Cloud Native Software Engineering

Brian Mitchell
College of Computing and Informatics
Drexel University, Philadelphia, PA, USA
bmitchell@drexel.edu

Abstract

The adoption of cloud computing has been growing in popularity since its inception in the early 2000's with all signs pointing to continued growth over the remainder of the decade [1]. While there is a significant research activity in many areas of cloud computing, we see little attention being paid to advancing software engineering practices needed to support the next generation of cloud native applications. By cloud native, we mean software that is designed and built specifically for deployment to a modern cloud platform. This paper will focus on the landscape of Cloud Native Software Engineering from a practitioners standpoint, and identify opportunities that we think should be investigated by the software engineering research community.

1. Introduction and Context

Delivering managed computing services via hosted infrastructure stared in the late 1990's with the introduction of the Software as a Service (SaaS) model. One of the early pioneers of this model is Salesforce.com, launched in 1999 [2]. Unlike other companies that licensed software that deployed to on premise equipment, Salesforce provides a pay-as-you-go SaaS subscription model. In this model, they manage all of the software and compute, you pay a monthly per-user charge that can access the solution from any device at any time.

While SaaS solutions marked the start of shifting software license spend to usage-based spend, cloud computing as we know it today can be attributed to the launch of AWS (Amazon) [3] in early 2006, with Azure (Microsoft) [4] and GCP (Google) [5] following in 2008. The primary early adopters of cloud computing were technology companies that innovated patterns, practices, and open sourced many tools and frameworks that have become best practices for running resilient and scalable business services in the public cloud. It's probably been less than 10 years, but now we see acceptance of cloud computing in companies of all sizes across many different industries.

Larry Wall, the creator of Perl, is credited with a quote that has become a popular software engineering meme – *"Good. Fast. Cheap. Pick two."*. There is some intuitive merit to this insight given software engineering is a rooted in making informed tradeoffs. For example, it would not be hard to argue that in order to move faster and build things cheaper, you will need to compromise on software features and/or software quality. Using the utility of the cloud, coupled with modern cloud computing tooling, one can now argue that you can build better software faster and cheaper. It's not that Larry Wall's insights were incorrect, but we can now have the technologies and practices to redefine *good* in terms of *fast* using the cloud. When computing components are deployed to the cloud, the simplest way (and thus the most popular way) to do this is via automation technologies [4,6–8]. The *automate everything* practice embraced by cloud computing not only allows deployments to be fast, but it also favors ephemeral computing components. These components by their nature are easier to test [9] and can be started, stopped, paused, or replaced at any time.

This combination of capabilities enables software engineers to rapidly deploy software to a known state at any time. With these building blocks new well-tested features can be quickly and consistently rolled out to users in very small batches. Goodness of the solution can now be validated via feedback from users, either directly, or via monitoring and instrumentation of their behavior. These cloud enabled capabilities have the potential to advanced software engineering practices in many ways, but transforming these practices across the entire community will be challenging. We think this represents a significant opportunity for the software engineering field given the likelihood that most industrial systems moving forward will be deployed on cloud runtimes[1]. Specifically:

[1] By cloud runtime we include public, private and hybrid cloud

1. Helping Software Engineers manage the expanded cognitive required to design, build and deploy at scale cloud applications. We will discuss this throughout the remaining sections of this paper.
2. Identify opportunities to accelerate and scale software engineering skillsets needed to deploy a broader suite of applications the cloud. Many industries will strive to move beyond deploying externally facing web and mobile applications to the cloud using their top engineers. This will require developing new skills for the broader engineering organization as more of their core business processing moves to the cloud.
3. Investigate how software engineering and computer science education can expand to address the demands of industry to create new, and retooling existing software engineers for the cloud.[2] Most cloud-proficient software engineers build their skillsets on the job versus in formalized academic programs.
4. Understand software engineering needs for new architectures enabled by the cloud. For example, IoT and smart devices that run at the edge increase the complexity of software engineering given the increasing distributed nature of these solutions. These challenges will be discussed in a later section of this paper

In the next section we start by providing a definition for cloud native computing. Subsequent sections will build on trends in cloud native computing that we think will impact software engineering practices. Throughout this paper, by cloud native, we are referring to systems designed specifically to favor managed cloud platform services (PaaS/Faas) [10], and not systems that are *lifted and shifted* [11] from an on premise virtual machine to a virtual machine that runs in the cloud (IaaS).

2. What is Cloud Native Computing?

Before we explore the software engineering landscape for the cloud, we need to address exactly what we mean by cloud native computing. According to the Cloud Native Computing Foundation (CNCF) [12] *"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds."*. Amazon's definition is *"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds"*. Google offers the definition *"Cloud native means adapting to the many new possibilitiesbut very different set of architectural constraintsoffered by the cloud compared to traditional on-premises infrastructure."*. The primary theme in these definitions centers around the role that technologies play in enabling the creation of cloud native applications. They also don't clearly define "Cloud Native", which we consider any application that is specifically designed to be deployed to a cloud platform.

We think a better definition of cloud native computing that focuses more on software engineering is *"Cloud native applications are well architected systems, that are "container" packaged, and dynamically managed"*. Specifically:

Well Architected Systems - By this we mean systems that adhere not only to established software engineering best practices but also embrace specific functional and non-functional capabilities offered by the cloud. For example, how are the computing components identified, how are they work with each other, how are security requirements met, how is the system designed for resiliency and scale?

Container Packaged - The term *container* is overloaded in the cloud computing terminology landscape. In many places its equated to a standardized package that is managed by Docker [13] technologies - aka "a docker container". We take a more generic view of container packaging. Specifically, we think container packaging is a mechanism to package and deploy code that is ephemeral, can operate across a variety of different hardware architectures (e.g., Intel, ARM, etc), and at runtime is supervised. Supervision includes full lifecycle management associated with version identification, startup, shutdown, health checks, and monitoring. Examples of container packaging and supervision include Docker, Docker Compose, Kubernetes, and serverless [14]. We also include in this category the emerging popularity of using server-side web assembly [15, 16] as a way to package and deploy cloud native application services.

Dynamically Managed - One interesting conceptual model for cloud computing is to consider the cloud as a large, highly distributed, special purpose operating system. Just like any operating system, there are a number of resource like storage, compute, network and security services that are needed by applications. The job of an operating system is to dynamically manage and optimize the allocation of these resources to the realtime computing demand on the system. When done well, every process being managed by the OS will perceive that

infrastructure

[2] We will talk about cloud certifications later, but they are targeted towards using the services of a cloud provider, not designing cloud native applications

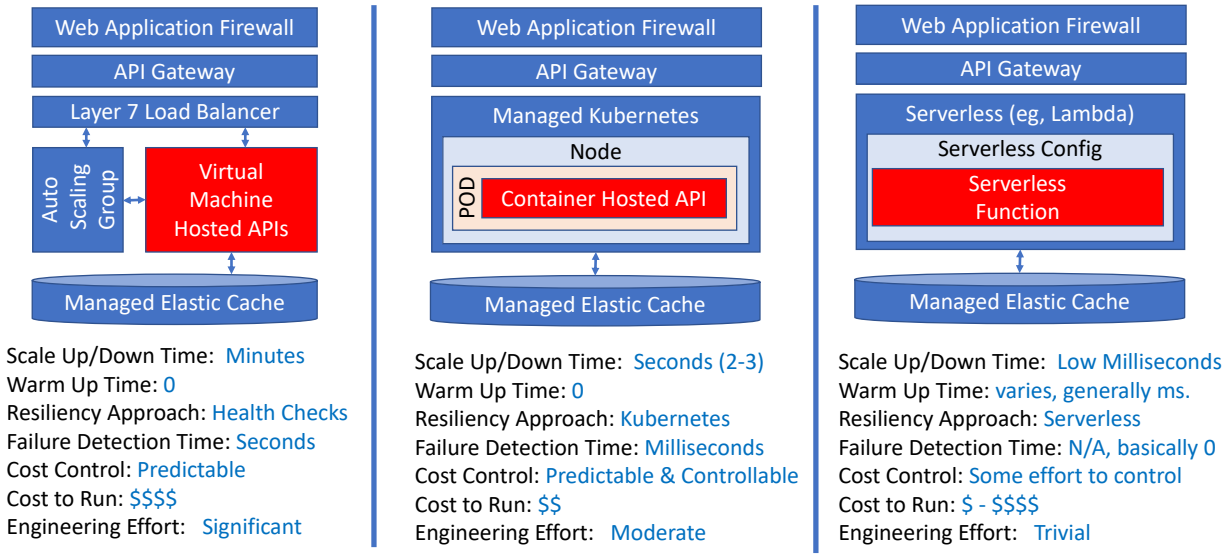

Figure 1. Cloud Native Architectural Tradeoffs for a Hypothetical Suite of APIs

it has access to the resources it needs, when it needs it. In a similar context, a cloud service provider, via Application Programming Interfaces (APIs), provides and manages resources to cloud native applications dynamically. What is generally different is that an OS manages physical resources on a system, whereas cloud resources are generally virtualized and distributed, and to a large degree fault-tolerant. For example, block storage that supports virtual machine reads and writes are automatically replicated across servers within an availability zone. Outside of initial configuration, the user does not worry about how durability is provided given its dynamically managed by the cloud service provider. Other examples include using auto scaling of virtual machines with health checks, or more advanced services like kubernetes [17] that can scale up or down dynamically based on demand. Function as a service (FaaS) solution's take this a step further by configuring what code should be run when a certain event happens.

Now that we have provided a definition, we describe next a number of interesting software engineering problems that warrant investigation.

Managing cloud native technical assets. In 2011 Adam Wiggins authored a set of technical principals that enable software engineers to create, manage and release code in support of cloud native applications. These principals were branded "The Twelve-Factor App" [18]. Over the years they were updated and revalidated [19,20], but consistently hold up as recommended engineering practices. One of the key challenges is that while none of these practices is overly complex, they require mastery and discipline. Also, existing standards enforced by enterprises might act as blockers to some or all of these practices. For example, some organizations might not have comfort or necessary controls to ensure that all deploys, to all environments are driven through the source code control system.

Identifying an appropriate cloud native software architecture. The technical principals associated with 12 factor apps is a good start, however, these only focus on how to manage cloud native technical assets. Good cloud native architectures are generally architected as a suite of horizontally composable components. This model introduces several interesting challenges for cloud native software engineers that must be addressed. What are these components? How do identify them? What will guide how they should be built? We think some of the newer concepts around app meshes [21] and data meshes [22] provide a good start for shaping the overall architecture. As far as identifying the architecture components themselves, we think concepts from Domain Driven Design (DDD) [23, 24] can be refreshed to support this activity.

Upskilling Software Engineers on the use Quality Attributes to make informed technology tradeoff decisions. Most cloud providers offer many different technology choices to create cloud native components. Applying discipline around the desired quality attributes should guide making technology stack decisions. For example, Figure1 shows three different ways to deploy cloud native APIs along with some quality attributes. It should be noted, that the values of these quality attributes will change for different APIs, the figure assumes these hypothetical APIs are very light weight,

event triggered, and manage their state via a clustered cache. Thus for the scenario shown in the figure, the VM option does not make a lot of sense given the high cost, large scale up/down times, and complex engineering effort. The container option in the middle and the serverless option on the right both seem like good options. The ultimate decision will be driven by the desire to have a high degree of control over cost, and the nature of the workload. For example, if the expected traffic has massive near realtime spikes, a serverless solution might be preferred. If the workload is not event-driven, or has many runtime dependencies such as database connections, then a container based solution might be a better choice. As we move more towards computing at the edge, containers might not be possible since they depend on Linux, so emerging lighter weight alternatives such as WebAssembly with WASI [25] might be a good choice. Cloud native software engineers must be able to make these types of choices and resist overstandardizing based on personal or organizational preference on the best runtime for a cloud native component.

Organizing teams for cloud native success. In 1967, Mel Conway published a paper called "How Do Committees Invent" - Fred Brooks cited this paper in the Mythical Man Month [26] calling it Conways law [27]. Conways law states *"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure"*. In the early days of Amazon, Jeff Bazos introduced the idea of the "2 pizza team rule" [28] where a team size should be no larger than can be fed by two pizzas. The basic ideas are rooted in concepts that productive teams should be small, and independent. This aligns nicely with cloud native concepts in that one way to ensure that components are independent and interoperate only via their published interfaces, that teams are also organized this way. Over the years various organizational models and changes have been debated. *Full-stack teams* bring together front-end, back-end, testing and infrastructure professionals to a common team where they have full responsibility over their technical assets. *Shifting left* along with the emergence of *DevOps* brings a testing and automation focus to teams that allows them to increase productivity. One problem is that while these concepts work well for driving individual team productivity, they are difficult to scale to larger organizations. Several companies have also published their attempts to scale their practices. One popular model was published by Spotify [29], where "Squads" represent full-stack teams, but they also introduce concepts like "Tribes" for coordinating squads, and "Guilds" to address cross-cutting

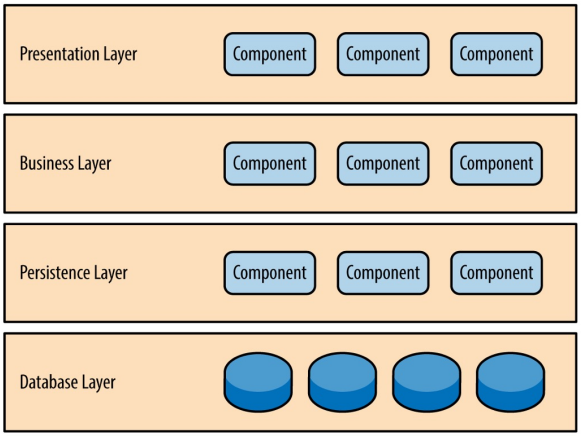

Figure 2. Layered Software Architecture

technical concerns. Commercial models have also been created to drive organizational changes to support cloud native architectures. One such example is the SAFe [30] framework, which is interesting in that it favors prescriptive organization and process rigor over streamlining software engineering practices in order to increase scale. We think there are some interesting problems in this space given the lack of alignment on the best ways to organize teams to work on cloud native applications at scale. Specifically, just copying a model used in one organization and using it in another organization does not address many of the challenges associated with things like politics and culture.

Market demand for engineering centric assets vs full blown applications. Historically, most applications are built to solve a targeted user or business problem end-to-end. In Mark Richards book entitled "Software Architecture Patterns" [31], Chapter 1 is focused on the *Layered Architecture Pattern*, which is shown in Figure 2. Richards shows this pattern containing 4 layers, but other variations exist like the 3-tier architecture that just focus on the presentation, business logic, and database layers are widely used. We found references going back to 1996 [32] describing this pattern. One of the foundational enablers of cloud computing is the plethora of first-class integration services that they provide opens the door for new software architectures based on offering API-enabled services as a product. Several well known examples of this new type of product are Google Maps (for location services), Twilio (for messaging), and Stripe (for payment). These all represent API-based products designed to be OEMd into other applications. We think this trend will expand and require software engineers to become familiar with designing API-centric products. For example, some of the best AI models are offered as an API service by Ope-

nAI [33], and the healthcare industry is being mandated to offer API services to support interoperability [34]. Some of these companies are even taking the opportunities to use APIs as a competitive differentiator, one example is the developer portal provided by Cigna**??**.

3. The Cloud is Expanding to the Edge

The underlying services that the major cloud providers offer to their customers continues to expand and evolve. These advancements provide significant innovation capabilities to customers, but also put pressure on how to effectively engineer solutions that take advantage of these services cost effectively. Figure 3 shows the high level view of the modern cloud. While it was once easy to identify the boundary of where the cloud started and ended, it is no longer easy to define the edge of the cloud. The following sections will provide an overview of the evolution of the cloud, along with highlighting some of the challenges that have to be overcome by software engineers to effectively use cloud computing services in their products.

3.1. The Basic Cloud Architecture

At a high level the basic cloud architectures deployed by major providers exhibit many similarities[3]. The foundational infrastructure building block of cloud compute is called an **Availability Zone** (AZ). An AZ is a data center that hosts cloud infrastructure (compute, storage, and network) and runs cloud provider services on behalf of their customers. A **Region** is a physical location where a collection of 2 or more AZs are located. Each AZ within a region are connected together with a fully redundant high bandwidth low latency network. The goal of a region is to have AZs close enough so that they can behave like a single cluster, but also separate them by enough distance to isolate them from issues associated with power failure, earthquakes, tornados, and so on. AWS, as an example, uses the general guideline of 100km (60 miles) [35] for placing AZs within a region. The global footprint of a cloud provider is defined by the number of regions and locations they have across the globe, along with the purpose-built underlying network they use to interconnect them together.

Given the cost and complexity of deploying cloud regions around the globe, cloud providers expand their network reach through the use of edge locations. Edge locations are useful for a couple of reasons. First, they serve as a point of presence to lower connection latency to the cloud, and second, they can run services at the edge which offers additional benefits. One of the classical applications to run at the edge is a content delivery network (CDN). CDNs speed up web and mobile applications. For digital web and mobile applications the combination of Regions, AZs, and Edge Locations could be considered the boundary of the cloud. These are shown on the right side of Figure3.

From a software engineering perspective, the basic cloud architecture described above introduces additional cognitive load on software engineers:

- Planning application deployment starts with the design of a virtual data center (VDC). VDCs logically carve out storage, compute, network and security policies from the cloud provider for customer usage. Traditional software engineers are not trained to think of the start of the software design process begins with the need to design a virtual data center and all of the complexity that comes with it. Historically, data centers are designed by specialty engineers and inherited "as is" into the final software architecture. The data center topology is now a critical software engineering concern.
- Quality attributes such as privacy, resiliency, reliability and scalability are foundational concepts that software architects use to reason about systems. These now move out of the conceptual realm and require a deeper understanding of technical constructs that now become part of the software design itself. For example, deploying microservices across different subnets, where each subnet is in a different AZ within a region. Also, declarative definition of the security policies that govern access to these microservices along with their entitlements to access other cloud resources becomes part of the software product itself.
- Although the cloud itself provides an infrastructure model to create resilient solutions that run at scale, its up to the software engineer to architect things properly to take advantage of these capabilities[4]. For example, it's still possible to deploy an application to a single virtual machine instance in the cloud, which will not elastically scale, nor will it be resilient to failure. Thus, to enable the creation of cloud services software engineers must have mastery of newer patterns for distributed applications (*e.g.*, especially asynchronous event-based architectures).
- One clear benefit of deploying to the cloud is that the easiest path to do so requires everything to

[3]It should be noted that the terminology used by the major cloud providers differs slightly for things like availability zones and regions.

[4]Some patterns such as rehosting [36] *a.k.a.* "lift-and-shift" should not be considered cloud native patterns.

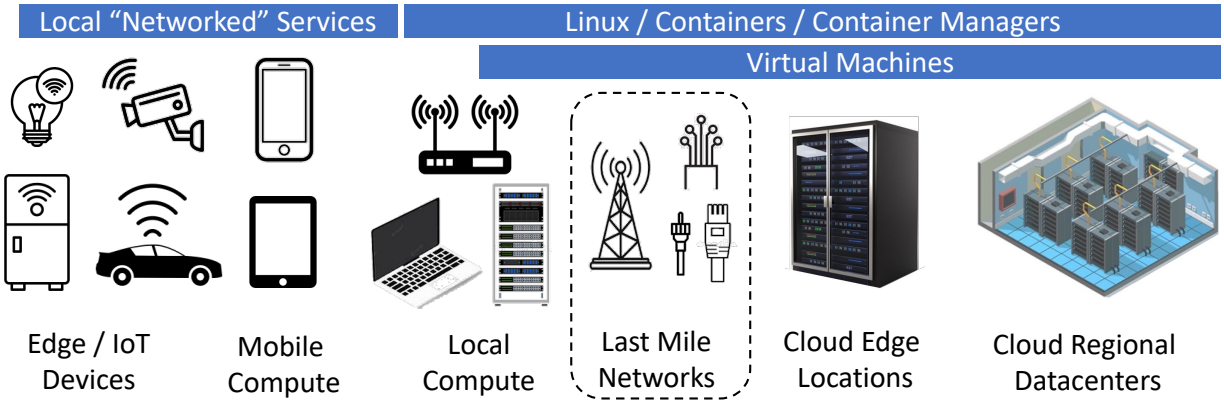

Figure 3. The Modern Cloud

be automated. While software engineers are comfortable with automation associated with software tasks like testing, they are not accustomed to automating infrastructure deployment. This becomes even more challenging given many of the existing infrastructure automation tools were designed for non-programmers, relying on verbose, complex and error-prone configuration formats like YAML and JSON. Some progress in this space has been accomplished via DSLs like Terraform [6] and tools that use real programming languages like Pulumi [8]

- Another additional aspect of software engineering for the cloud that is underdressed is that of the influence of design decisions that can materially impact operational runtime costs. This is often referred to as *finops*, short for financial operations. At its core, the cloud transforms compute, network, storage, and security into a pay-as-you-go utility. Software engineers generally don't factor in things like programming language selection, database platforms, the type of processor silicone, frameworks, fully-managed services and so on into their design from the perspective of cost and carbon footprint impact. We will explore this topic more in Section 4.

3.2. The Emergence of Edge Computing

With the rapid growth of devices that are connected to the internet, we are now entering the era of edge computing [37]. Edge computing is different architecturally from traditional cloud computing. Consider a cloud-enabled web or mobile application. The architecture of these applications is often based on calling cloud-hosted APIs and then using the data returned from these to power the user experience. This architecture will not scale or meet the needs of all of the smart devices that connect to the internet. As its name implies, edge computing moves more computing services to the edge, with requirements not found in web or mobile applications: Specifically:

- They must be able to work autonomously. The cloud would not scale to support all device events, local processing is used to filter important events from less important events.
- They must be able to work fully disconnected, or with unreliable network connectivity.
- They must be able to perform compute locally, either independently, or in local clusters.

These added capabilities basically extend the edge of the cloud all the way back to the client devices themselves as shown in Figure3. The overall architecture of the modern cloud that extends to the edge is shown in Figure 4[5].

According to research by IoT Analytics, there were 14.4 smart devices connected to the internet in 2022, expected to rise to almost 30 million by 2025 [38]. This many deployed devices could not be supported if they required connectivity to the large cloud data centers discussed earlier. Processing will need to move to the devices themselves supported by a new layer of cloud compute that is closer to the devices. This new layer of compute is often referred to as *fog computing*. The term comes from a play on the word cloud, given clouds are high up in the sky and fog is closer to the ground.

[5]Figure copied from `https://www.spiceworks.com/tech/cloud/articles/edge-vs-fog-computing/`

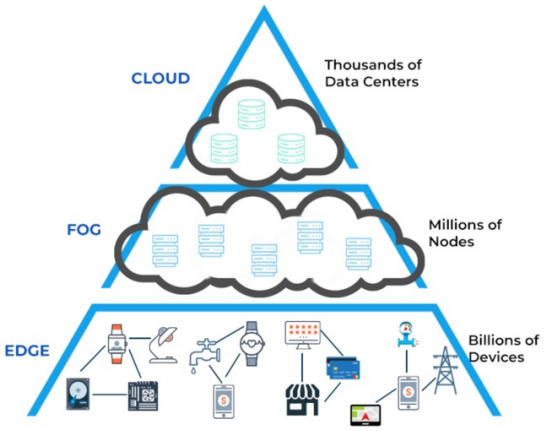

Figure 4. Edge Compute Architecture

As cloud providers expand their footprint across the globe, creating new regions with multiple availability zones represents an expensive and time consuming investment. Creating new regions is required to increase compute capacity as global cloud adoption expands, and to meet specific compliance requirements associated with conducting business in the cloud. For example, many countries are adopting data residency laws, which place controls over where data is stored at rest.

In many cases, new regions are not required, but cloud applications continue to push for better performance, which in many cases needs to be achieved via providing lower latency. Approaches to lower latency have been deployed before the modern cloud with content delivery network (CDN) [39] solutions.

4. Cloud Native Requires Embracing "Polyglot"

Underlying HW runtime diversity, Microcontrollers, etc

4.1. Polyglot Programming

blah

4.2. Polyglot Databases

blah

4.3. Polyglot Hardware

blah Docker - AMD and ARM AWS - Graviton Edge Computing - Microcontrollers - Go/Rust API Business model, twilleo and stripe

5. Software Architecture Requirements

6. Software Construction Requirements

6.1. Polyglot Programming

7. Acknowledgements

We would like to thank the following individuals for their valuable feedback to this paper.

References

[1] "IDC Forecasts Worldwide "Whole Cloud" Spending to Reach $1.3 Trillion by 2025". IDC. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS48208321

[2] "The History of Salesforce". Salesforce.com. [Online]. Available: \url{https://www.salesforce.com/news/stories/the-history-of-salesforce}

[3] "About AWS". Amazon Web Services. [Online]. Available: https://aws.amazon.com/about-aws/

[4] "The History of Microsoft Azure". Microsoft. [Online]. Available: \url{https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204}

[5] B. Stevens. "Google Cloud Platform: your Next home in the cloud". [Online]. Available: \url{https://cloud.google.com/blog/products/gcp/google-cloud-platform-your-next-home-in-the-cloud}

[6] "Automate Infrastructure on Any Cloud". Hashicorp. [Online]. Available: \url{https://terraform.io}

[7] "AWS CloudFormation". AWS. [Online]. Available: \url{https://aws.amazon.com/cloudformation}

[8] "Pulumi: Universal Infrastructure as Code". Pulumi. [Online]. Available: \url{https://www.pulumi.com}

[9] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw, *The DevOps Handbook: How to Create World-class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, LLC, 2016. [Online]. Available: https://books.google.com/books?id=Kvq5zQEACAAJ

[10] L. F. Albuquerque Jr, F. S. Ferraz, R. Oliveira, and S. Galdino, "Function-as-a-service x platform-as-a-service: Towards a comparative study on faas and paas," in *ICSEA*, 2017, pp. 206–212.

[11] D. S. Linthicum, "Cloud-native applications and cloud migration: The good, the bad, and the points between," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 12–14, 2017.

[12] "Cloud Native Computing Foundation Homepage". CNCF. [Online]. Available: \url{https://www.cncf.io/}

[13] "Use containers to Build, Share and Run your applications". Docker.com. [Online]. Available: \url{https://www.docker.com/resources/what-container}

[14] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research advances in cloud computing*. Springer, 2017, pp. 1–20.

[15] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with webassembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017, pp. 185–200.

[16] B. Bosshard, "On the use of web assembly in a serverless context," in *Agile Processes in Software Engineering and Extreme Programming–Workshops*, 2020, p. 141.

[17] "Production-Grade Container Orchestration". CNCF. [Online]. Available: \url{https://www.kubernetes.io}

[18] A. Wiggins. "The Twelve-Factor App". Heroku. [Online]. Available: \url{https://12factor.net}

[19] K. Hoffman, *Beyond the Twelve-factor App: Exploring the DNA of Highly Scalable, Resilient Cloud Applications*. O'Reilly Media, 2016. [Online]. Available: https://books.google.com/books?id=Ib3iuQEACAAJ

[20] "12 Factor App Revisited". architecture notes. [Online]. Available: \url{https://architecturenotes.co/12-factor-app-revisited/}

[21] A. Thomas and A. Gupta, "Adopt a mesh app and service architecture to power your digital business," Gartner Research, Tech. Rep. G00392875, July 2022.

[22] Z. Dehghani. "Data Mesh Principles and Logical Architecture". Thoughtworks. [Online]. Available: \url{https://martinfowler.com/articles/data-mesh-principles.html}

[23] E. Evans, M. Fowler, and E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004. [Online]. Available: https://books.google.com/books?id=xColAAPGubgC

[24] V. Vernon, *Implementing Domain-Driven Design*. Pearson Education, 2013. [Online]. Available: https://books.google.com/books?id=X7DpD5g3VP8C

[25] "WASI – The WebAssembly System Interface". CNDF. [Online]. Available: \url{https://wasi.dev/}

[26] F. P. Brooks, *The mythical man-month – Essays on Software-Engineering*. Addison-Wesley, 1975.

[27] J. D. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond." *IEEE Software*, vol. 16, no. 5, pp. 63–70, 1999. [Online]. Available: http://dblp.uni-trier.de/db/journals/software/software16.html#HerbslebG99

[28] A. Atlas, "Accidental adoption: The story of scrum at amazon.com," in *2009 Agile Conference*, 2009, pp. 135–140.

[29] "Discover the Spotify Model". Atlassian. [Online]. Available: \url{https://www.atlassian.com/agile/agile-at-scale/spotify}

[30] "Scaled Agile Framework (SAFe)". Scaled Agile. [Online]. Available: \url{https://www.scaledagileframework.com/}

[31] M. Richards, *Software architecture patterns*. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA , 2015, vol. 4.

[32] A. Aarsten, D. Brugali, and G. Menga, "Patterns for three-tier client/server applications," *Proceedings of Pattern Languages of Programs (PLoP96)*, vol. 4, no. 6, 1996.

[33] Build next-gen apps with openais powerful models. OpenAI. [Online]. Available: https://openai.com/api/

[34] Hl7 fhir. HL7. [Online]. Available: https://www.hl7.org/fhir/http.html/

[35] "Regions and Availability Zones". AWS. [Online]. Available: \url{https://aws.amazon.com/about-aws/global-infrastructure/regions_az}

[36] A. Engelsrud, "Moving to the cloud: Lift and shift," in *Managing PeopleSoft on the Oracle Cloud*. Springer, 2019, pp. 229–242.

[37] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, vol. 8, pp. 85 714–85 728, 2020.

[38] M. Hasan. State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally. IOT Analytics. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[39] A. Vakali and G. Pallis, "Content delivery networks: status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.