Unlock Your Potential and Master Web Development

PROGRAMMING STEP BY STEP AND MORE AN INSTRUCTIVE GUIDE

DevOps ENGINEERING
CLOUD ENGINEER

A Professional Instructional Series for Modern Web Engineering

Book 9 - Cloud-Native Platform Engineering

Security, Observability, Scalability, and Microservices - – Capstone Project

Rene F. Ruano Domínguez

DEDICATION

TO THE GENERATIONS OF THE FUTURE, MILLENNIALS, BABY BOOMERS AND BEYOND.

This How-To is dedicated to everyone: To you, the brilliant young minds shaping the world with your ingenuity and creativity. To the millennials, who navigate the digital age with ease and constantly seek new ways to innovate. To the baby boomers, who pioneered the technological revolution and now contribute their experience and wisdom. To the retirees who dream of staying up-to-date, filling their free time with useful hours, and exercising their minds.

FOREWARD

Digital transformation has driven the adoption of cloud-native platforms capable of offering greater flexibility, scalability, and resilience than traditional infrastructures. In this context, this book marks a new stage in the series, focusing on the technologies and practices required to deploy applications and services into real-world production environments using Docker, Google Cloud Run, Cloud Build, Artifact Registry, and Cloud SQL.

Beyond merely learning individual tools, the book presents a comprehensive architectural vision aimed at building modern applications based on modularity, automation, and microservices. The chapters explore fundamental aspects such as security, secret management, observability, continuous operations, scalability, and distributed architectures, providing the knowledge needed to design robust, resilient, and scalable platforms.

To culminate the learning process, the book includes a capstone project—*git-hub-api-cloud*—an enterprise-grade cloud-native platform that allows readers to apply all the concepts covered in a practical setting. Through this project, readers will understand how to integrate Docker, Cloud Run, Cloud SQL, microservices, security, observability, scalability, and multi-cloud strategies into a complete solution, thereby solidifying a modern perspective on platform engineering and distributed cloud applications.

Beyond the technical aspects addressed in this work, it is important to acknowledge the effort invested in research, organization, and systematization to consolidate the content presented here. Creating this book required an ongoing process of analysis, experimentation, practical validation, and technological updating, with the aim of transforming scattered knowledge into a coherent, accessible, and results-oriented educational structure. Each chapter is designed as part of a progressive learning journey, where theoretical concepts are closely linked to real-world experiences in cloud system development, deployment, and operations.

The author's dedication extends beyond the mere writing of technical content; it is part of a broader vision aimed at facilitating effective knowledge acquisition through a practice-based approach. Through a Learning Management System—complemented by repositories, guided projects, digital resources, and progressive exercises—the aim is for

students to access information from various perspectives and via diverse learning tools, thereby strengthening their conceptual understanding and developing competencies applicable to real-world scenarios.

Experience shows that true learning stems not merely from reading, but from the constant interplay between theory and practice. For this reason, the projects, labs, deployments, and exercises featured throughout this book are designed to enable readers to actively build knowledge, validating each concept through direct application. The ultimate goal is for students not only to understand how modern cloud-native technologies work but also to gain the confidence and expertise needed to design, implement, and operate their own technology solutions—applying professional standards grounded in the practical experience gained from each project.

The author hopes that the content presented here serves as inspiration to continue exploring the vast universe of cloud computing and to confidently tackle the challenges inherent in developing modern systems. Each chapter represents a step forward in the journey toward building platforms capable of transforming ideas into real, scalable, and future-ready solutions.

With admiration,

Idalmy Baluja Conde

Index

GRATITUDE

To my family, my colleagues, my friends.

Thank you, thank you so much for the unconditional support you've given me since I started this project. And to those who didn't believe in me, I'm grateful for motivating me to prove that it was worth it.
Without the encouragement, suggestions, trust they placed in me, and the personal challenge of creating a useful product, this project would not have been possible.

OTHER BOOKS BY THE AUTHOR

If you would like to continue exploring the Programming Step by Step and More series, you can access all related volumes, updates, and additional books available on Amazon through the QR code below.
All books in this series are available on Amazon.
Scan the QR code to discover more:

Book 9 - Cloud-Native Platform Engineering

Security, Observability, Scalability, and Microservices - – Capstone Project

Rene F. Ruano Domínguez

1. INTRODUCTION

The development of modern applications has undergone a profound transformation in recent years. What was once limited to building monolithic programs running on local servers has evolved into distributed ecosystems capable of operating simultaneously across multiple environments, scaling on demand, and responding to millions of requests from users, smart devices, and interconnected services across the Internet.

The previous volumes in this collection have guided the reader through this evolutionary process. Book 7 explored the development of modern APIs using Node.js, Express, and serverless architectures, while also examining deployment on platforms such as Vercel and the use of Git repositories as a central element of the development lifecycle.

Later, in Book 8, the focus shifted toward API engineering through RESTful design, OpenAPI/Swagger documentation, automated testing, service versioning, and continuous integration and deployment processes on platforms like Render, all oriented toward production-grade environments.

Book 9 represents the next natural step in this technological progression. Its purpose is to introduce the reader to the design, construction, and deployment of cloud-native architectures based on containers and microservices, incorporating practices currently used in enterprise-level platforms worldwide. Throughout its chapters, the reader will study technologies such as Docker, Google Cloud Run, Cloud Build, and various automation mechanisms that enable the transformation of traditional applications into distributed services ready to operate in the cloud.

Beyond the simple act of deploying applications, this book focuses on understanding the architectural principles that underpin modern systems: functional decoupling, horizontal scalability, fault tolerance, modularity, component reuse, and process automation. These concepts form the foundation of today's technological environments, used in fields as diverse as the Internet of Things (IoT), Artificial Intelligence (AI), enterprise SaaS platforms, digital financial services, real-time data analytics, and distributed computing ecosystems.

The reader will learn how to build Docker images, manage containers, deploy services on Google Cloud Run, integrate automated build processes, and design modular architectures prepared to evolve into independent microservice models. The book also examines the differences among the various cloud platforms studied throughout this collection, clarifying when to use serverless solutions, when to rely on containers, and how to select the most appropriate architecture based on the technical and operational goals of each project.

As the practical core of the book, new projects will be developed based on the evolution of previously implemented architectures, progressively transforming them into more robust, scalable solutions suitable for production environments. This approach allows the reader to understand not only the individual function of each technology but also how they interact to build complete systems capable of meeting the current challenges of software engineering.

The ultimate goal of this volume is not merely to deploy applications in the cloud, but to provide a comprehensive understanding of the architectures shaping the present and future of software development. By the end of this journey, the reader will possess the knowledge required to design, implement, and operate modern platforms based on containers, microservices, and managed cloud services—laying the groundwork for exploring even more advanced technologies related to automation, observability, container orchestration, and next-generation distributed ecosystems.

2. CONTAINERIZATION WITH DOCKER

Containerization is one of the fundamental pillars of modern cloud-native architectures. Thanks to Docker, developers can package applications, dependencies, configurations, and runtime environments into portable units called containers, ensuring that an application functions consistently regardless of the operating system or infrastructure where it is deployed.

In this chapter, we'll learn the fundamental principles of Docker, build our first containers, and prepare the projects for deployment on Google Cloud Run. Throughout the process, we'll explore concepts such as images, containers, Dockerfiles, Docker Compose, and local testing strategies, culminating in the complete containerization of the EnergyNer project's APIs.

2.1. Introduction to Containers

As web applications grow in complexity and need to run in different environments, ensuring they function identically throughout development, testing, and production becomes essential. One of the biggest historical challenges for developers has been precisely this inconsistency between environments: a project might run flawlessly on the local machine but fail on a remote server due to differences in dependencies, configurations, or operating system versions.

Containers emerged as the definitive solution to this problem. By packaging everything needed to run an application—code, libraries, dependencies, and runtime environment—into a lightweight, portable unit, they allow the application to function consistently on any compatible system. Today, containerization is a cornerstone of cloud-native architectures and modern distributed systems. In this chapter, we will explore the fundamentals of Docker and how this technology transforms traditional applications into services ready to run on platforms like Google Cloud Run, laying the groundwork for modular architectures and scalable deployments.

2.1.1. Evolution of Execution Environments

Technology has constantly evolved to solve the challenges developers face when running applications in different environments. For years, one of the

biggest problems has been ensuring that an application functions identically across all scenarios: from the programmer's local computer to remote servers with completely different configurations. This evolution has followed a progressive path—Local Application → Dedicated Server → Virtual Machine → Container → Cloud-Native—where each stage emerges to overcome the limitations of the previous one.

Today we understand this evolution as a natural process toward standardization, portability, and scalability. Containers and cloud-native architectures represent the most advanced stage of this journey, enabling applications to run consistently, automatically, and with high efficiency regardless of the physical or virtual environment in which they are deployed.

2.1.1.1. Cloud – native

The term cloud-native describes a software design and development approach specifically intended to run in the cloud, taking full advantage of its elasticity, automation, and scalability. It's not simply about "moving an application to a remote server," but about creating systems that are designed from the ground up to operate in distributed, dynamic, and highly automated environments.

A cloud-native application is characterized by being modular, containerized, automatically scalable, observable, resilient, and decoupled from the physical infrastructure. This allows services to be deployed quickly, updated without disruption, adapted to real-time demand, and integrated with managed service ecosystems such as databases, message queues, registries, load balancers, and CI/CD tools.

In essence, cloud-native is not a specific technology, but an architectural philosophy that combines containers, microservices, automation, declarative infrastructure, and serverless platforms to build applications capable of evolving, scaling, and operating efficiently in the cloud.

2.1.1.2. Progressive evolution of technology

As we pointed out earlier, this evolution has followed a progressive path from:

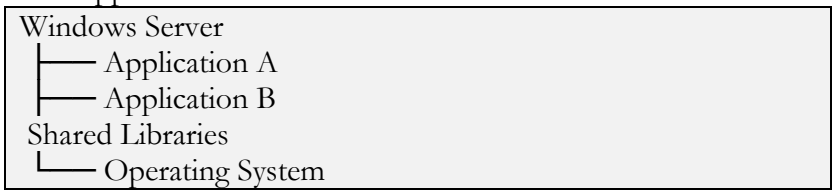
a) Applications installed directly on the operating system

In the early years of web and desktop development, applications were installed directly onto the host operating system.

Characteristics:

- Total dependence on the operating system.
- Manual installation of libraries.
- Individual configuration of each server.
- High probability of conflicts between applications.

Table 1. Applications installed on the server



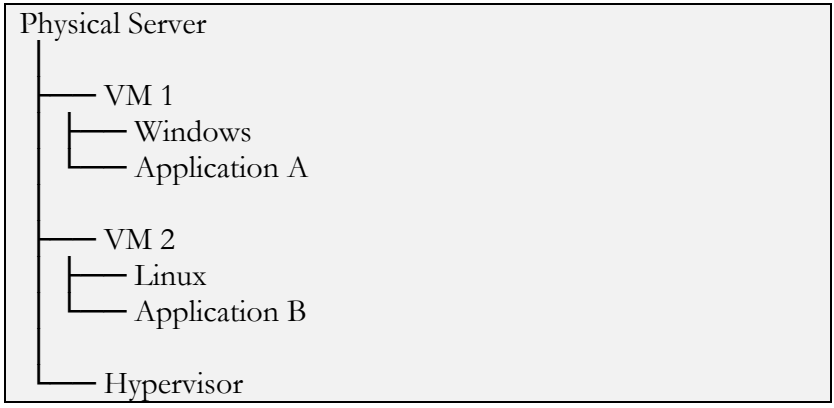
Main problem:

An update or modification of one application could affect the operation of others.

b) Virtualization using Virtual Machines

To resolve conflicts between applications, virtual machines (VMs) emerged. Each application could run within a completely independent operating system.

Table 2. Virtual Machines



Advantages:

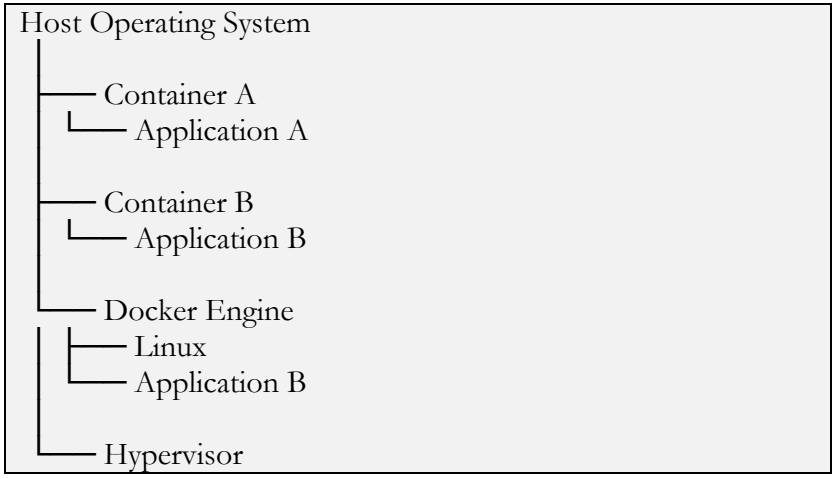
- Complete isolation.
- Greater security.

• Compatibility between different operating systems.
Disadvantages:
• High memory consumption.
• Slow start.
• Higher operating costs.

c) Appearance of the Containers

Containers emerged as a lighter alternative to virtual machines. Instead of virtualizing an entire operating system, containers share the host system's kernel and isolate only the application and its dependencies.

Table 3. Containers



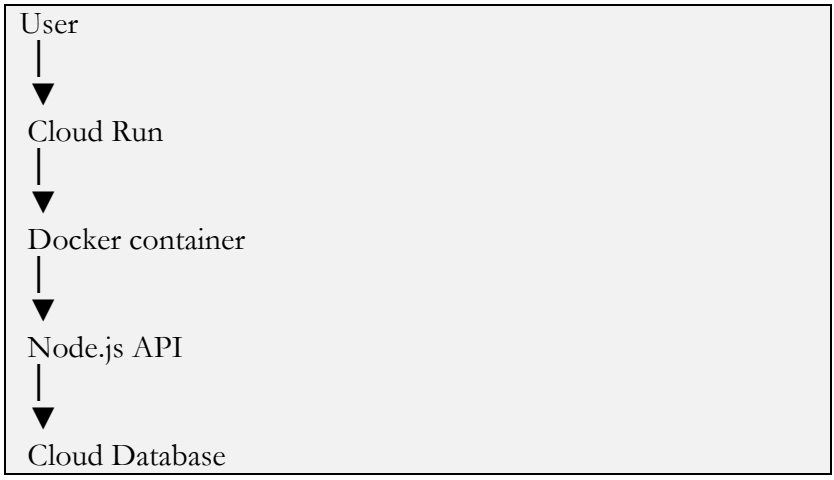
Advantages:
• Lower consumption of resources.
• Almost instant start.
• Portability.
• Ease of deployment.

d) Cloud-Native Architectures

The natural evolution of containers gave rise to cloud-native architectures. In this model, applications are designed from the ground up to operate in the cloud.
Characteristics:
• Automatic scaling.
• High availability.
• Microservices.

• Deployment automation.
• Integrated observability.

Table 4. Cloud-Native Architecture



e) Relationship of the evolution with the projects developed in this collection

The technological evolution studied in Books 7 and 8 and that which we will address in Book 9 precisely reflects this transformation.

Table 5. Technological evolution

Book	Predominant Technology
Book 7	APIs Node.js, Express, Vercel
Book 8	OpenAPI, Testing, Render
Book 9	Docker, Cloud Run y Microservices

Whereas in previous books we deployed applications directly onto managed cloud platforms, in this volume 9 we will learn to build our own containers, gaining greater control over the runtime environment and preparing our projects for modern enterprise architectures.

This last subsection (e) is very valuable because it immediately connects the theory with the journey the reader has already taken in Books 7 and 8,

preventing the chapter from seeming like an isolated explanation of Docker.

2.1.2. Traditional deployment problems

Next, we will analyze the most frequent problems that have historically affected application deployment processes.

a) Differences between Environments

One of the most common problems occurs when development, testing, and production environments don't have exactly the same configuration. For example, a developer might create and test a Node.js API on a computer running Windows 11, Node.js version 22, and certain libraries installed locally. However, when deploying that same application to a Linux server with a different version of Node.js, errors may appear that were never present during development.

Table 1. Example of differences between environments.

Environment	Operating System	Node.js	State
Development	Windows 11	22.x	Works
Testing	Ubuntu Linux	20.x	Error
Production	Debian Linux	18.x	Error

These differences required specific configurations for each server, increasing the complexity of administration and making it difficult to reproduce the results.

b) Incompatible dependencies

Modern applications rely on multiple external libraries to provide advanced functionality. These dependencies can include frameworks, authentication modules, database access tools, or components for communicating with external APIs.

When a dependency required by the application is not installed on the target server, or when an incompatible version exists, execution may fail partially or completely.

For example, an API developed with Express might require certain features only available in recent versions of the framework. If the server is running an older version, some instructions might stop working correctly.

Table 2. Problems arising from incompatible dependencies.

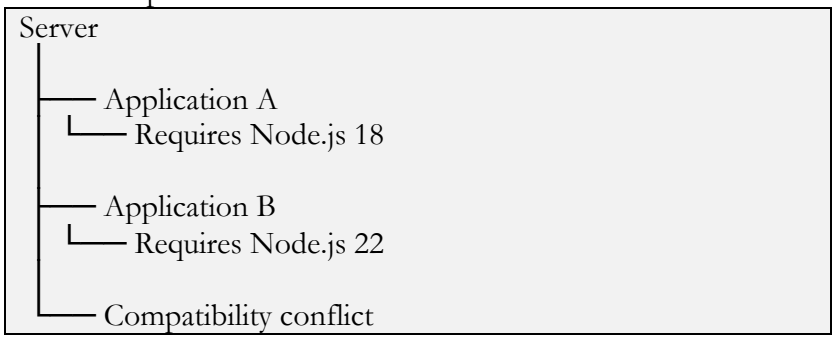
Dependency	Required version	Installed version
Express	5.x	4.x
CORS	2.8.x	1.x
PM2	5.x	Not installed

In complex environments, where dozens or even hundreds of dependencies are involved, this problem can become a constant source of errors.

c) Version Conflicts

One dependency-related problem is the coexistence of multiple applications on the same server. Suppose one application requires Node.js version 18 while another needs Node.js version 22. Installing both versions and ensuring each application uses the correct one can be complex and lead to operational conflicts.

Table 3. Example of version conflict.



Similar situations can occur with databases, libraries, drivers, or any other component shared by multiple applications. Manually managing these conflicts significantly increases maintenance effort and reduces the stability of the production environment.

d) Portability problems

Portability refers to the ability to move an application from one environment to another without significant modifications. Traditionally, moving an application from a local computer to a remote server required repeating multiple tasks:

- Install dependencies.
- Configure environment variables.
- Adjust permissions.
- Configure ports.
- Check compatibility.

Each new server involved repeating virtually the same installation process.

- Developer
- Copy files
- Install dependencies
- Configure server
- Correct errors
- Operational application

This procedure was slow, prone to human error, and difficult to automate, especially when an organization needed to deploy the same application on multiple servers.

2.1.3. What is a Container?

A container is a lightweight, portable, and self-contained software unit that includes everything needed to run an application consistently in any environment. The application's source code, libraries, dependencies, configurations, and components required for its operation are stored within a container, eliminating the differences that traditionally existed between development, testing, and production environments.

Unlike virtual machines, containers don't need to run a full operating system. Instead, they share the host operating system's kernel, which significantly reduces resource consumption and enables much faster startup times. Thanks to this feature, containers have become the dominant technology for deploying cloud-native applications, microservices, and scalable platforms.

2.1.3.1. Components of a Container

Although from the outside a container appears to be a single unit, internally it is composed of several fundamental elements:

- **Application:** Source code that executes the main functionality of the system.
- **Dependencies:** Libraries, modules, and frameworks required by the application.
- **Configuration files**: Environment variables, execution parameters, and specific configurations.
- **Runtime environment:** Components necessary for the application to function correctly, such as Node.js, Python, or Java.
- **Base system:** Minimum set of operating system files on which the application runs.

Table 1. Basic components of a container.

2.1.3.2. Container Characteristics

Containers have a number of characteristics that explain their widespread adoption in modern development and deployment environments:

- **Portability:** They can run identically on local computers, physical servers, virtual machines, or cloud platforms.
- **Isolation:** Each container works independently, avoiding conflicts with other applications.
- **Lightweight:** They consume fewer resources than virtual machines by sharing the operating system kernel.
- **Scalability:** They allow you to create multiple instances of an application quickly and efficiently.
- **Consistency:** They ensure that the application behaves exactly the same in all environments.
- **Automation:** They facilitate integration with continuous construction, testing, and deployment (CI/CD) processes.

In practical terms, a container can be understood as a standardized package that encapsulates an application and everything needed to run it. This ability to move the same environment between different infrastructures is one of the fundamental pillars of modern architectures based on Docker, Kubernetes, and cloud services like Google Cloud Run.

2.1.4. Containers versus Virtual Machines

Virtual machines offer complete isolation by incorporating their own operating system, but they consume more resources and require longer boot times. Docker containers, on the other hand, provide a lighter, more portable, and more efficient solution, becoming the predominant technology for deploying modern applications in cloud environments.

Table 1. Comparison between VM and Docker.

Characteristics	Virtual Machine (VM)	Docker Container
Level of virtualization	Virtualizes the entire hardware stack.	Virtualizes only the application and its dependencies.
Operating system	Each VM includes its own operating system.	Shares the host operating system kernel.
Resource consumption	High (CPU, memory, and storage).	Low and efficient.
Startup time	Minutes.	Seconds or milliseconds.
Size	Typically several gigabytes.	Usually tens or hundreds of megabytes.
Portability	Limited by the guest operating system.	High portability across different environments.
Scalability	Slower due to higher resource usage.	Fast and flexible.
Cloud deployment	Suitable for traditional applications.	Ideal for cloud-native architectures and microservices.
Usage examples	VMware, VirtualBox, Hyper-V.	Docker, Cloud Run, Kubernetes.

2.1.5. Benefits of containerization

Containers offer numerous advantages that have driven their adoption in modern software architectures. Their portability allows an application to run identically in different environments, from development teams to cloud platforms. Consistency ensures that the application's behavior is the same regardless of where it is deployed, eliminating problems arising from different configurations. Thanks to their lightweight design, they facilitate high scalability, allowing instances to be created or deleted quickly on demand. Furthermore, they provide isolation between applications, preventing dependency conflicts or shared configurations. Finally, their rapid deployment allows new services to be launched in a matter of seconds, optimizing development, testing, and production processes.

2.1.6. Current use cases

Containers are now widely used in a variety of technology solutions. In API development, they enable the consistent and scalable deployment of services across different environments. In microservices architectures, they facilitate the independence and autonomous deployment of each system component. SaaS (Software as a Service) platforms leverage containers to manage thousands of users and update applications without interruption. In the field of Artificial Intelligence (AI), they enable the portable execution of models, training environments, and inference services. Meanwhile, Internet of Things (IoT) solutions use containers to process data from connected devices and deploy distributed applications from the network edge to the cloud. Thanks to their flexibility, efficiency, and automation capabilities, containers have become a fundamental technology for the development of modern systems.

2.2. Docker Architecture

Docker architecture provides the fundamental model for consistently building, packaging, and running applications in any environment. Its design is based on separating the application from the infrastructure, encapsulating code, dependencies, and configurations within lightweight containers that run on a specialized engine. Understanding this architecture is essential for visualizing how ….

…..

….

….
With this project now complete, we have successfully launched a modular and independent architecture online, designed to facilitate the integration of new APIs. The project's environment and visual identity accurately reflect its purpose: to offer energy calculation services on a clear, scalable platform aligned with its thematic content.

We can verify the deployment of the project developed in this Chapter by visiting the publication "Guide to deploying APIs in Google Cloud" by accessing this page and scanning the QR code on the right.

…
…
…
…
…
…

…
…
…

…
…
…

…
…
…

…
…
…

…
…
…

…
…

10. CONCLUSIONS OF BOOK 9

The development of this book provided a progressive and practical journey through the complete lifecycle of a modern application—from its local construction to its operation on a production cloud-native infrastructure. Across nine interconnected chapters, we covered the technological, architectural, and operational foundations required to design, deploy, secure, monitor, and scale services based on containers and cloud computing.

The first major conclusion is that containerization via Docker constitutes one of the fundamental pillars of modern software development. The concepts of images, containers, Dockerfile, Docker Compose, and dependency management demonstrated how to eliminate traditional problems associated with discrepancies between development, testing, and production environments. The ability to encapsulate entire applications into portable and reproducible units considerably simplifies deployment and maintenance processes.

The study of Google Cloud Run provided insight into the evolution toward event-driven, pay-as-you-go serverless models. The seamless integration of Artifact Registry, Cloud Build, Cloud Logging, Cloud Monitoring, IAM, Secret Manager, and autoscaling mechanisms made it evident that modern cloud platforms provide comprehensive tools to manage enterprise applications without the need to maintain physical infrastructure or traditional servers.

Another relevant takeaway was the importance of managed databases in cloud-native architectures. Utilizing Cloud SQL, PostgreSQL, and MySQL allowed us to analyze enterprise persistence strategies, connection management, migrations, and integration mechanisms within Node.js applications. Separating business logic from data storage proved to be an essential requirement for building scalable and long-term maintainable solutions.

The analysis of modular architectures and microservices showed how modern systems evolve from monolithic applications into structures composed of independent services. Comparing different deployment models, repositories, and platforms provided a clear understanding of the

advantages of modularity, component reuse, deployment independence, and the capacity for the progressive growth of a technological solution.

Regarding security, it was demonstrated that the protection of credentials, secrets, API keys, and service accounts must be integrated into the design from the earliest stages of development. Incorporating Secret Manager, IAM policies, OAuth, JWT, and the principle of least privilege made it clear that security is not an additional layer, but a cross-cutting component of any modern cloud architecture.

Likewise, observability proved to be a critical element for operating distributed systems. Centralized logging mechanisms, monitoring, metrics, alerts, and recovery strategies make it possible to detect failures, analyze behaviors, and guarantee the operational continuity of services deployed in production. Practical experience demonstrated that the ability to troubleshoot and diagnose incidents is just as important as the ability to develop software.

The study of scalability and distributed architectures provided a deep understanding of how services such as Load Balancing, Autoscaling, Cloud CDN, Failover, Disaster Recovery, and multi-region deployments contribute to building resilient applications capable of enduring significant workload increases without impacting availability or user experience. These technologies form the operational core of current enterprise systems.

The culmination of this book materialized in the Git-Hub-API-Cloud comprehensive project, where all previously developed concepts converged into a functional solution deployed on Google Cloud. Code migration and refactoring, PostgreSQL integration, Secret Manager implementation, Docker image building, publishing to Artifact Registry, deploying to Cloud Run, incorporating AI-powered auditing, and final system validation made it possible to verify the applicability of the studied principles in a real-world environment.

Final testing confirmed that the platform operates stably across both desktop and mobile devices. The multilingual translation system, the deployed APIs, database integration, secure credential management, visual components, static resources, and the Service Worker-based installable application all responded successfully, validating the consistency of the entire implemented architecture.

As a general conclusion, this book demonstrates that modern application development can no longer be understood through coding alone. Building enterprise solutions requires integrating knowledge of containerization, DevOps, serverless computing, cloud databases, security, observability, artificial intelligence, and scalability. The architecture resulting from the Git-Hub-API-Cloud project stands as a fully operational, cloud-native platform—ready to incorporate new services, new APIs, and future AI-driven capabilities while maintaining the principles of modularity, security, maintainability, and sustainable growth that define contemporary technological architectures.

11. REFERENCES

1. "Designing Data-Intensive Applications" by Martin Kleppmann
2. "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin
3. "Building Microservices" by Sam Newman
4. "Docker Deep Dive" by Nigel Poulton
5. "Learning AWS" (or an equivalent for GCP, if found, though the concepts are similar) by Adrian Cantril
6. "Node.js Design Patterns" by Mario Casciaro and Luciano Mammino
7. Official Google Cloud Platform (GCP) Documentation
8. Official Git Documentation and GitHub Docs
9. Official Docker Documentation
1. Mozilla Developer Network (MDN) Web Docs
10. Stack Overflow
11. Software Architecture and DevOps sites and blogs (e.g., Martin Fowler's Blog, InfoQ, The New Stack)
12. "Artificial Intelligence: A Guide for Thinking Humans" – Melanie Mitchell (2019)
13. Explains the fundamentals of artificial intelligence and its application in machine learning models, including computer vision and NLP.

12. MOTIVATIONAL QUOTES

"Success is the sum of small efforts repeated day after day."
Robert Collier

"The only limit to your fulfillment tomorrow will be your doubts today."
Franklin D. Roosevelt

"It doesn't matter how slow you go as long as you don't stop."
Confucius

"The path to success and greatness is to develop what you already have."
ZigZiglar

"The secret of success lies in knowing more than others."
Aristotle Onassis

"It's not about how many times you fall, but how many times you get up."
Vince Lombardi

"Success is not the key to happiness. Happiness is the key to success. If you love what you do, you will be successful."
Albert Schweitzer

"The only way to do great work is to love what you do."
Steve Jobs

"The biggest risk is not taking any risks. In a world that's changing so fast, the only strategy that's guaranteed to fail is not taking risks."
Mark Zuckerberg

"Motivation drives us to begin, and habit keeps us going."
Jim Ryun

"It's not enough to tell the truth; you have to prove that you have it."

Baltasar Gracián

If you can't fly, run; if you can't run, walk; if you can't walk, crawl, but keep moving toward your goal."
Martin Luther King Jr

“No matter your physical age, you can always motivate your mind to face tomorrow.”
From the Author

OTHER BOOKS BY THE AUTHOR

If you would like to continue exploring the *Programming Step by Step and More* series, you can access all related volumes, updates, and additional books available on Amazon through the QR code below.

All books in this series are available on Amazon.

Scan the QR code to discover more:

13. ABOUT THE AUTHOR

After several decades dedicated to improving the energy efficiency of heating and cooling systems and equipment, I faced a major life transition: retirement. Upon retiring, I felt disoriented and without a clear sense of purpose, which affected my emotional well-being. To overcome this situation, I sought a new activity that would keep me active and motivated.

I chose to share the knowledge and experience gained over more than four decades as an engineer through this project, publishing articles, tools, and instructional materials across multiple disciplines.
These self-taught technical skills, including web programming, were developed through continuous learning and applied to optimize projects, finance, and business processes.

From my own experience, web programming is a powerful tool for solving complex problems, not only in industrial environments but also in technical services and everyday life.

My renewed perspective for the future, centered on the ongoing digital transformation, has been essential in keeping my mind active, staying up to date, and learning new technologies through hands-on experience. In this context of rapid and constant change, my vision expands toward a horizon where artificial intelligence not only optimizes processes but also redefines industries and creates opportunities that were once unimaginable.

This instructional work highlights the importance of programming languages as essential tools in the modern world. It is designed to support individuals of all ages in developing their professional skills and achieving their career goals. It provides valuable resources both for those beginning their journey and for those seeking new ways to apply their experience.

The Author

ROGRAMMING STEP BY STEP AND MORE ®

An Instructive Guide Book 9 - Cloud-Native Platform Engineering Security, Observability, Scalability, and Microservices - – Capstone ProjectMiami, Florida, USA, 2026

Translated from Spanish by René F. Ruano Domínguez

Originally published in Spanish as
"PROGRAMANDO PASO A PASO Y MÁS – Serie"
Instructivo Libro 9 - Ingeniería de Plataformas Cloud-Native
Seguridad, Observabilidad, Escalabilidad y Microservicios – Proyecto Integrador

Publisher: Corporate Luxury Group, LLC