Introducing Microsoft Orleans

Implementing Cloud-Native Services with a Virtual Actor Framework

—

Thomas Nelson

Apress®

Introducing Microsoft Orleans

Implementing Cloud-Native Services with a Virtual Actor Framework

Thomas Nelson

Apress®

Introducing Microsoft Orleans: Implementing Cloud-Native Services with a Virtual Actor Framework

Thomas Nelson
Louisville, KY, USA

ISBN-13 (pbk): 978-1-4842-8013-3 ISBN-13 (electronic): 978-1-4842-8014-0
https://doi.org/10.1007/978-1-4842-8014-0

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

As with every journey, it is never made alone.
I would like to thank those who have encouraged me to reach several goals, including authoring this book that I now share with you.

Mom (Roxanne Lucas), you have cheered for me since day one. You have always pushed me to better myself with love and support. Thank you for everything. Without you, I would not have been the person I am today.

Aaron Lucas, MD, your advice has never led me astray. "This too shall pass," and it has. You took me in as your own and taught me much. I am grateful for your counsel.

Dad (Vincent Nelson Jr.), without you, I would not be where I am today. No matter the seriousness of a problem, you always find the silver lining through humor and kindness.

Leo and Cora, my wonderful fur-kids, I treasure every day we have together.

Cynthia F. C. Hill, PhD, you have single-handedly reconstructed my writing. This book would not have come to fruition without your guidance. You helped better my life.

Corey, Britney, Knox, Roscoe Cooley, you are my second family. You have been my rock during difficult times and accepted me as family. I always look forward to working on the next game dev jam.

Joan Murray and Jill Balzano, thank you for working with me on this book. It has been a pleasure. This book would never have happened without you both.

Micheal Kelly and Reece Schwegman, great friends, great advice, and great times. You guys are irreplaceable.

Phuong Pham, Kevin Turner, Alex Olson, we are lifetime friends. We have urged each other to do our best, sharing strength and calm.

My work friends: Dat Trinh, Karuna Byrd, Lorenzo Castro, Barry Chase, Keith Meyers, Hussein Rashid, and many others. Thank you for giving me the opportunity to fulfill my dream as a software architect. You each have taught me so much, which has helped me learn and expand to better myself. I am grateful for your time, support, and critical evaluations.

Mei Ying Tan, my counterpart in class, you have become a great friend and have brought out my academic competitive side. You make the classes exciting, and I am grateful we have met.

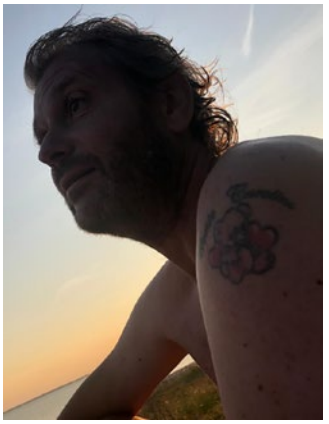
Lydia Pearce, Amy Rawson, Kiera, HBomb, Angel Ortiz, Dean Dusk, BunchiJumpi, Ha Nguyen, and Marissa Pelchat, thank you for your time, unwavering support, and wonderful friendship. The pandemic has been horrible; however, it did bring us together, and I am grateful to have each one of you in my life.

Table of Contents

About the Author

Thomas Nelson, Lead Cloud Architect and Microsoft Certified Azure Solutions Architect Expert, has worked in several technical fields spanning from the graphic design of websites to development and architecture. During his 10+ years of backend development, his interest has gravitated toward DevSecOps and automation. He is involved in core automation for cloud development and infrastructure for enterprises. He enjoys teaching others and is often found at local meetups presenting various technologies, patterns, and software examples. He is thrilled to be using Orleans and considers it one of those wonderful and valuable frameworks that should be in the tool kit of every architect and backend developer. Also, he is pleased to have extensive experience with monolithic and microservice systems to build cloud-native solutions, including actor framework back ends. He has an associate's degree in graphic design and bachelor's degree in computer information systems and is currently attending Harvard Extension School pursuing his master's degree in information management systems.

About the Technical Reviewer



Carsten Thomsen is a backend developer primarily but working with smaller frontend bits as well. He has authored and reviewed a number of books and created numerous Microsoft Learn courses, all to do with software development. He works as a freelancer/contractor in various countries in Europe, with Azure, Visual Studio, Azure DevOps, and GitHub as some of his tools. Being an exceptional troubleshooter, asking the right questions, including the less logical ones, in a most logical to least logical fashion, he also enjoys working with architecture, research, analysis, development, testing, and bug fixing. Carsten is a very good communicator with great mentoring and team-lead skills and great skills researching and presenting new material.

Acknowledgments

Thank you Joshua Grimaud for your contributions to Chapter 9. Without your expertise, the chapter could not have been completed. Josh is an expert in programming and specifically DevSecOps. He is a creative thinker and a solution-oriented person. Also, he is a great friend and has an extreme passion for learning tech and creating music.

Introduction

This book was written to introduce the Microsoft Orleans framework and to provide an understanding of why it was created and how it can advance your current and potential applications. Actor model frameworks are not widely taught or known in mainstream development. Since the frameworks are not common knowledge, this book was written with beginners and intermediate developers in mind. An overview of Orleans is provided concerning topics such as how it can enhance projects, and how to get started in the code base.

First, we cover the origins of Orleans which was established in 2010. This book explains what it was built to accomplish and examples that showcase its successes in production. Next, we look at how Orleans works concerting lifecycles, which removes a massive amount of overhead from developers. Lifecycles are emphasized since they play a significant role in what makes Orleans unique by abstracting the work from the developers and making projects faster and more maintainable, compared with previous actor model frameworks. Next, we dive into commonly used architectures – monolithic and microservices – to show how Orleans can possibly enhance them. New and existing developers can benefit by learning how Orleans can extend their monolithic or microservice patterns.

The remainder of this book is hands-on coding where we set up a project and walk through adding features to it such as

- Timers and reminders
- Unit tests
- Dashboard

Then we walk through deploying Orleans to AKS. What good is our work if we cannot deploy for others to use it? I believe that all development books should have a deployment section so that the applications can be treated as real work. It allows you to deploy as we would in a business. We walk the setup of an app, code it, and deploy it. The pipeline will trigger when there is a commit and deploy if it meets the standards.

Finally, we conclude by summarizing what we have covered. We discuss where you can go next to further your journey with Orleans. I would treat this book as a strong starting point. It provides a strong understanding of what Orleans can do for you, how coding differs, resource requirements, and next steps.

CHAPTER 1

A Primer on Microsoft Orleans and the Actor Model

Origins of Orleans

Microsoft Orleans is an open source project, 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, allowing the .NET developer to easily build software capable of processing high volumes of data and deployable to the cloud or on-premises.

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

First, what is an actor model? An actor model is defined as "a mathematical model of concurrent computation that treats 'actor' as the universal primitive of concurrent computation" (Patent Issued for Actor Model Programming (USPTO 10,768,902), 2020). Actor model frameworks use the model as a basis on which the frameworks pass messages between actors, and actors are created on a needed basis. Actor model frameworks are able to take advantage of concurrency through multicore computers and software abstraction. This means that actor model frameworks are able to process millions and billions of messages in real time or near-real time. We will cover this more in this chapter and throughout the book.

T. Nelson, *Introducing Microsoft Orleans*, https://doi.org/10.1007/978-1-4842-8014-0_1

Orleans, created in 2010 by Microsoft Research, is an actor framework that harnesses the inert capabilities of cloud architecture (Orleans – Virtual Actors, 2018). The framework helps distribution experts and novices by using prebuilt libraries to create globally distributed, highly available, highly elastic robust solutions. The libraries remove many complexities –such as lifecycles – while using a common programming language. Ultimately, Orleans allows a single developer to create an extremely scalable application without being an expert in distributed system development. Other actor model frameworks require the developer to determine how to handle the life span of each actor and monitor and check health. Also, Orleans is a production-proven – through the use of IoT applications and game studios – and open source framework for Microsoft's internal and external flagship applications.

Origins and Use Cases of Orleans

Initially, Orleans was created to help expand cloud computing to a larger developer audience when cloud computing was new and gaining momentum. Orleans has the ability to provide hyper-scalable interactive services that support high throughput and availability with low latency in the cloud and local systems. Existing actor frameworks fulfill the technical needs; however, the additional overhead of knowledge, experience, and complex initial coding can be challenging to accomplish. Attempting to satisfy these requirements with a traditional three-tier service design is challenging as well. Orleans combines the common coding structures of three-tier development and .NET Core libraries, reducing entry barriers.

In addition to cloud development, it has been used for backend development of video games, such as the blockbusters *Halo 4, 5* and *Gears of War 4*. Cloud computing is extremely helpful in hosting and running distributed services for a global audience, such as gaming. Before cloud computing, the World Wide Web grew from its infancy, where servers were not readily globally distributed for businesses' applications, and clients accepted latency as a part of the Internet. I recall waiting for images to download a row or two at a time when using a 33.6k dial-up modem. Overtime, companies have risen to global entities, and cloud architecture creates the ability to cater to these clients in a distributed and real-time-like capacity. These changes lead to several new patterns emerging based on lessons learned, leading to Orleans' creation. To understand this process, we will discuss the development of services in a high-level perspective where we cover monolithic services, microservices, and Orleans.

Initially, monolithic applications were used to conduct multiple scopes of work. For instance, an application may house business logic server items, such as account, order, and shipping. Housing these together is necessarily a bad option, as each item depends on a single team. It allows the ability to walk through a single application for debugging and traceability. Unfortunately, this ability is associated with cons as well.

Monolithic applications have a large footprint. Since their scope maintains several items in the business, it is likely supported by several teams. Housing multiple large workflows together, a single application can get large and possibly unmaintainable overtime, if it is not watched closely. The larger the application, the more hardware it will require to support it, which can be costly in hosting hardware and perhaps startup time when needed on demand by consumers.

The application is usually tightly coupled and requires the teams to work and maintain open communication. This can hinder deployment as it is difficult, if not impossible, to deploy without all of the items being implemented. Feature flags can toggle various logic. However, strict discipline is required for the teams to stay within scope and communicate the changes internally; otherwise, it will delay deployment until all the work has been completed.

Overtime, *microservices* – a term coined by Dr. Peter Rogers – emerged to create granularly scoped applications that interact with one another to complete a holistic action(s). *Microservice* is a general term that refers to the scope of work of the application that is loosely coupled and generally uses standards such as REST, XML, JSON, and HTTP. Based on the preceding monolithic example, each logically scoped item that we mentioned – account, order, and shipping – is separated into its own service. Decoupling removes the need for multiple teams to maintain a single application and reduces complexity, thus facilitating communication and possibly on-time code completion to meet deployment deadlines. The footprints are a portion of what we would use for monolithic applications, which leads to the ability to scale out on demand.

Also, decoupling allows complex and extensive systems to be built, tested, and deployed individually. This adds value in an Minimal Viable Product (MVP) manner by being able to proceed with testing and deployment before the entire application completed whereas, the legacy application cannot move forward until all of the dependent work has been completed. Feature flags can be implemented however, the teams need to implement and maintain. When we refer to microservices in further chapters, we will be referring to a three-tier architecture (client, back end, database).

I have worked with monolithic services and microservices in several companies. I have gone through several transformation processes of converting monolithic applications to microservices. Microservice patterns are a solid choice for many of the

current business needs and are supported easily by cloud computing. Cloud computing monitors the applications on set triggers to determine health and the need for scaling, among additional items.

I have moved between companies that had engineering teams of 40 to a few hundred to thousands. Each move to a larger company solidified --to me-- the need for maintainable services as cross-team communication was harder to sustain. This was/is due to members constantly moving or leaving within the teams, which removed the oversight and communication needed to maintain larger, monolithic applications. Again, this depends on the company structure, as the smaller teams made monolithic applications acceptable to work within.

This book will not debate the legitimacy of microservice vs. monolithic architectures but will discuss how the Orleans implementation can enhance applications and when Orleans is a good fit. Ultimately, the application needs and company structure should determine the design and technology that are chosen. The common goal of applications is to be a robust, extensible, and available system for consumers. For instance, a project might be required to implement a cloud hosting's concurrency and distribution as well as its dynamic and interactive in nature, which allows actor frameworks to flourish (Bernstein. P., Bykov, S.). Orleans was created, in part, to take advantage of these native cloud architecture abilities. The solution should be a negotiation between company ability and consumer needs.

To take full advantage of cloud computing, we can use Orleans, which is a production-ready – and battle-tested by supporting projects, like AAA video games and smart home IoT devices – framework written in C# and .NET Core and distributed in NuGet (New Get) packages with version 2.0+, which creates cloud-agnostic solutions. This allows .NET developers to extend their applications within an already familiar ecosystem. Orchestration overhead is reduced through the framework's ability to maintain lifecycles and elasticity and harness the actor model's concurrency. The actor model uses actors, which are fine-grained isolated objects that receive and send asynchronous messages with the ability to create additional actors (Bernstein & Bykov, 2016).

Actor frameworks can be complicated for developers to understand since they can be complex and are not commonly taught in universities. It usually takes a team or teams of advanced engineers to create and maintain a system. Orleans came into existence to help reduce the overhead and barriers and take advantage of the actor model's capabilities. In 2015, Microsoft chose to open-source Orleans, which resulted in