, 208
Cronjobs, 218–219, 225–226
Crosscutting issues
in deployment pipelines, 84–86, 125
in environment teardowns, 85
in interoperability, 191–192
in modifiability, 194–195
in monitoring. *See* Monitoring
in negative tests, 84–85
in other “ilities” in, generally, 181–182
in performance, 186–188
in recoverability, 190–191
in regression testing, 84
in reliability, 188–190
in repeatability, 183–186
in small components, 85
in test harnesses, 84
in testability, 192–193
in traceability, 85
Cross-team coordination, 19–20
“Crossing the chasm,” 299–300
Culture of organizations, 20–22
Current state, 216
CVS (Concurrent Versions System). *See also* Version control, 86–88

Daemons, 218, 225–226
Data
decommissioning of, 171–172
distributed, 146–147
immutable, 269
interpretation of, 139–143
migration of, 278–279
outdated, 171
tagging of, 186
transactional, 220–221
volume of, 138
Data models, 19, 192
Data schemas, 111
Data scientists, 301
Databases
in the cloud, generally, 43
compatibility of, 111
configuration management. *See* Configuration management databases (CMDBs)
DynamoDB. *See* DynamoDB
infrastructure support in, 221–222
NoSQL, 39–40, 111
relational, 52, 111
Round-Robin, 138
session data in, 222–223
tiers in, 220–223
transactional data in, 220–221
Datacenters
access to private, 149
automating switches in, 230–232
controlled switching, 225–229
supporting multiple. *See* Rafter case study
switching, 225–233
testing switches in, 232–233
uncontrolled switches in, 229–230
Datamation 2012 IT salary guide, 23
DataNodes, 44
Dean, Jeff, 35
Debian, 92
Decommissioning data, 171–172
Deep-hierarchy systems, 146
Defense in depth, 156–157
Defensive programming, 303
Degraded applications, 84
Demand management, 70–71
Demilitarized Zone, 170
Deming, W. Edwards, 287
Denial-of-service (DoS) attacks, 161, 172–173
Dependability, 72–74
Deployment
A/B testing in, 118
backward/forward compatibility in, 108–111
blue/green, 103–104
business continuity in, 115–117
canary testing in, 117–118
compatibility in, 111
discovery in, 109–110
exploration in, 110
feature toggling in, 107–108
further reading on, 122–123
future of DevOps and, 306–307
introduction to, 101–102
logical consistency in, 105–111
management strategies for, 102–105
monitoring configurations and, 139
to multiple environments, 114–117
multiple service versions active in, 105–111
of operations services, 55–56

Deployment, *continued*
packaging in, 111–114
partial, 117–118
pipelines. *See* Deployment pipelines
portability layers in, 110–111
private clouds in, 116–117
public clouds in, 115–116
rollbacks in, 118–120
rolling upgrades in, 104–105
stage of, 245, 249–250
summary of, 121–122
times in, 267
tools for management of, 121
version control and, 186
Deployment pipelines
architecture and. *See* Architecture
of BlobStore, 276–277
branching in, 86–88
build process and, 91–93
build scripts in, 91–92
build status in, 93–94
building/testing, generally, 79–81
configuration parameters in, 89–90
continuous. *See* Continuous deployment pipelines (CDPs)
continuous integration in, 93–94
crosscutting in. *See* Crosscutting issues
defined, 80–81
deployment via. *See* Deployment
design of, 176–177
development of, generally, 86–91
early release testing and, 97
environment teardown in, 86
environments in, 82–84
error detection in, 97
feature toggles in, 88–89
further reading on, 99
incidents in, 98
integration testing in, 91, 94–95
live testing and, 97
moving systems through, 81–84
negative tests in, 84
overview of, 63
packaging in, 92–93
performance testing of, 95–96
pre-commit testing in, 86–91
production and, 96–97
regression testing in, 84–85
security in, generally, 155–156
small components in, 85
staging in, 95–96
summary of, 98–99
test harnesses in, 84
testing of, generally, 91
traceability in, 81–82, 85
UATs in, 95–96
version control in, 86–88
Design
of deployment pipelines, 176–177
of IT services, 54–55
of operations services, 54–55
of security applications, 175–176
Detection. *See also* Security
of errors. *See* Error detection
of failures, 130
of intrusions, 133–134
of performance degradation, 130–131
security audits for. *See* Security audits
“Developer on Support,” 275–276
Development (Dev)
in BlobStore example, 273–279
build and deployment pipelines in, 276–278
consumer applications in, 278–279
deployment pipelines and, 86–90
path to production in, 278–279
of security, 173–174
support in, 274–276
teams for. *See* Development teams
Development teams
coordination of, 18–19
cross-team coordination and, 19–20
gatekeepers in, 16–17
inter-team coordination of, 72–73
introduction to, 12
reliability engineers in, 15–16
roles in, 14–17
service owners in, 15
size of, 13–14
structure of, 13–16
DevOps overview
Agile in, 12–13
architecture change in, 65–66
automation in, 11–12
background of, 1
barriers to adoption in, 20–23
benefits, 7–11
cloud platform in. *See* Cloud platform
compliance in, 207–208
continuous deployment in, 7
coordination in, 7–10, 17–20
cross-team coordination in, 19–20
culture/type of organizations in, 20–22
definition of, 3–5
department types in, 22
development teams in. *See* Development teams
further reading on, 24–25
future and. *See* Future of DevOps
gatekeepers in, 16–17
IMVU, Inc. and, 7
introduction to, 3–7
monitoring in, 143
operations in. *See* Operations (Ops)
operations services in. *See* Operations services
personnel issues and, 23
perspective of, 11–12
practices in, generally, 5–6, 206–208
release process in, 7–9
reliability engineers in, 15–16
roles in, 14–17
service owners in, 15

silo mentality in, 22–23
size of teams in, 13–14
stakeholder satisfaction in, 208
success in, 206–207
summary of, 23–24
team coordination in, 18–19
team structure in, 13–16
tool support and, 23
tradeoffs in, 24
DevOps-style activities, other groups with, 300–301
Diagnosis
of anomalies, 148–152
of errors in operations, 296
of monitoring results, 142–143
Dilbert cartoons, 22
Direct coordination, defined, 18
Disaster recovery. *See also* Error recovery, 200, 268
Disciplined Agile Delivery, 12–13
Disciplined Agile Delivery: A Practitioner's Approach, 12
Discovery, 109–110
Displaying results, 141
Distributed environments. *See also* Cloud platform
of the cloud, generally, 34–35
concerns in, generally, 45
consistency in, 39
elasticity in, 40
failures in, 34–38
large volumes of data in, 146–147
long tails in, 37–38
NoSQL databases in, 39–40
time in, 34–35
VM failures in, 36–37
DNS (Domain Name System). *See* Domain Name System (DNS)
Docker, 121
Dogfooding, 268
Domain Name System (DNS)
in the cloud, generally, 31–33
datacenter switches and, 227–228
at Rafter, generally, 224
release stage and, 250–252
Domain-specific languages (DSLs), 225, 231
DoS (denial-of-service) attacks, 161, 172–173
Drift between production/non-production, 259
DSLs (Domain-specific languages), 225, 231
Duplication of effort, 20
DynamoDB
baking stage and, 249
BlobStore and, 271
complex applications and, 255–256
persistence and, 256
provider limitations and, 260

Early error detection/repair, 189–190
Early release testing, 97
Edwards, Damon, 143, 207
Elastic Compute Cloud (EC2)
AMIs and, generally, 30
in CDP baking process, 248–249
deploying to multiple environments and, 114–115
ELB and, 265
IAM roles and, 258
key pairs in, 175
Elastic Load Balancing (ELB)
in deploy stage, 249
microservice deployment and, 265–267
in release process, 251
in rolling upgrades, 104–105, 289–291, 295–296
Elasticity, defined, 40
Elasticsearch, 223–224
Ellison, Larry, 27
Empowerment *vs.* control, 302
Encryption, 171, 279
England, Rob, 47
Enterprise Resource Planning (ERP), 21
Environments
in the cloud, generally, 41–43
crosscutting in, 86
definition of, 243–245
deploying to multiple, 114–117
in deployment pipelines, generally, 82–84
distributed. *See* Distributed environments
future of DevOps and, 305–306
integrated development, 163, 184
in microservice architecture, 73
pre-commit, 82–83
teardown in, 86
Ephemeral coordination, 18
Erl, Thomas, 54
ERP (Enterprise Resource Planning), 21
Error detection
automation of, 304, 308
in deployment pipelines, 97
early, 189–190
in future of DevOps, 309
in operations process, 294–295
Error diagnosis, 296
Error recovery. *See also* Disaster recovery
automation of, 304, 308
in future of DevOps, 309
in operations process, 295–296
Eureka, 68–69
Eventual consistency, 39
Exception handling, 190
Existing system adoption, 76–77
Expiration of licenses, 209
Exploration, 110
Extensibility, 273
External stakeholders, 201

Failures
in the cloud, generally, 34–38
detection of, 130
"failing gracefully," 84
instance, 73–74
of virtual machines, 36–38

Fanout systems, 146
Fault trees, 296–297
Feature flags. *See* Flags
Feature toggling
  in deployment, generally, 107–108
  in deployment pipelines, 88–89
  future of DevOps and, 307
  for mixed-version race conditions, 107
  removing from source code, 208
  rollbacks and, 119
  supporting continuous deployment with, 202
Finagle RPC, 269
Financial attacks, 159
First-class stakeholders, 200, 204
Flags
  feature toggling and, 108
  for features, generally, 88
  "update stack if already exists," 251
Floating licenses, 209
Flume, 148
Fu, Min, 287
Future of DevOps
  charging models in, 303–304
  continuous deployment pipelines in, 306–309
  empowerment *vs.* control in, 302
  error reporting/repair in, 309
  final words on, 310
  further reading on, 310
  implementation in, 309
  introduction to, 299–300
  operations as process in. *See* Operations process
  organizational issues in, 300–302
  other groups with DevOps-style activities, 300–301
  overview of, 285
  ownership and reorganizations in, 301–302
  process issues in, 302–305
  standards in, 303
  technology issues in, 305–309
  velocity of changes in, 304–305
  vendor lock-in in, 303

Ganglia, 147
Gartner Hype Cycle for Application Development, 3
Gatekeepers, 16–17
Gates, 204–205
Gem repository servers, 223–224
Gibbons, Barry, 197
Git
  AMIs and, 248
  Atlassian Stash and, 276
  interoperability and, 191
  introduction to, 86–88
GitHub
  in deployment pipelines, 88
  Enterprise version of, 241
  in Rafter case study, 218–219
Go software, 192
Goldman Sachs, 9
Google
  Analytics by, 150
  on build errors, 187, 190
  on datacenter failures, 35
  future of DevOps and, 299
  shared microservices and, 283
  on site reliability engineers, 15
  trunk-based development, 87
Graphs, 141
Graylog 2, 147

Hackett, Buddy, 181
Hadoop Distributed File System (HDFS)
  in the cloud, 43–44
  introduction to, 37
  retention policies and, 139
  RPOs in, 52
Hammant, Paul, 87
Hamming, Richard, 127
Hand, Daniel, 213, 237
Hardware, 59, 166
Hardware provisioning, 48–49, 59
Hawking, Stephen, 3
HDFS. *See* Hadoop Distributed File System (HDFS)
Health Insurance Portability and Accountability Act (HIPAA), 164
Heavily baked deployment, 121
Heavily baked VM images
  in build process, 92–93
  defined, 30–31
  in deployment, 112–113
HIPAA (Health Insurance Portability and Accountability Act), 164
HTTP Status Code 7XX: Developer Errors, 101
HTTPS, 172
Hypervisors, 30, 92

IaaS. *See* Infrastructure as a Service (IaaS)
IBM, 12
Idempotent, defined, 74
Identity and Access Management (IAM), 175, 250, 257–258
Identity management
  authentication for. *See* Authentication
  authorization for, 155–156, 168–169
  generally, 165
  hardware for, 166
  introduction to, 165
  password controls in, 167–168
  roles in, 166–167
  software for, 166–167
  in Sourced Group case study, 258
IDEs (integrated development environments), 163, 184
Ilities
  appropriate levels in, 183–185
  availability. *See* Availability

in BlobStore example, 272–273
data models in, 192
dependability, 72–74
early error detection/repair in, 189–190
extensibility, 273
interfaces in, 191
interoperability, 191–192
maintainability, 273
modifiability. *See* Modifiability
overview of, 181–182
performance in, 186–188
recoverability, 190–191
reliability. *See* Reliability
repeatability, 183–186
resource utilization and, 187–188
scalability, 272
service characteristics in, 189
summary of, 194–195
testability, 192–193
tool interactions in, 195
traceability. *See* Traceability
version control in, 185–186
Images. *See* Amazon Machine Images (AMIs); Virtual machines (VMs)
Immutability, 93, 269
Implementation, 309
Improvement, continual, 58–59
IMVU, Inc., 7
Incentives, 22
Inception phase, 12–13
Incident management
in business considerations, 210
in deployment pipelines, 98
in operations services, 56
stakeholders in, 200, 204
Incinga, 147
Inconsistencies, 106–111
Indirect coordination, defined, 18
Individual hardware, 48–49
Information
at rest, 159
security of, 51–53, 59–60, 159
technology for. *See* IT (information technology)
in transit, 160
in use, 159–160
Information technology. *See* IT (information technology)
Information Technology Infrastructure Library (ITIL)
change models and, 56
continual service improvement and, 58–59
DevOps and, generally, 59
events and, 56
incident management and, 56–57
introduction to, 47–48
service design and, 54–55
service strategies and, 53–54
service transitions and, 55–56, 60–61
Infrastructure
Atlassian and, 267
code as. *See* Infrastructure-as-code
discovering, 219–220
ITIL for. *See* Information Technology Infrastructure Library (ITIL)
in Rafter case study, 217
as services. *See* Infrastructure as a Service (IaaS)
supporting in database tiers, 221–222
testing of, 233
tools for, 223–224
Infrastructure as a Service (IaaS)
defined, 29
IP management in, 30–33
PhotoPNP and, 150
vendor lock-in and, 260–261
virtualization in, 29–30
Infrastructure-as-code
in microservice architecture, 73, 81
security and, 155–156
testability and, 192–193
Instance, defined, 30
Instance failures, 73–74
Integrated development environments (IDEs), 163, 184
Integration
continuous, 91–94, 192
environments for testing, 83
executing, 94–95
scripts, 305
testing, generally, 91–94
Intellectual property attacks, 159
Inter-team coordination, 72–73
Interactions between Dev/Ops, 209–210
Interfaces, 191
Internal stakeholders, 200–201
International Organization for Standardization/ International Electrotechnical Commission (ISO/IEC) 27001, 163
Interoperability, 191–192
Interpretation of data, 139–143
Intrusion detection, 133–134
IP address management
in the cloud, generally, 31
DNS in. *See* Domain Name System (DNS)
persistence in, 33
IPsec/VPN, 257
ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) 27001, 163
Isolation, 170–171
IT (information technology)
day-to-day provisioning of, 50
in operations services, 50, 59
professionals in, generally, 6
salaries in, 23
security and, 162
ITIL. *See* Information Technology Infrastructure Library (ITIL)

Janitor Monkey, 43, 161
Java, 34
Java archives, 92
Java virtual machine (JVM), 280
JavaScript Object Notation (JSON), 217–219
Jenkins tools, 169, 303
JIRA, 274–275
Joomla, 150
JSON (JavaScript Object Notation) files, 217–219
JVM (Java virtual machine), 280

Kafka, 148
Kerberos, 167, 281
Key performance indicators (KPIs), 202
Keys, 160–161, 269–271
Kibana, 266–267
Knight Capital, 9, 89
KPIs (key performance indicators), 202

Lamport, Leslie, 66
Latency
  business continuity and, 116
  monitoring and, 130–131
  of user requests, 132
Latency Monkey, 97
Launch configurations (LCs), 249, 289, 292–295
Lead times, 208
Levels, 183–185
Library usage, 82
Licenses, 209–210
Life cycles
  of applications, 245–248
  introduction to, 6
  of services, 48
Lightly baked VM images
  in build process, 92–93
  defined, 30–31
  in deployment, 112, 121
Lightweight containers, 92
Likely changes, 75
Lincoln, Abraham, 299
Live testing, 97
Load balancing
  distributed environments and, 40
  DNS and, 32–33
  ELB for. *See* Elastic Load Balancing (ELB)
Local mirrors, 189
Logic, business. *See* Business logic
Logical consistency
  backward/forward compatibility in, 108–111
  compatibility of data in, 111
  in deployment, generally, 105–106
  discovery in, 109–110
  exploration in, 110
  feature toggling in, 107–108
  multiple service versions in, 105–111
  portability layers in, 110–111
Logs
  in microservice deployment, 267
  of monitoring results, 140
  operations process and, 289–295
Logstash, 147, 291
Long tails, 37–38
Long-term capacity planning, 132
LoudCloud, 184

Maintainability, 273
Man-in-the-middle attacks, 166
Mapping, 71–72, 269–271
Market-based charging models, 303
Masters
  in controlled switches, 226–227
  in database replication, 221–222
  in uncontrolled switches, 229
Maturity model, 203
Maven, 82
Measured service, 27–28
Memcached, 36–37, 222–223
Metrics, 133
Microservice architecture. *See also* Architecture
  definition of, 67–68
  dependability of, 72–74
  environments in, 73
  existing system adoption in, 76–77
  instance failures in, 73–74
  inter-team coordination in, 72–73
  introduction to, 67–68
  likely changes in, 75
  modifiability of, 74–75
  monitoring of, 146
  quality of, generally, 72
  ripple effect reduction in, 75
  security in, generally, 155
Microservice deployment, 101–102
Migration
  of code, 278–279
  of data, 278
  future of DevOps and, 303
  to microservices. *See* Atlassian case study
Mining techniques, 291
Mixed-version race condition, 106–107
Modifiability
  as ility, generally, 194
  of microservice architecture, 74–75
  single tools for, 194–195
  tool interactions in, 195
Modules, defined, 66
Monitoring
  agent-based/agentless, 136–137
  alarms and. *See* Alarms
  alerts and, 139–144
  bottom-up *vs.* top-down, 145–146
  capacity planning in, 131–132
  challenges in, 143–147
  in the cloud, 145–146
  collection and, 137–139

under continuous changes, 143–145
defined, 128
for demand management, 71
of DevOps processes, 143
diagnosis from results of, 142–143
diagnosis of anomalies from, 148–152
displaying results of, 141
elasticity and, 40
failure detection in, 130
further reading on, 153
future of DevOps and, 307
graphs of results of, 141
interpretation of data from, 139–143
introduction to, 127–129
intrusion detection in, 133–134
latency in, 130–131
logs of results of, 140
of microservice architecture, 146
of operations, 137, 296–298
of performance degradation, 130–131
Platformer.com example of. *See* Platformer .com example
procedures for, 134–139
reaction to results of, 142–143
as service operation, 57–58
solutions for various platforms, 151
storage and, 137–139
subjects of, 129–134
summary of, 152
throughput in, 131
times to change configuration of, 139
tools for, 147–148
user interactions in, 132–133
utilization in, 131
Moore, Geoffrey, 299
Motivation, 288–289
Mozilla, 17
Multiple environments, 114–117
Multiple versions of services, 105–111

Nagios, 147
NameNodes, 44
National Institute of Standards and Technology (NIST)
"800-53" by, 162–168, 171–173
"AC-3, Access Enforcement" by, 168
on charging models, 132
on the cloud, 27–29
on development, 173
on elasticity, 40
on encryption, 171
on PaaS, 34
on security, 162–165, 167
Negative tests, 84–85
Netflix
Archaius tool of, 108
Asgard, 121
Conformity and Security Monkeys, 176
Edda, 186
error detection at, 97
Eureka, 68–69
future of DevOps and, 299, 301
gatekeepers at, 16
shared microservices and, 283
Simian Army, 97, 167, 309
Nexus, 276
NIST. *See* National Institute of Standards and Technology (NIST)
No-downtime paths, 278–279
Nonfunctional tests, 96
NoSQL databases
in the cloud, 39–40
data schemas in, 111
DynamoDB. *See* DynamoDB

Offline activities, 289–293
On-demand self-service, 27
Online activities, 294–296
Open-loop controls, 57
OpenVZ, 263
Operating system packages, 92
Operations (Ops)
in the cloud, generally, 44
monitoring of, 137
personnel for, 6, 10–11
as process. *See* Operations process
responsibilities of staff for, 10–11
services and. *See* Operations services
Operations process
error detection in, 294–295
error diagnosis in, 296
error recovery in, 295–296
further reading on, 298
introduction to, 287–288
monitoring in, 296–298
motivation and, 288–289
offline activities in, 289–293
online activities in, 294–296
overview of, 288–289
summary of, 298
Operations services
business continuity in, 51–53, 59–60
capacity planning in, 51, 59
continual improvement of, 58–59
deployment of, 55–56
design of, 54–55
DevOps and, generally, 59–61
hardware provisioning, 48–49, 59
IT functions in, 50, 59
overview of, 47–48
security in, 51–53, 59–60
service level agreements for, 50–51, 59
service operations in, 56–58
software provisioning, 49–50, 59
strategy planning for, 53–54
transitions to, 55–56
Operators, defined, 6
Ops (operations). *See* Operations (Ops)

Opsware, 184
Organization-wide hardware, 49
Organizational issues
  in business cases, 201–202
  culture/type of organizations, 20–22
  empowerment *vs.* control, 302
  in future of DevOps, 300–302
  other groups with DevOps-style activities, 300–301
  ownership, 301–302
  regulations, 21
  in Sourced Group case study, 238–240
OSGi, 282
Outages, 208
Outdated data, 171
"Outside of channels" controls, 164
Ownership, 172, 301–302
Oxford English Dictionary, 17

PaaS. *See* Platform as a Service (PaaS)
Packaging
  in deployment, generally, 111–114
  in deployment pipelines, 92–93
  of files, 91
PagerDuty, 274
Painter, John, 213, 237
Parnas, David, 71
Partial deployment, 117–118
Passwords, 160–161, 167–168
Patching, 171–172
Paxos, 107
Payment Card Industry (PCI), 164, 239
Performance
  Apdex on, 148
  of deployment pipelines, 95–96
  detecting degradation of, 130–131
  DNS and, 32
  as ility, generally, 186
  measurement of, 187
  resource utilization and, 187–188
  of security audits, 174–175
  testing, 83
Persistence
  coordination, 18
  of IP addresses, 33
  in Sourced Group case study, 256–257
  in virtual machines, 36–37
Personnel issues, 23
Perspective of DevOps, 11–12
PhotoPNP, 150–152
Physical distance, 201–202
Physical hardware, 48–49
Pipeline state, 255–256
Platform as a Service (PaaS)
  Atlassian and. *See* Atlassian case study
  in the cloud, generally, 33–34
  defined, 29
  Platformer.com as. *See* Platformer.com example
  vendor lock-in and, 261
Platform providers, 162, 172
Platform, the cloud as. *See* Cloud platform
Platformer.com example
  anomaly detection in, 150
  conclusions about, 150–152
  context of, 148–150
  data collection in, 150
  monitoring in, generally, 148
Portability layers, 110–111
Pre-commit environments, 82–83
Pre-commit testing, 86–91
Private clouds, 116–117, 149
Private datacenters, 149
Problem-solving issues, 198–199
Process issues
  charging models, 303–304
  in future of DevOps, generally, 302
  standards, 303
  velocity of changes, 304–305
  vendor lock-in, 303
Process models. *See* Operations process
Production
  canary testing and, 193
  deployment pipelines and, 96–97
  environments, 41–43, 83
  non-production environments and, 259
  path to, 278–279
Products *vs.* processes, 182
Programming for safety/performance, 269–272
Project hardware, 49
ProM, 291
Provider limitations, 260
Provisioning
  of hardware, 48–49, 59
  of IT, 50
  of software, 49–50, 59
  of virtual machines, 70
Public clouds, 115–116
Puppet, 121, 194
Push/pull commands, 86–87

Quality assurance (QA), 169, 273
Quality issues. *See also* Ilities, 72
Questionnaires, 208

Race conditions, 112
Rafter case study
  adding applications in, 217–218
  application logic in, 217
  business logic in, 216–220
  continuous deployment pipeline in, 233
  controlled switch steps in, 225–229
  current state of, 216
  database tiers in, 220–223
  datacenter switches in, 225–232
  defining/automating switch steps in, 230–232
  DNS in, 224
  Elasticsearch in, 224
  further reading on, 234–235
  gem repository servers in, 223–224

infrastructure discovery in, 219–220
infrastructure in, generally, 217
infrastructure support in, 221–222
infrastructure testing in, 233
infrastructure tools in, 223–224
introduction to, 215–216
session data in, 222–223
summary of, 233–234
testing in, 232–233
transactional data in, 220–221
uncontrolled switches in, 229–230
web tiers in, 216–220
Rapid elasticity, 27–28
RBA (role-based authentication), 167
RBAC (role-based access control), 168–169
RDBMSs (relational database management systems), 52, 111
Reaction to results, 142–143
Real user monitoring (RUM), 133
Recoverability, 190–191
Recovery
from disasters, 200, 268
from errors, 304, 308
Recovery point objectives (RPOs), 51–53
Recovery time objectives (RTOs), 51–53
Red/black deployment, 103, 306
Red Hat, 92, 161
Redis
datacenter switches and, 226–227
supporting database tiers, 221–222
uncontrolled switches and, 229
Regression testing, 84–85
Regulated organizations, 21
Relational database management systems (RDBMSs), 52, 111
Release packages, 60–61
Release process
beta releases, 96
early release testing, 97
overview of, 7–9
planning steps for, 8
release stage in, 245–247, 250–254
Reliability
DNS and, 32
early error detection/repair in, 189–190
of services, 189
user interactions and, 132
Reliability engineers, 15–16, 275
Repeatability
at appropriate levels, 183–185
as ility, generally, 183
version control in, 185–186
Replication, 307
Repudiation, 173
Resource management
access control in, 168
in architecture, 69–71
in Atlassian case study, 266–267
in ilities, 187–188
introduction to, 20
pooling resources in, 27–28
security in, 159–161
Retention policies, 138
Revision control. *See* Version control
Ripple effect reduction, 75
Risk mitigation, 201–203
Role-based access control (RBAC), 168–169
Role-based authentication (RBA), 167
Roles, 14–17, 162–164
Rollbacks
definition of, 8
in deployment, 118–120
future of DevOps and, 307
in Sourced Group case study, 254
Rolling upgrades
in deployment, 102–107
future of DevOps and, 307
monitoring configurations and, 139
operations and, 289–295
Rollout plans, 203–205
Root passwords, 169
Round-Robin Database (RRD), 138
Route 53, 250–254, 265
RPOs (recovery point objectives), 51–53
RRD (Round-Robin Database), 138
RSpec, 232–233
RTOs (recovery time objectives), 51–53
Ruby
Gems, 217, 228, 309
on Rails, 216–218
switch steps in, 230
RUM (real user monitoring), 133
Runtime-specific packages, 92
Rutan, Burt, 79

S3. *See* Simple Storage Service (S3)
S4 stream processing tool, 148
SaaS (Software as a Service), 29, 261
Sabotage, 159
Salary guides, 23
SAP Business Suite, 90
Sarbanes-Oxley, 301
Scalability, 40, 272
Schemas, 111
Scout, 225, 229
Scripts
build, 91–92
defined, 81
integration, 305
version control and, 185
SDNs (software-defined networks), 162
SecOps (security operations), 239–240
Secure Shell (SSH), 136, 258
Security. *See also specific attack types*
access control for. *See* Access control
activities in, 162–164
application design and, 175–176
architects, 162, 172
auditing for. *See* Security audits
authentication for. *See* Authentication

Security, *continued*
  authorization for, 155–156, 168–169
  BlobStore and, 273
  boundary definition for, 170
  definition of, 156–159
  denial-of-service attacks and, 161, 172–173
  deployment pipeline design and, 176–177
  detection for, 172–173
  development of, 173–174
  encryption for, 171
  further reading on, 178–179
  future of DevOps and, 301, 306
  hardware for, 166
  identity management for, 165–169
  introduction to, 155–156
  isolation for, 170–171
  in operations services, 51–53, 59–60
  outdated data in, 171
  ownership/responsibility for, 172
  passwords for, 167–168
  personnel for, 162–164
  resource protection in, 159–161, 168
  role-based access control in, 168–169
  role-based authentication for, 167
  software for, 166–167
  in Sourced Group case study, 257–258
  summary of, 177–178
  threats to. *See* Threats to security
  version updating in, 171–172
  vulnerability patches in, 171–172
Security audits
  audit trails *vs.* logs, 173
  introduction to, 155–156, 164
  overview of, 172–173
  performance of, 174–175
Security Monkey, 161, 167
Security operations (SecOps), 239–240
SEI (Software Engineering Institute), 158–159
Sensu, 147
Serverspec, 193
Service level agreements (SLAs)
  Amazon and, 76
  for demand management, 71
  future of DevOps and, 307–308
  for operations services, 47, 50–51, 59
Services
  agreements at level of. *See* Service level agreements (SLAs)
  AWS. *See* Amazon Web Services (AWS)
  characteristics of, 189
  defined, 67
  deployment of. *See* Deployment
  desk operations of, 50
  life cycles of, 48
  operations and. *See* Operations services
  owners of, 15
  S3. *See* Simple Storage Service (S3)
  SNS, 265
  software as, 29, 261
Session data, 222–223
SHA-1 hash, 269–271
Shard allocation, 183
Shek, Sidney, 213, 261
Shellshock, 161
Short-term capacity planning, 132
Silo mentality, 22–23
Simian Army, 97, 167, 309
Simple Queue Service (SQS), 258
Simple Network Management Protocol (SNMP), 136
Simple Notification Service (SNS), 265
Simple Storage Service (S3)
  in Atlassian case study, 279
  in Sourced Group case study, 245, 250, 254–258
Single sign-ons, 166–167
Site Reliability Engineers, 15
Size of teams, 13–14
SLAs. *See* Service level agreements (SLAs)
Slaves
  in controlled switches, 226–227
  in database replication, 221–222
  in uncontrolled switches, 229
Slow-moving organizations, 21
Small components, 85
Smoke tests, 91, 95, 278
Snapshots, 259
SNMP (Simple Network Management Protocol), 136
SNS (Simple Notification Service), 265
Software
  audits, 210
  for authentication, 166–167
  provisioning, 49–50, 59
  third-party, 75
Software as a Service (SaaS), 29, 261
Software-defined networks (SDNs), 162
Software Engineering Institute (SEI), 158–159
Solution architects, 162, 172
Source code revision systems, 241
Source control. *See* Version control
Sourced Group case study
  access management in, 258
  advanced concepts in, generally, 259
  AWS native services in, 261
  build and test stage in, 248–249
  CloudFormation in, 243–245, 257–258
  complex applications in, 255–256
  continuous deployment pipeline in, 240–243, 258–260
  deploy stage in, 249–250
  drift between production/non-production in, 259
  environment definition in, 243–245
  further reading on, 262
  identity management in, 258
  introduction to, 237–238
  organizational context in, 238–240

persistence in, 256–257
provider limitations in, 260
release stage in, 250–254
rollbacks in, 254
security in, 257–258
standardized application life cycle in, 245–248
summary of, 261
teardown stage in, 254–255
traffic matching in, 253
vendor lock-in in, 260–261
Spacewalk, 161
Spencer, Rob, 60
Splunk, 147
Spoofing, tampering, repudiation, information disclosure, denial-of-service, elevation of privilege (STRIDE), 157–158, 173
Sprint methodology, 247
SQS (Simple Queue Service), 258
SSH (Secure Shell), 136, 258
Staging
environment of, 83
introduction to, 80–81
testing in, 95–96
Stakeholders
external, 201
first-class, 200, 204
impact on, 200–201
internal, 200–201
satisfaction of, 208
Standardized application life cycles, 245–248
Standards, 303
State management, 115–116, 202
Storage, 137–139
Storm, 148
Strategic planning, 53–54, 301
Stream processing tools, 148
STRIDE (spoofing, tampering, repudiation, information disclosure, denial-of-service, elevation of privilege), 157–158, 173
Structure of architecture. *See also* Architecture, 66–72
Subnets, 170
Subscription-based charging models, 303
Subversion (SVN). *See also* Version control, 86–88
Success criteria, 205
Support
in Atlassian case study, 274–276
of continuous deployment, 202
of infrastructure, 221–222
of tools, 23
SVN (Subversion). *See also* Version control, 86–88
Switches
controlled, 225–229
defining/automating, 230–232
in deployment pipelines, generally, 88
uncontrolled, 229–230
Synchronization, 18, 107
Synthetic monitoring, 133
"System and Communications Protection," 167
System-managed credentials, 166–167

Tagging data items, 186
Tampering, 169–170
TeamCity, 233
Teams. *See* Development teams
Teardown stage
future of DevOps and, 307
in Sourced Group case study, 245–246, 254–255
Technical controls, 164
Technical risks, 202–203
Technology experts, 50
Technology issues
continuous deployment pipeline concepts in, 306–308
continuous deployment pipeline quality in, 308–309
error reporting/repair in, 309
in future of DevOps, generally, 305–306
implementation in, 309
Test-driven development, 91
Test Kitchen, 121, 193
Testability, 192–193
Testing
of continuous deployment pipelines, 233
of datacenter switch applications, 232–233
deployment pipelines and. *See* Deployment pipelines
DNS and, 32
environments for, 41–43
harnesses in, 84
of infrastructures, 233
integration, 91
in Rafter case study, generally, 232–233
Third parties, 75, 149
Threats to security. *See also* Security
denial-of-service attacks, 161, 172–173
financial attacks, 159
intellectual property attacks, 159
man-in-the-middle attacks, 166
overview of, 156–159
Throughput, 131
Ticketing systems, 241–242
Time
in the cloud, 34–35
collating related items by, 138
between commit/deployment, 206–207
between error discovery/repair, 207
in operations process, 289
prediction of, 308–309
Time to live (TTL), 31–32, 224
Toggling. *See* Feature toggling
Tools. *See also specific tools*
for continuous deployment pipelines, 241
for deployment management, generally, 121
for infrastructure, 223–224
interactions of, 195

Tools, *continued*
- for monitoring, 147–148
- specifications for, 12
- for stream processing, 148
- support for, 23

Top-down monitoring, 145–146
Traceability
- crosscutting and, 85
- in deployment pipelines, 81–82
- in future of DevOps, 305

Traffic matching, 253
Transactional data, 220–221
Transitions, 13, 55–56
Trunk-based development, 87–88
TTL (time to live), 31–32, 224
Twain, Mark, 127
Twitter, 269
Two-pizza rule, 14, 75
Type of departments/organizations, 20–22

Uncontrolled switches, 229–230
Unit tests, 91
URLs, 31–33
User acceptance tests (UATs)
- in deployment pipelines, 80–81, 95–96
- environment of, 83
- in Sourced Group case study, 246–248

User interactions, 132–133
Utilization, 131

Vagrant, 121
Varnish, 271
Velocity of changes, 304–305
Vendor lock-in, 260–261, 303
Verizon, 158–159
Version control
- in deployment pipelines, 86–88
- in ilities, 185–186
- for mixed-version race conditions, 107
- in security, 171–172

Virtual hardware, 49
Virtual machines (VMs)
- AMIs and. *See* Amazon Machine Images (AMIs)
- application state in, 36
- baking images in. *See* Baking VM images
- charging models and, 304
- client state in, 36
- consistency in, 39
- creation of, 30, 43
- deployment of. *See* Deployment pipelines
- DNS in, 31–33
- elasticity in, 40
- failure of, 36–38
- image sprawl in, 43
- images in, 30–31, 92–93, 161
- introduction to, 29–30
- IP address management and, 31–33
- in Java, 280
- launching, 207
- loading, 30–31
- long tails and, 37–38
- NoSQL databases in, 39–40
- in operations process, 289
- persistent state in, 33, 36–37
- provisioning/deprovisioning, 70
- sprawl in, 161
- stateless, 36

Virtual private clouds (VPCs)
- CloudFormation and, 257
- number of security groups per, 260
- as virtual private networks, 244

Virtual private networks (VPNs), 244, 264
Virtualization, defined. *See also* Virtual machines (VMs), 29–30
VM instance, defined, 30
VMs. *See* Virtual machines (VMs)
VMware, 114, 184, 216
VPCs. *See* Virtual private clouds (VPCs)
VPNs (virtual private networks), 244, 264
Vulnerability patches, 171–172

Web tiers
- adding applications in, 217–218
- application logic in, 217
- infrastructure discovery in, 219–220
- infrastructure in, generally, 217
- in Rafter case study, generally, 216

Wikipedia
- on DevOps, 4–5, 18
- on environments, 83–84
- on operations staff responsibilities, 10–11
- on regression testing, 84
- on release planning steps, 8

Williams, Chris, 213, 215
Windows Management Instrumentation (WMI), 136
"Within channels" controls, 164
WMI (Windows Management Instrumentation), 136
Work assignments, 71–72
Workload/application behavior changes, 304–305
World Wide Web, 31
Wrappers, 189
www.atlassian.com/company, 263
www.rafter.com/about-rafter, 215
www.sourcedgroup.com.au, 237

XebiaLabs example, 9
Xu, Xiwei, 287

ZAB algorithms, 107
ZooKeeper
- distributed environments and, 36
- feature toggling in, 107–108
- state management and, 202
- VM provisioning/deprovisioning in, 70

Big Data: Architectures and Technologies

About the Course

Scalable "big data" systems are significant long-term investments that must scale to handle ever-increasing data volumes, and therefore represent high-risk applications in which the software and data architecture are fundamental components of ensuring success. This course is designed for architects and technical stakeholders such as product managers, development managers, and systems engineers who are involved in the development of big-data applications. It focuses on the relationships among application software, data models, and deployment architectures, and how specific technology selection relates to all of these. While the course touches briefly on data analytics, it focuses on distributed data storage and access infrastructure, and the architecture tradeoffs needed to achieve scalability, consistency, availability, and performance. We illustrate these architecture principles with examples from selected NoSQL product implementations.

Who Should Attend?

- Architects
- Technical stakeholders involved in the development of big data applications
- Product managers, development managers, and systems engineers

Topics

- The major elements of big data software architectures
- The different types and major features of NoSQL databases
- Patterns for designing data models that support high performance and scalability
- Distributed data processing frameworks

Three Ways to Attend

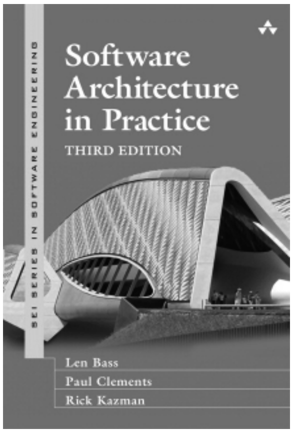
- Public instructor-led offering at an SEI office
- Private, instructor-led training at customer sites
- eLearning

For More Information

To learn more and to register for the course, visit www.sei.cmu.edu/go/big-data

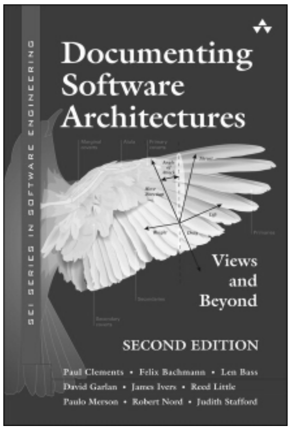
Addison Wesley

More Software Architecture Guidance from Len Bass


Software Architecture in Practice
THIRD EDITION
SEI SERIES IN SOFTWARE ENGINEERING
Len Bass
Paul Clements
Rick Kazman

ISBN-13: 978-0-321-81573-6

In a real-world setting, *Software Architecture in Practice, Third Edition* introduces the concepts and best practices of software architecture—how a software system is structured and how that system's elements are meant to interact. This guide is structured around the concept of architecture influence cycles and each cycle shows how architecture influences, and is influenced by, a particular context in which architecture plays a critical role


Documenting Software Architectures
Views and Beyond
SECOND EDITION
SEI SERIES IN SOFTWARE ENGINEERING
Paul Clements • Felix Bachmann • Len Bass
David Garlan • James Ivers • Reed Little
Paulo Merson • Robert Nord • Judith Stafford

ISBN-13: 978-0-321-55268-6

Documenting Software Architectures, Second Edition, provides the most complete and current guidance, independent of language or notation, on how to capture an architecture in a commonly understandable form. Drawing on their extensive experience, the authors first help you decide what information to document, and then, with guidelines and examples (in various notations, including UML), show you how to express an architecture so that others can successfully build, use, and maintain a system from it.

informIT
Safari Books Online

To see complete coverage and check out sample content visit informit.com.

eBook and print formats available.