Cloud Native Architecture

Efficiently moving legacy applications and monoliths to microservices and Kubernetes

Fernando Harris

www.bpbonline.com

First Edition 2024

ISBN: 978-93-55516-770

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

www.bpbonline.com

Dedicated to

My parents and sister:

Stella Clarissa Harris Pedro Francisco

Fernando Duarte Pedro Francisco

Carol Miriam Harris Pedro Francisco

and

My wife ***Paz Ramos*** *and my sons,* ***Sebastián*** *and* ***Fernando***

About the Author

Fernando Harris is a Mozambican-Portuguese solution architect based in Spain. Expert in API management, service-oriented-architectures, event-driven-architectures and microservices, he supported dozens of organisations implementing Agile and DevOps best practices to design and implement cloud-native products and modernise existing legacy systems. He holds an MSc in Information Systems Management from the Technical University of Lisbon (ISCTE), and a BSc. in Computer Science and Business from the Polytechnic Institute of Coimbra (ISCAC), both in Portugal.

About the Reviewer

Phil Wilkins has spent over 30 years in the software industry with a breadth of experience in different businesses and environments, from multinationals to software startups and consumer organizations. He started as a developer and has worked through technical and development leadership roles. Today Phil's interest and focus is on observability, APIs and cloud technologies and techniques. Phil works for Oracle as a Cloud Architect & Evangelist.

Phil has authored books on Fluent Bit and Fluentd and co-authoring on API and Integration development. He has also had several articles published in technical journals and is an active blogger. When not writing, Phil explores new tech or presents at conferences physically and virtually around the world - from DeveloperWeek to JAXLondon.

Acknowledgement

Writing this book was a pleasant challenge which I very humbly accepted. Hopefully, it will be as helpful for the readers as it was for me. I have written it in the good spirit of open-source communities: to share what I learn.

My first acknowledgment goes to my family, especially my wife Paz for her daily support, for her infinite patience and encouraging me when I thought about giving up.

Thanks to Phil Wilkins, whose ideas about music are as interesting as his writings about software engineering. At some point, his technical revision looked like a mentorship process. His vast experience in real projects and products is impressive. As a veteran book author, his contributions, suggestions, and amendments were key to finishing the book and a privilege to have.

Thanks to Gonçalo Alves, an experienced information systems expert, one of my best friends, and my former colleague who offered valuable insights and opinions on the presented subjects.

Thanks to Ewan Slater, my colleague, for the timeless discussions around cloud native and for being the person who almost 8 years ago convinced me to embrace this area.

Thanks to all the authors I have cited, books and articles I have read, blog posts and websites I consulted and open-source code that I used to explore important concepts.

Finally, I would like to express my gratitude towards the staff at BPB Publications for the exceptional support and constant encouragement. This mission would have been impossible without their assistance.

Preface

This book explains in 9 chapters how to plan, manage, build, and run applications such as microservices in an agnostic, scalable and highly available cloud native runtime such as Kubernetes. This is done by effectively applying DevOps principles through the tactical use of **CNCF** (**Cloud Native Computing Foundation**) tools.

It covers cloud native history and the business drivers we must understand to adopt this paradigm. It sets a pragmatic definition of cloud native, based on five principles: open-source, container-based loosely coupled systems, ubiquitous integration, operational benefits, and DevOps adoption. The book also proposes a framework to achieve cloud native success that starts with a cultural shift and goes through the interaction between teams, people, ethics, and skills with key organizational processes based on Agile, Scrum, Domain Driven Design, API first, DevOps, Observability and Chaos Engineering.

This framework presents a deep technical section as well. In it, we explore Kubernetes architecture, topology and key components which will let us learn how to design, build, and deploy evolutionary cloud native monoliths and microservices based on the Twelve-Factor App principles and Kubernetes best practices. The book also covers important aspects of automating the deployment of cloud native applications with real examples configured with Jenkins CI/CD pipelines.

A special end chapter is dedicated to Kubernetes security and how to establish a secure perimeter for the cluster. We will also explore what is needed to define and manage cloud native applications' security requirements in build and runtime.

Chapter 1: History and Business Drivers - explaining the business and organizational needs behind the history of cloud native.

Chapter 2: Five Different Cloud Native Perspectives - explore five angles to understand cloud native: open source, container-based loosely coupled systems, operational benefits, ubiquitous integration and DevOps.

Chapter 3: The Cultural Shift Introducing a Framework to Succeed - propose a framework to help achieve cloud native success. Try to answer why organizations should invest in cloud native. Explore the cultural change most cloud native successful organizations face and the need to develop a culture where performance improvements can be measured.

Chapter 4: People: Who is Doing What - study the attributes, which are important on individual and collective dimensions, and constitute a baseline to build an engaged team. The importance of ethics in cloud native and how Agile influences individuals and their interactions when setting up self-organizing and cross-functional teams.

Chapter 5: Processes: How Should We Do It - discuss in detail Scrum and Agile, Domain driven design, API first, DevOps, Observability and Chaos engineering and their impacts on the journey for cloud native success.

Chapter 6: Technology: Where Are We Running It - discuss the provisioning of a cloud native runtime. Define what Kubernetes is, its needs, and what real problems can it solve. Learning Kubernetes architecture, fundamentals, and key concepts

Chapter 7: Technology: What Are We Building - how a modular monolith **-** that follows the Twelve-Factor App principles and is deployed on a runtime like Kubernetes - might be a valid approach to starting a cloud native project when influenced by the evolutionary architecture attributes. Discuss modularity and incremental change. Learn how to use Jenkins with real examples to set up automation for CI/CD.

Chapter 8: Technology: Transition from Monolith to Microservices - discuss microservices key attributes and anti-patterns. Learn to decide when and how to adopt microservices, identify the impacts and define pros and cons. Execute a transition from a monolith to microservices with a real application.

Chapter 9: Technology: Addressing Kubernetes Security - discuss security at the cluster, pod, and container levels. For each level, address the main concerns and solutions. Learn how to restrict access to the kube-apiserver, Kubernetes RBAC and leverage existing enterprise security controls at the cluster limits. Discuss security requirements inside the cluster in terms of component recommended configurations and communication best practices. Discuss security at the application level.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/3af63c

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Cloud-Native-Architecture.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1

History and Business Drivers

Introduction

For the **Cloud Native Computing Foundation** (**CNCF**)[1] cloud native encompasses all *techniques such as containers, service meshes, microservices, immutable infrastructure and declarative APIs that enable loosely coupled systems to be resilient, manageable, and observable. Combined with robust automation they allow engineers to make critical and high-impact changes frequently and predictably with minimum toil for the business*. Despite being a purely technical definition, it has business and historical reasons behind it. In this first chapter, we will start by taking a quick look at cloud native history and business drivers and how information technology evolved to the cloud native paradigm. Understanding the business and organizational drivers behind the history of cloud native is key to understanding its definitions and why organizations are becoming cloud native enterprises.

Structure

In this chapter, we are going to discuss the following topics regarding cloud native:

- Business and organizational drivers
- Relationship with distributed architectures
- History: From virtualization to containerization

[1] Cloud Native Computing Foundation or CNCF.

Objectives

In this chapter, you will learn why enterprises are becoming cloud native and the historical, business, and organizational reasons behind that transformation. You also learn about the relevance of studying distributed architecture and why cloud native differentiators are more tangible in that context. Finally, you will learn about the differences between virtualization and containers and why the latter made possible the advent of cloud native technology.

Business and organizational drivers

The typical cloud native book will start by telling you what cloud native is. As of today, this might seem unnecessary. Who does not know what a container is? Or, for what Kubernetes should be used? Nonetheless, there are many different definitions for cloud native. Some products or services are sometimes listed as cloud native depending on the context they are being discussed. In this book, to define a tangible scope for the readers, we will refrain from judging whether a specific product is cloud native or not and just assume those that are certified[2] by the CNCF as such.

Simplistically, in a cloud native environment, development produces more releases with minimal toil, and operations get high availability, scalability, and resilience as easy-to-use commodities. In the end, it is the organization and its information systems that get the real benefits. In sum, is about speed, agility, and efficiency. Cloud native is not necessarily about cloud[3] (though cloud computing helps a lot), but it is about resources - or perhaps scarce resources - and how agile is an organization in managing memory, RAM, or storage to respond to changes in the business needs.

It is a fact that cloud native is helping organizations in the creation of value for their customers by changing the way products and services are planned, produced, and delivered. This is effectively achieved by leading a sort of revolution in software development and operations, facilitated by - an early - adoption of DevOps principles. Effective ways to successfully do that is what this book is about.

The strategic significance of cloud native for companies is undeniable. In a 2019 survey of 2500 developers around the world[13], almost half responded that they are training to develop mission-critical applications with cloud native technology such as microservices, containers, container orchestration frameworks, and serverless functions for their companies. Companies that understood the importance to adopt cloud native will present more probability to keep IT as a source of competitive advantage and will eventually lead their markets and industries. Companies that fail to understand this will need to accept the rules dictated by the leading cloud native competitors and find themselves

[2] CNCF Landscape [*https://landscape.cncf.io/*] categorizes all certified CNCF open-source projects in different technical functions or domains.

[3] Cloud native doesn't need to be in the cloud the same way microservices don't need to be small or Serverless doesn't mean there are no servers involved!

at a competitive disadvantage[1]. This applies whether we are talking about start-ups, or experienced companies already playing a role in their markets and industries[4]. And the reason is the demand to have more digital products, services, and operational processes on the scale required to face competition. Cloud native enterprises are software builders as opposed to consumers of off-the-shelf software. This transition requires a shift in terms of cultural, operational, and technical values [13]. It is clear that cloud native is a source of competitive advantage, and it does that by supporting the company's value chain in a very agile way. A company value chain (as depicted in *Figure 1.1*) is a system of interdependent activities which are connected by linkages supporting its critical processes to develop products and services[2]. When these interconnected activities are internal, they represent linkages between different functions and domains inside the organization. When external they might represent linkages between the company and providers or linkages between the company and customers that is, **business-to-business** (**B2B**) or **business-to-consumers** (**B2C**)[5] commercial relationships:

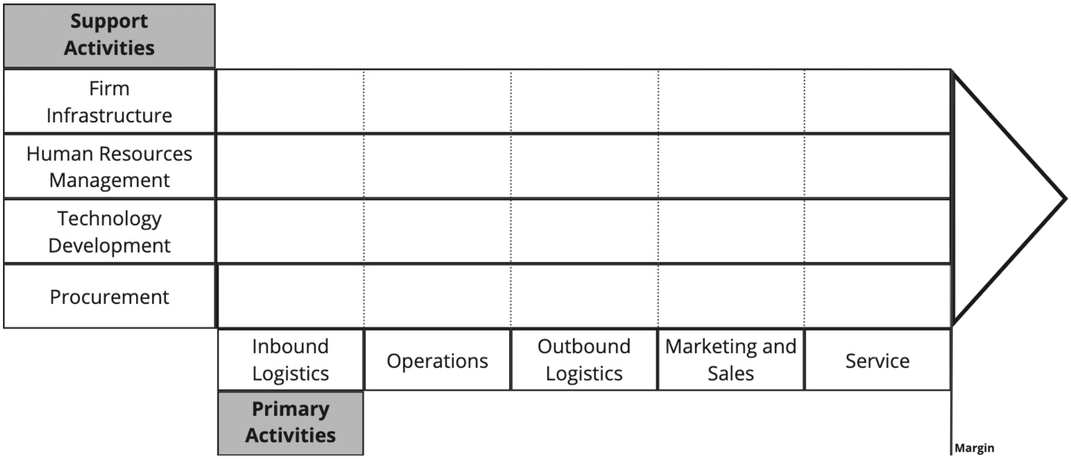


Figure 1.1: *Porter's linkages and value chain*

This value chain is evident when we look at the ERP systems and their history for example. An **enterprise resource planning** system or **ERP** is a software platform supporting the seamless integration of different business functions and domains - for example, supply chain, human resources, customer data, accounting and so on - by connecting information flows using process or data integration to support specific business processes such as hire-to-retire or order-to-cash[3]. These different functions or modules typically run against a single database. The ERP promised to cover all functions within the organization's different domains. In some way, it succeeded as of today the ERPs still in the market keep improving and adding functional domains, responding well to internal and external forces and demands within the organization. However, the modern ERP no longer expects

[4] There are many famous examples of this such as Spotify, Netflix, Amazon.

[5] Business-to-business (B2B) and business-to-consumer(B2C) are acronyms that represent commercial activities based on transactions between companies or organizations (B2B), or transactions between companies and consumers, or individual customers (B2C).

to respond to all integration needs. External forces such as those generated in a B2B or B2C context brought different challenges, and a need to learn how to integrate dynamically with other systems. This need was coming from new constant market demands, different domain functions with new projects, new services and new products, companies merging and changes in the relationship between providers and customers (EDI, Just-In-Time).

The space of the ERP was challenged by integration technologies such as EAI or Enterprise Application Integration. ERPs and EAI descendants still live side by side in the organizations and the diversity of requirements they brought including the different integration needs they covered, ultimately helped consolidate distributed architecture as a de facto standard in many organizations' integrated systems.[4] We will address distributed architecture and its close relationship with cloud native technology in more detail in this chapter.

The idea is that an organization's business changes dictate the evolution of integration architectures. Mainframes and ERPs fed for a while a dream of homogenous integration which was soon revealed to be very difficult to achieve. EAI and later SOA or Service-Oriented-Architecture, came in to solve these limitations by consolidating the principle of heterogeneous integration, based on "industry-defined open standards" and interoperability between different applications and systems with multiple components based on XML, SOAP, and Webservices.

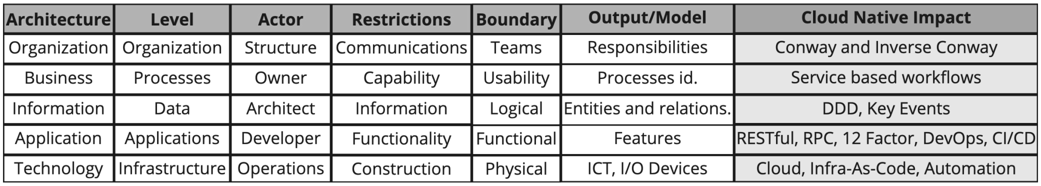
Richards and Ford resumed this phenomenon with a simple and interesting analogy: "*Architecture styles, like art movements, must be understood in the context of the era in which they evolved*"[4]. In the same way Impressionism, Realism or Cubism left their inspiration on societies, so did the IT "art movements" leave their mark on organizations' enterprise architectures. A great example of this is evident in microservice's many patterns and principles created for EAI, SOA and **Event-Driven Architectures** (**EDA**)[5]. The point is that any art movement or tech trend is temporary and only some parts and core principles will survive the initial hype and the test of time. It does not matter what technology or architecture approach you use to connect these Porter linkages; technology will eventually become outdated. The focus of the architect should be on how to manage these linkages and how to manage their inevitable change. We know that the only constant is change itself, and as architects, we should plan the system and the organization for it[6].

If B2B and B2C were generating a huge demand for change, new patterns such as IOT and D2C[6] are demanding enterprises to do it at an even faster and unprecedented scale. Cloud native techniques and principles seem to accommodate this constant need to change in systems and organizations way better than past technology – and at a bigger scale and higher velocity - as its core attributes have been thought to support, collaborate, and even instigate business change and digital transformation.

[6] D2C stands for Device to Cloud and IOT stand for Internet Of Things, referring to technologies associated with the connection of devices and machines between them and the cloud.

Enterprise architecture

In large organizations, enterprise architecture is a well-known tool to manage complexity and change. Though its correct application is not always visible, it is a discipline which is present in many industries. One can use it to understand and build the enterprise itself by defining what it encompasses in terms of business, information, applications, and infrastructure scope and how each of these levels and actors relates and behaves when managing the information flows to support the organization's strategic goals. Information Architecture is a critical level or perspective of Enterprise Architecture as it states which data is fundamental for the organization in terms of business entities and inputs/outputs for different information flows[7]. In *Figure 1.2* we can see these levels, perspectives, actors, restrictions, boundaries, and models represented. We can also see how it relates in generic terms with cloud native concepts. Most of these concepts we will address in detail in the following chapters.



Architecture	Level	Actor	Restrictions	Boundary	Output/Model	Cloud Native Impact
Organization	Organization	Structure	Communications	Teams	Responsibilities	Conway and Inverse Conway
Business	Processes	Owner	Capability	Usability	Processes id.	Service based workflows
Information	Data	Architect	Information	Logical	Entities and relations.	DDD, Key Events
Application	Applications	Developer	Functionality	Functional	Features	RESTful, RPC, 12 Factor, DevOps, CI/CD
Technology	Infrastructure	Operations	Construction	Physical	ICT, I/O Devices	Cloud, Infra-As-Code, Automation

Figure 1.2: *Enterprise architecture and cloud native impacts*

Integration architecture

Different stakeholders in different domains have different information needs. The Integration Architecture is an abstraction that represents what is needed to assure that the above-mentioned information flows circulate between different business domains and linkages without any siloes or boundaries. It tries to explain and detail these flows in terms of interoperability needs between different applications and different building blocks. These applications may be consumer applications, the client services that deliver information for the end user, provider applications, the business services which receive requests from the clients and provide responses from the end targets (for example a server, a database or a Rest API) and brokering applications, the middleware that manages the relationship between consumers and providers. For TOGAF this foundation is called the **integrated information infrastructure – Reference Model (III-RM)**[7] and is part of the application architecture. The vision portrayed by this reference model can be extremely valid to understand how the value chain and linkages relate to integration building blocks inside an organization. Understanding this can be difficult not only because integration technologies can be complex to master, but also because the organization has its own "organic" complexity reflected by its communication structure.

The Conway Law[7] states that *any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure*[8]. As

[7] James Lewis and Martin Fowler coined a concept called the 'Inverse Conway Maneuver which recommends optimizing the team and organizational structure to create some sort of structural parity between the technology and business architectures to achieve the desired product.[16]

exemplified in *Figure 1.3*, the more hierarchical the structure is, the more complicated will be the integration and the effort to integrate:

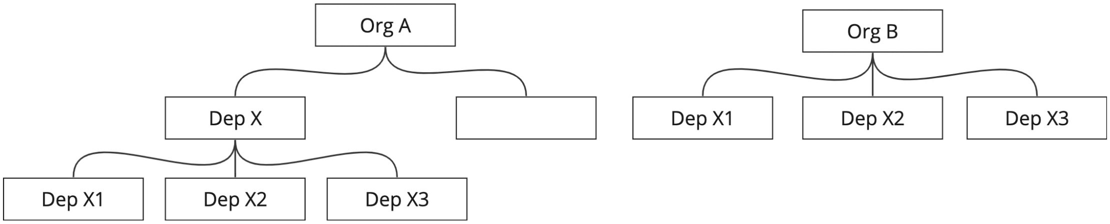


Figure 1.3: *Organization Structure: A – Hierarchical versus B – Flat*

This complexity might help us understand the inherent difficulty in integrating programs into a system. It was normal to measure software complexity by the number of lines of code, or other indicators related to the application itself and ignoring the difficulties related to the need to integrate and make that application communicate with others as part of a system. Keen and Gambino[9] revisiting Brooks, built important evidence around the indicator that *if X is the effort required to write and test a program, 3X is needed to make it into a program product and 9X to integrate it into a system product*, as depicted in *Figure 1.4*:

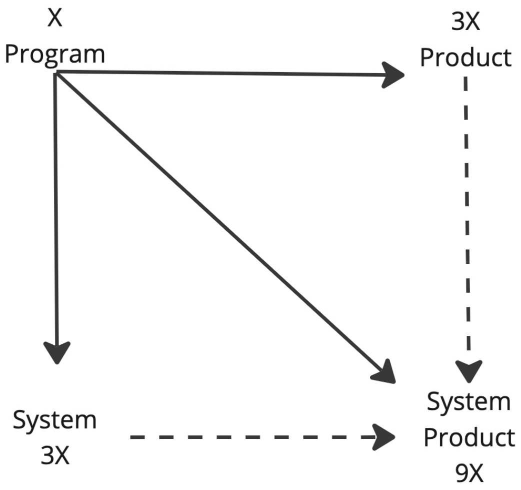


Figure 1.4: *Brook's assessment of relative programming effort*[9]

According to these authors, *Integrating a program into a system requires substantial testing of linkages, and often additional code must be written to ensure consistency*. Thus, integrating might take - at least - 9X more effort than building an isolated program. This additional code to ensure consistency is needed to make sure that the new program fits the existing system. The existing system, – in theory – translates the organization's communication distributed structure with all its complexities. Today, to measure these complexities in the design phase, we define the different modules' coupling and cohesion metrics[8]. These

[8] A modular system with high cohesion and low coupling is usually considered to be well designed.

indicators will show the degree of interdependence between different modules and the level of functional cohesion inside each one of them.

Cloud-Native and distributed architectures

It does not matter where you look in terms of integrating linkages and value chains. Whether the organization structure is flat or hierarchical, or you are considering developing and connecting an application into new or existing systems or integrating internal or external linkages, regardless of the technical approach, you will certainly produce some sort of distributed system to consistently represent the organization's communication structure. This has been shown by the evolution of different architectural styles for distributed systems such as those based on service orientation, event-driven, or microservices[5], backed by principles such as API First – in which the tactical or strategic definition of the API comes before everything else as a proper contract to follow between parties– and techniques such as those based on asynchronous messaging in which the dependency between producers and subscribers of events is very low, allowing the integration of loosely coupled components through the utilization of queues and topics. Though they have key differences and represent different movements, in general, we are discussing complementary approaches to implementing distributed systems[4]. Simply put, we can quickly define some common major pros and cons of these architectures. They are typically hard to test, and integration testing is very challenging. They might result in complex solutions as they typically address complex use cases. Last, they can be expensive, whether because you are acquiring an SOA platform made available by a vendor, or because you will need senior engineers and senior architects in your team to implement EDA or microservices patterns and you might not always find them available in the market. To compensate for these disadvantages, this type of architecture can be very agile. To some extent, the design tends to present more loosely coupled modules which can give the team managing it more autonomy and less risky decisions to take when deploying in production. The performance can vary as it is not its strongest attribute. On the other hand, scalability is probably its best quality. Of course, this list presents a mix of pros and cons where some of which can be easier to track in microservices than in SOA for example. Microservices should be always deployed independently and not share the database while services in SOA are very constrained to achieve the same, not only because typically services will live in different containers but inside the same application server and share the same database, but also because they will keep some sort of tight coupling with the Service Bus[9].[7] Despite some key differences, we can in general assume that most of the pros and cons we mentioned are present in distributed architectures[5] with different orders of magnitude and importance, as shown in the following figure:

[9] It's possible to have multiple application servers with different services deployed. The Service Bus is typically a monolith, though cloud native is also changing that.