Community Experience Distilled

Learning Akka

Build fault-tolerant, concurrent, and distributed applications with Akka

Jason Goodwin

[PACKT] PUBLISHING open source* community experience distilled

Learning Akka

Build fault-tolerant, concurrent, and distributed applications with Akka

Jason Goodwin

BIRMINGHAM - MUMBAI

Learning Akka

First published: December 2015

Production reference: 1181215

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78439-300-7

www.packtpub.com

Credits

Author
Jason Goodwin

Reviewer
Taylor Jones

Commissioning Editor
Akram Hussain

Acquisition Editor
Nikhil Karkal

Content Development Editor
Dharmesh Parmar

Technical Editor
Gebin George

Copy Editor
Yesha Gangani

Project Coordinator
Nikhil Nair

Proofreader
Safis Editing

Indexer
Tejal Daruwale Soni

Graphics
Jason Monteiro

Production Coordinator
Melwyn Dsa

Cover Work
Melwyn Dsa

About the Author

Jason Goodwin is a developer who is primarily self-taught. His entrepreneurial spirit led him to study business at school, but he started programming when he was 15 and always had a high level of interest in technology. This interest led his career to take a few major changes away from the business side and back into software development. His journey has led him to working on high-scale distributed systems. He likes to create electronic music in his free time.

He was first introduced to an Akka project at a Scala/Akka shop—mDialog—that built video ad insertion software for major publishers. The company was acquired by Google eventually. He has also been an influential technologist in introducing Akka to a major Canadian telco to help them serve their customers with more resilient and responsive software. He has experience of teaching Akka and functional and concurrent programming concepts to small teams there. He is currently working via Adecco at Google.

Acknowledgments

I wish to write a thank-you note here to a few people who have shaped and formed my opinions, and supported me through my own journey.

First, to my wife Kate, thank you for the many, many months of support while I wrote this book and worked on crazy projects. Without your constant support, patience, and care, changing my career to do things that I love to do, and writing this, would not be possible. We made it to the finish line. Time for painting, fixing the house, and Netflix and chill!

To my parents and grandparents, who always told me that I can do anything that I set my mind to: thank you for your advice. You were right.

To my mDialog/Google team, thanks for your reviews and discipline on my journey—I feel lucky to have had the opportunity to work with you all. To Chris especially, thanks for your faith that my interest would be enough to help me grow into a decent engineer, and for always expecting that the team keep it clean.

To Craig and Herb, thanks for the early start. If I wasn't doing bubble sorts, drawing pixelated circles, or converting customer databases when I was 17, I'm not sure I would have been able to find my way to the work that I love to do so much today.

About the Reviewer

Taylor Jones is a full-stack software engineer specializing in Java-based webapp development currently working at Cisco Systems. He enjoys designing and building complex applications with open source technologies and playing with his dog, and is semi-competent at DotA 2.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

This book attempts to give both the introductory reader and the intermediate or advanced reader an understanding of basic distributed computing concepts as well as demonstrates how to use Akka to build fault-tolerant horizontally-scalable distributed applications that communicate over a network. Akka is a powerful toolkit that gives us options to abstract away whether a unit of work is handled on the local machine or a remote machine on the network. Throughout this book, concepts will be introduced to help the reader understand the difficulty of getting systems to talk to each other over the network while introducing the solutions that Akka offers to various problems.

Soaked in these pages is my own journey, my own discovery—and I hope that you will share that with me. I have a fair amount of professional Akka experience working with both Java8 and Scala, but I have learned a lot of the finer details of Akka while writing this book. I feel that this work is a good introduction to how and why to use Akka, and demonstrates how to start building scalable and distributed applications with the Akka toolkit. It does not simply repeat the documentation, but covers many of the important topics and approaches you should understand to successfully approach building systems to handle the scale-related problems we encounter as developers today.

What this book covers

Chapter 1, *Starting Life as an Actor*: Introduction to the Akka Toolkit and Actor Model.

Chapter 2, *Actors and Concurrency*: Reactive. Working with Actors and Futures.

Chapter 3, *Getting the Message Across*: Message Passing Patterns.

Chapter 4, *Actor Lifecycle – Handling State and Failure*: Actor Lifecycle, Supervision, Stash/ Unstash, Become/ Unbecome, and FMSs.

Chapter 5, *Scaling Up*: Doing work concurrently, Router Groups/Pools, Dispatchers, Handling Blocking I/O, and APIs.

Chapter 6, *Successfully Scaling Out – Clustering*: Clustering, CAP Theorem, and Akka Cluster.

Chapter 7, *Handling Mailbox Problems*: Overwhelmed mailboxes, choosing different mailboxes, Circuit Breakers.

Chapter 8, *Testing and Design*: Specification, Domain Driven Design, and Akka Testkit.

Chapter 9, *A Journey's End*: Other Akka Features. Next steps.

What you need for this book

You will need a PC with access to install tools such as Java JDK8 (for Java development) or Java JDK6 (for Scala development). You will also require sbt (Simple Build Tool) or Typesafe Activator, which contains sbt. Installation is covered in this book.

Who this book is for

This book is intended for beginner to intermediate Java or Scala developers who want to build applications to serve the high-scale user demands in computing today. If you need your applications to handle the ever-growing user bases and datasets with high performance demands, then this book is for you. *Learning Akka* will let you do more for your users with less code and less complexity by building and scaling your networked applications with ease.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
// The executor function used by our promise.
// The first argument is the resolver function,
// which is called in 1 second to resolve the promise.
function executor(resolve) {
    setTimeout(resolve, 1000);
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Select the folder and click **Next**."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/LearningAkka_ColoredImages.pdf`

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

eBooks, discount offers, and more

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `customercare@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Starting Life as an Actor

This book is primarily intended for intermediate, to senior-level developers wishing to explore Akka and build fault-tolerant, distributed systems in Scala or modern versions of Java.

This book has been written for the engineer who is faced with building applications that are fast, stable, and elastic, meaning they can scale to meet thousands or tens of thousands concurrent users. With more users having access to the Internet with faster devices and networks, today, more than ever, we need our applications to be able to handle many concurrent users working with larger datasets and with higher expectations of application stability and performance.

This book does not assume that you have a deep understanding of concurrency concepts and does try to introduce all of the concepts needed to know how to start a project from scratch, work with concurrency abstractions, and test and build standalone or networked applications using Akka. While this book should give you everything you need in those regards, it's not meant for an absolute beginner and does assume some does assume some programming proficiency.

Here is a quick overview of what you'll need and what you'll get out of this book.

- Requirements:
    - Intermediate Scala or Java experience
    - A computer
    - Internet connectivity
- Recommendations (but you can learn as you go):
    - If using Java, Java8 lambda exposure
    - Git and GitHub experience for assignments

- What you'll learn:
    - Learn to build distributed and concurrent application
    - Learn techniques for building fault-tolerant systems
    - Learn techniques for sharing code between projects and teams
    - Learn several concepts and patterns to aid in distributed system design

What's in this book?

To meet the modern challenges a platform developer may face, this book puts a strong focus not only on Akka but also on distributed and concurrent computing concepts. It is my intention to give you a toolkit to understand the problems you'll face while trying to scale these distributed and concurrent applications.

These pages are not a re-iteration of the Akka documentation. If you want a desk reference or manual, the 460-page Akka documentation will serve that purpose well. This book is not simply a book about Akka, it is a book about building concurrent and distributed systems with Akka.

This book will take you on a journey to show you a new way of working with distributed and concurrent applications. This book will arm you with an understanding of the tools, and then will show you how to use them. It will demonstrate how to build clusters of applications that talk to each other over the network and can have new computing nodes added or removed to be able to scale to meet the needs of your users. We'll learn how to do things like building pools of workers to handle huge jobs at scale to show how it's done. We will talk about important theorems and common approaches in distributed systems and show how they affect our design decisions, and we will discuss problems you will encounter related to network reliability and demonstrate how we can build our applications to be resilient to those problems.

Chapter overview

At the heart of Akka is an implementation of the Actor Model, which is a theoretical model of concurrent computation. In this, chapter we will introduce core concepts in Akka by looking at the history of Akka and the actor model. This will give you insight into what Akka is and help you understand what problems it tries to solve. Then, the goals of this book will be introduced with recurring examples that will be used.

After covering these concepts, the chapter will move into setting up your development environment with the tools you need to start building. We will set up our environment, **Integrated Development Environment** (**IDE**), and our first Akka project, including unit testing.

What is Akka

This section will introduce Akka and the actor model. Akka, purportedly named after a mountain in Sweden, is often referred to as a distribution toolkit—a collection of tools that are used to do work across remote computing resources. Akka is a modern implementation of the actor model of concurrency. Akka today could be seen as an evolution of other technologies, borrowing from Erlang's actor model implementation while introducing many new features to aid with building applications that can handle today's high-scale problems.

Actor Model origins

To better understand what Akka is and how it is used, we will take a brief trip through time looking at the Actor model to understand what it is and how it has evolved into a framework for building fault-tolerant distributed systems in Akka today.

The actor model of concurrency was originally a theoretical model of concurrent computation proposed in a paper called *A Universal Modular Actor Formalism for Artificial Intelligence* in 1973. We will look at the actor model's qualities here to understand its benefits in aiding our ability to reason about concurrent computation while protecting against common pitfalls in shared state.

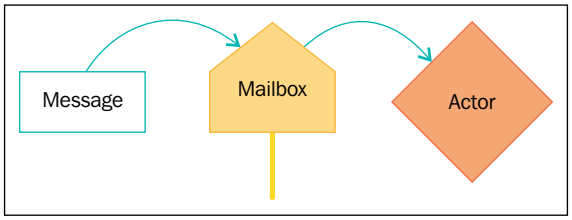
What's an Actor anyway?

First, let's define what an Actor is. In the actor model, an actor is a concurrency primitive; more simply stated, an actor can be thought of as a worker like a process or thread that can do work and take action. It might be helpful to think of an actor as a person in an organization that has a role and responsibility in that organization. Let's say a sushi restaurant. Restaurant staff have to do various pieces of work throughout the day such as preparing dishes for customers.

Actors and Message passing

One of the qualities of an object in object oriented languages is that it can be can be directly invoked–one object can examine or change another object's fields, or invoke its methods. This is fine if a single thread is doing it, but if multiple threads are trying to read and change values at the same time, then synchronization and locks are needed.

Actors differ from objects in that they cannot be directly read, changed, and invoked. Instead, Actors can only communicate with the world outside of them through message passing. Message passing simply means that an actor can be sent a message (object in our case) and can, itself, send messages or reply with a message. While you may draw parallels to passing a parameter to a method, and receiving a return value, message passing is fundamentally different because it happens asynchronously. An actor begins processing a message, and replies to the message, on its own terms when it is ready.



The actor processes messages one at a time, synchronously. The mailbox is essentially a queue of work outstanding for the worker to process. When an actor processes a message, the actor can respond by changing its internal state, creating more actors, or sending more messages to other actors.

The term *Actor System* is often used in implementations to describe a collection of actors and everything related to them including addresses, mailboxes, and configuration.

To reiterate these key concepts:

- **Actor**: A worker concurrency primitive, which synchronously processes messages. Actors can hold state, which can change.
- **Message**: A piece of data used to communicate with processes (for example, Actors).
- **Message-passing**: A software development paradigm where messages are passed to invoke behavior instead of directly invoking the behavior.
- **Mailing address**: Where messages are sent for an actor to process when the actor is free.

- **Mailbox**: The place messages are stored until an actor is able to process the message. This can be viewed as a queue of messages.
- **Actor system**: A collection of actors, their addresses, mailboxes, and configuration, etc.

It might not be obvious yet, but the Actor Model is much easier to reason about than imperative object oriented concurrent applications. Taking a real world example and modeling it in an actor system will help to demonstrate this benefit. Consider a sushi restaurant, We have three actors in this example: a customer, a waiter, and the sushi chef.

Our example starts with the customer telling our waiter their order. The waiter writes it down this onto a piece of paper and places this message in the chef's mailbox (sticks it in the kitchen window). When the chef is free, the chef will pick up the message (order) and start preparing the sushi. The chef will continue to process the message until it's done. When the sushi is prepared, the chef will put this message (plate) in the kitchen window (waiter's mailbox) for the waiter to pick up. The chef can go work on other orders now.

When the waiter has a free minute, the waiter can pick up the food message from the window and deliver it to the customer's mailbox (for example, the table). When the customer is ready, they will process the message by eating the food.