Microsoft Orleans for Developers

Build Cloud-Native, High-Scale, Distributed Systems in .NET Using Orleans

Richard Astbury

Apress®

Microsoft Orleans for Developers

Build Cloud-Native, High-Scale, Distributed Systems in .NET Using Orleans

Richard Astbury

Apress®

Microsoft Orleans for Developers: Build Cloud-Native, High-Scale, Distributed Systems in .NET Using Orleans

Richard Astbury
Woodbridge, UK

ISBN-13 (pbk): 978-1-4842-8166-6 ISBN-13 (electronic): 978-1-4842-8167-3
https://doi.org/10.1007/978-1-4842-8167-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Joan Murray
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub.

Printed on acid-free paper

This book is dedicated to all those who have contributed to the Orleans community.

Table of Contents

About the Author

Richard Astbury works at Microsoft UK, helping software teams build software systems to run in the cloud. Richard is a former Microsoft MVP for Windows Azure. He is often found developing open source software in C# and Node.js, navigating the river on his paddle board, and riding his bike. He lives in rural Suffolk, UK, with his wife, three children, and golden retriever.

About the Technical Reviewer

Sergey Bykov is one of the creators of Orleans and a long-time lead of the project since its inception within Microsoft Research. During his nearly 20 years at Microsoft, he has worked on servers, embedded systems, online and gaming services. His passion has always been about providing tools and frameworks for software engineers, to help them build better, faster, more reliable and scalable systems and applications with less effort. He continues to follow that passion, now at Temporal Technologies.

You can read Sergey's blog at https://dev.to/sergeybykov and follow him on Twitter @sergeybykov.

Introduction

Welcome!

This book aims to help the experienced dotnet developer understand what Orleans is, what problems it solves, and how to get started with an Orleans project. No prior knowledge of distributed systems is required, but it is useful to have a good grasp of C#, ASP.NET Core, and unit testing.

The first two chapters provide a brief history and backstory, helping you understand where Orleans began and the core concepts, namely, Grains and Silos.

The main body of the book starts with a "Hello World" example and then explores several of the features of Orleans by building more functionality into this application. This helps us to understand the functionality of Orleans by taking practical steps and building something that's a, sort of, real system.

I have included a couple of chapters on more advanced features, including optimizations. I don't attempt to add all of these to the sample application, and instead explain how they're used and the problems they solve.

Software is built by people, and it was important to me to include some interviews in this book from people who are/were in the core team as well as the community. I hope you enjoy reading their perspectives.

Orleans is a great fit for some really interesting challenges, particularly around high-scale, real-time systems, but it doesn't fit every problem. Part of being an experienced developer is knowing the best tool to use for a given problem. My hope is that this book will help you develop your intuition, so when you have finished reading (however far you get!), you'll know an Orleans-shaped problem when you see one.

CHAPTER 1

Fundamentals

Microsoft Orleans is a free, open source framework built on Microsoft .NET, which provides the developer with a simple programming model enabling them to build software which can scale from a single machine to hundreds of servers.

You can think of Orleans as a distributed runtime, which runs software across a cluster of servers, treating them as a single address space. This allows the developer to build software which keeps lots of data in memory, by spreading objects across the cluster's shared memory space. This enables low-latency systems, where requests can be processed using data held in memory without deferring to a database. This allows the system to deal with a high volume of traffic.

Orleans is designed to support cloud-native applications, with elasticity and fault tolerance.

Orleans is a "batteries included" framework, which ships with many of the features required for distributed systems built in.

Motivation for Orleans

When I first started serious programming in the early 1990s, it was just me, my PC, and Turbo Pascal 7. The terrible software I wrote just ran on a single machine, an IBM PS/2 8086, which had a single CPU core and no network.

Fast-forward to the present day and the progress made in computer hardware is astonishing. Computers now have multiple cores, with 440 being the current state of the art. Cloud computing provides us with near instant rental to computers with per-minute billing, almost limitless horizontal scalability, and accessible to anyone via a ubiquitous global network – the Internet.

R. Astbury, *Microsoft Orleans for Developers*, https://doi.org/10.1007/978-1-4842-8167-3_1

While programming languages have also evolved, most mainstream languages are still generally focused on running a sequence of instructions on a single computer. Some languages' runtimes don't even have support for multithreading and distributing work across multiple cores.

In order to satisfy the requirements of real-time web applications and IoT gateways, developers are required to build systems that can deal with high volumes of traffic, handle hardware failure, and do this with elasticity to optimize for cost. We need to make efficient use of the hardware, which means running code efficiently on multicore computers, horizontally scaled in the cloud.

I think most experienced developers would agree that programming concurrent or distributed systems is hard. Bugs in concurrent code, such as race conditions, are sometimes hard to detect, to replicate, to debug, and to test. Software that runs across networks must respond to transient outage and delays, which are common in a cloud environment. The developer will often have to consider eventual consistency or reconciling state mutations across multiple nodes.

Languages have emerged that certainly help the developer to build concurrent and networked software, with reference to Go and Rust in particular, but my opinion is that while these languages offer constructs and safety for concurrent programming, they don't provide a finished solution for a developer to build software that's distributed by default.

In response to these challenges, the design I see the majority of developers take is to make application code stateless and use the database to handle concurrency using transactions and optimistic locking. This allows horizontal scalability of the application server. The servers do not cooperate or communicate between each other, as every state mutation is performed by the database. This requires the web server to gather the required records from the database to respond to each request. This is an excellent solution, which has been proven to work countless times. However, there are a couple of problems that can emerge:

1. The database can become a performance bottleneck, and while databases can be scaled vertically (i.e., upgrade the hardware), you will eventually reach a limit. Some databases can be horizontally scaled. However, this often comes with a loss in some guarantees, such as eventual consistency rather than strong consistency.