Preface

Three recent trends are dramatically extending the responsibilities of software engineers and, hence, the requisite skill set. The widespread adoption of the cloud has fundamentally expanded the skill set required by software engineers. They need to be familiar with virtualization both of hardware and operating system, with how networks work, and with managing failure in the cloud, scaling, and load balancing.

Driven by the adoption of the cloud, release cycles have shortened dramatically. It is no longer sufficient to release once a quarter. Monthly, weekly, daily, and hourly releases are common with the release time dependent more on business considerations than on technical ones. Short release cycles are enabled by the migration to microservices, by the proliferation of tools for a deployment pipeline, and by the growth of container orchestration tools. Yet rapid release cycles also depend on architectural decisions and disciplined use of the available tools.

Today, a software engineer is expected to be responsible for their portion of the system from “cradle to grave”. This, together with the architecture of the cloud, requires an engineer to understand operations concepts such as business continuity and incident handling. They also must understand logging, monitoring, and alerts.

A software engineer, typically, learns this material on the job requiring much work going through blogs, tutorials, and system documentation. This is a difficult and time consuming process. Some organizations have training courses for new hires where an organizations’ specific processes and tools are covered. Usually, however, these courses do not provide a broad context for the processes and tools.

This book is aimed at software engineers (and prospective software engineers) who are juniors/seniors or first year graduate students in Computer Science or Software Engineering. It provides the context within which modern engineers work and the skills necessary to operate within that context. Each chapter discusses the theory associated with its topics and then has hands on exercises to apply the theory. There are also discussion questions that can be done in a classroom setting or in discussion groups to enable the students to gain a better understanding of the theory and its implications. This combination of exercises and discussions supports either traditional lecture style teaching or the “flipped classroom” style. The modular nature of the presentation of the material makes it easy to skip sections that your students may already be familiar with.

Our goal in this book, admittedly ambitious, is to distill the knowledge required of modern software engineers beyond programming languages and a three tiered architecture. We cover six major topics: Virtualization and the cloud; networks and distribution, microservice architecture, deployment, security, and operations. We make no claim that the reader of this book will emerge as an expert in any of these areas but they will have a basic understanding of the topics and some hands on exercises to cement this understanding. This provides the basis for an engineer to more quickly come up to speed and become productive in an organizational setting.

Since the beginning of computer science education there has been a debate about the extent to which a student should understand what is below a given level of abstraction. Higher level languages mask machine language – how much does the student need to know about machine language? Optimizing compilers mask certain reorganizations of computation – how much does the student need to know about these reorganizations? Java masks memory allocation – how much does the student need to know about memory allocation? And so on. In almost every case, the answer has been some, but not too much.

This is the view we are taking in this book. The software engineer needs to know something about virtualization, networking, the cloud, and the other topics we cover but they do not need to know the gritty details. Computer Science and Software Engineering curricula have courses in distributed networks (the cloud), security, and software architecture. These courses go into more detail than the typical software engineer needs. This leaves a gap that we are attempting to fill with this book. Enough knowledge of these topics for the software engineer but not so much knowledge that an engineer is overwhelmed.