A Study of the Evolution of Secure Software Development Architectures

Leah Winkfield
l.d.winkfield14051@spartans.nsu.edu

Yen-Hung Hu
yhu@nsu.edu

Mary Ann Hoppa
mahoppa@nsu.edu

Norfolk State University
Norfolk, Virginia, USA 23504

Abstract - Emerging technologies such as containers, microservices, DevOps, Agile software development life cycle (SDLC), and cloud-native applications have gained popularity and traction in the industry and among enterprises. These modern application technologies and architectures are being adopted because they enable greater flexibility, scalability, portability and more rapid development. Consequently, how to build and maintain secure applications and systems is being reevaluated. Since the total responsibility is now larger and more complex, the application developer role is expanding to include greater security obligations and concerns. This paper explores the evolution of software development architectures and consequent implications on security, to better understand the technology landscape driving this change and its impacts on application development. To remain competitive, organization must be prepared to invest in ongoing training of their developers in the latest best practices. To remain relevant, higher education must adapt curriculums to prepare future professionals in the appropriate cybersecurity and secure coding practices to match the development shifts observed in industry.

Keywords

Secure Software Development, Agile SDLC, Container, Microservice, DevOps.

1 INTRODUCTION

Technologies such as the Internet of Things (IoT), social media and cloud computing are both drivers and consequences of an increasingly connected world, where software and computing interfaces are being integrated into even the most basic daily interactions and environments. Along with all the benefits these advancements contribute to society, comes an increasing number of software security concerns and vulnerabilities.

The number of software vulnerabilities is growing by orders of magnitude. Nearly 15,000 new vulnerabilities were discovered in 2017, up 128 percent from the prior year. Of these, over half (56%) were caused by inappropriate program codes such as arbitrary code execution, buffer overflow, SQL injection, Cross Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and file inclusion [1]. Since the resources provided to mobile software development are limited compared to traditional computer applications, we can expect this issue to become even more serious as the demand for mobile apps continues to grow.

Considering potential damages that can result from these bad codes, the software industry faces a formidable challenge when seeking mitigation approaches, among them innovating software development frameworks, along with adopting and improving the efficiency of better SDLCs. The underlying question is how software developers can continue to encourage application innovation in ways that ensure secured systems, inter-service communication and transactions. To serve as better security stewards, developers must solidify themselves as competitively qualified assets within their organizations, while at the same time influencing cultural and technological standards from the bottom up.

The remainder of this paper provides the motivation behind the whys and hows of these changes, and is organized as follows: Section 2 presents trends in innovating software development frameworks. Section 3 explains the need for adopting new SDLCs. Section 4 describes several approaches for improving SDLC efficiencies. Section 5 summarizes modern software development challenges and solutions. Section 6 discusses the implications of shifting security concerns. Section 7 summarizes key points and hints at some future directions.

Timeframe	Architecture paradigm	Security / Deployment Concerns	Environment Effects
1970-1990	Monolithic terminal server systems (mainframes)	Deploy once; secure the monolith	Transactional enterprises (e.g. banks, airlines, hospitality)
1990's	Client-server	More nodes to secure increases the security burden	Application logic abstractions; rich user interfaces
Early 2000's	Thin client	HTML forms introduce new vulnerabilities	High-latency; poor performance of JavaScript; browser incompatibilities
Mid-2000's	SOA, web toolkits	W3C standards improved security; but XML requires	Single-sign on is an infrequently realized ideal

Timeframe	Architecture paradigm	Security / Deployment Concerns	Environment Effects
		special security measures	
2010	Purely JavaScript frameworks	Rich, responsive, client-side experiences interacting with server-side data via RESTful backends	Strong testing frameworks & improved JavaScript makes security more manageable
Present day	Micro-services, containers	Full-stack developers; demand for client applications and real-time data	Ubiquitous internet, mobile devices, IoT

Table 1: Some Significant Paradigm Shifts in Software Development.

2 INNOVATING SOFTWARE DEVELOPMENT FRAMEWORKS

To keep pace with changes in the software industry, software development frameworks have evolved through several states: from centralized to distributed architectures; from dedicated hardware-software to virtualized systems; and from infrastructure-oriented to platform-oriented services. Table 1 summarizes some significant paradigm shifts in software development discussed in this section.

2.1 Centralized vs. Distributed Architecture

During the 1970's, 80's and 90's, monolithic terminal server systems (i.e., mainframes) ruled the world. In these systems, application logic was centralized, while terminals simply displayed data and captured user input. Security and deployment for the mainframe was relatively straightforward: no part of the application resided on the client, so it was possible to deploy once and simply secure the monolith. These systems were considered extremely fast and powerful at the time, but the character-based terminal interface was slow and very limited by current standards [2]. Today, many large enterprises still rely on mainframe technology, especially highly transactional businesses such as banks, airlines, and the hospitality industry. Only now – for security reasons – mainframes usually are walled off from the Internet and accessed via an abstraction layer.

The successor to terminal server architecture came in the early 1990's with the introduction of client-server architecture [2]. Much of the application logic was abstracted out to the client, which essentially rendered the mainframe a highly efficient transactional database server for storing and sharing data between clients. One benefit of this transition was more rich interfaces; but it also introduced its share of challenges [3]. Deployments had to reach each individual client, and any client could potentially be running incompatible operating systems (OS). Standalone clients also increased the security burden, since there were more nodes to secure.

The deployment complexity of early client-server architectures influenced the shift to thin client architectures in the early 2000's. The majority of application logic was pulled back onto the server-side. This restored deployment simplicity, but also reintroduced the original challenges associated with server-side call backs for every client-side interaction. The growing accessibility of the Internet meant call back latency in transactions had become even less tolerable [4]. Securing client-server applications also was more complex due to vulnerabilities introduced by HTML forms, the poor performance of early client-side JavaScript, and multi-browser incompatibilities [2].

2.2 Dedicated Hardware – Software vs. Virtualized System

A dedicated hardware-software system is designed for a single customer running a single OS. Before virtualization and cloud computing became popular, there was only the concept of bare metal and single tenancy. Virtual machines (VMs) were born out of the inability of most bare metal applications to take full advantage of rapidly increasing processing power and capacity [5].

VMs add a layer of abstraction called a hypervisor on top of the host operating system, making it independent of the underlying hardware resources. Each VM thus can run its own unique OS alongside the other tenants, creating a multi-tenant environment on a single server. Operating system virtualization "has grown in popularity over the last decade as a means to enable software to run predictably and well when moved from one server environment to another. Containers provide a way to run these isolated systems on a single server/host OS" [5].

In contrast to dedicated hardware-software systems, VMs provide new usage models (i.e., virtual environments) that benefit security in current systems. Some of the security advantages are [6] [7] [8] [9] [10]:

- Transience: Dedicated hardware-software systems are always on, even when they are not in use. In contrast, VMs are used on-demand, which means they should be always in use and directly monitored at all times. Therefore, users of VMs are more likely to detect malicious activities and intrusions than users of dedicated hardware-software systems.
- Abstraction: The abstraction layer (i.e., hypervisor) of VMs provides additional security. VMs do not know the hardware and OS configuration details of their host machine, so compromising a guest VM gives the attacker no immediate help in gaining access to the host machine's resources.
- Isolation: The hypervisor of a virtualized system reserves specific hardware sources for each VM and allows them to run independently. This approach restricts interactions among all VMs and reduces impact on the host OS and other VMs when a VM is affected by an attack.

- State Restoration: The hypervisor treats each VM image like a file on the host OS, which means a VM's state can be saved, cloned, moved, encrypted, or restored as desired. In case a VM is affected by an attacker, the hypervisor can restore the VM to a previous state by reloading its snapshot. This provides some protection against data loss, as well as a pathway to investigate attacks.

2.3 Infrastructure – Oriented vs. Platform – Oriented Services

The mid-2000's witnessed significant advancements in browsers and web tooling technologies that gave rise to more advanced client-side logic [2]. Adoption of and compliance with World Wide Web Consortium (W3C) [11] standards improved browser security. The introduction of the Google Web toolkit made rich client architectures possible and more accessible to traditional Java developers.

Service-Oriented Architecture (SOA) [12] gained traction during this time too, with the backing of a W3C recommendation [13]. An SOA is a software design pattern in which application components provide services to other components via network communication protocols. SOAs, such as Simple Object Access Protocol (SOAP) [14] [15] are implemented with web services. This makes the functional building-blocks of applications accessible over standard Internet protocols that are platform and programming language agnostic.

An infrastructure-oriented service focuses on providing customized software development services for clients (i.e., software developers) by taking advantage of VM technology. In an Infrastructure-as-a-Service (IaaS) model – also known as Hardware-as-a-Service (HaaS) – each client of the platform can have his own software configuration on hardware that is virtualized and pre-configured based on the service agreement. IaaS offers clients the flexibility to deploy their own guest OS as well as all required applications. This enables them to take advantages of VM technology without labor-intensive server management and/or hardware investments. Advantages and disadvantages of VMs, described in the previous section, also apply to IaaS. In addition, security patches and OS updates are still the responsibility of each client.

In a Platform-as-a-Service (PaaS) model, a client manages only the applications he installs and configures; all other hardware and software, including the OS, are pre-configured based on the service agreement. In this model, clients can spend more time coding, testing, and deploying their applications without worrying about managing OS updates and security patches. PaaS also provides tools and APIs that enable clients to adopt new features and guidelines to support their software development best practices. However, fully relying on the service provider for underlying security implementations may introduce uncertainty and expose the client to unacceptable risks. This suggests that clients should perform a thorough investigation and assessment of security implementations for alternative PaaS providers before making an investment decision, and ongoing auditing after selection.

3 ADOPTING BETTER SDLC

As mobile technology advanced and demand increased, the software development framework started moving from centralized to distributed architectures. This change inherently requires reconsidering security assumptions and practices that often are overlooked and/or undervalued by key enterprise stakeholders – including those engaged in development, security, networking, architecting, etc. – who likely have become accustomed to managing tiered monolithic systems in ways that are no longer adequate in a distributed computing context.

In contrast to the traditional and sequential Waterfall delivery cycle, new philosophies enable speed to market by leveraging cross-functional teams, shorter feedback loops, and iterative, continuous delivery. Agile Development is one such example.

4 IMPROVING EFFICIENCY OF SDLC

Microservices architectures [16] [17] [18], container virtualization [19], and DevOps [20] are among the solutions born to enable efficiencies in the SDLC.

Microservices is a subset of SOAs [21] that describes "a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal" [16]. Microservices applications offer several developmental, operational and security advantages. For example, developers are able to create and deploy smaller incremental features or application changes, without the burden of redeploying an entire application system.

Containers virtualize an underlying host's OS and provide a resource-isolated runtime environment [19]. Although containers have existed for quite some time, Docker [22], an open-source technology company, recently increased their popularity. Because they inherently enable portability, containers have played a significant role in facilitating the migration of traditionally on-premise application workloads to more distributed platforms, such as public cloud infrastructures.

DevOps is described as "the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support" [20]. The principles and practices that constitute the DevOps culture have been so widely adopted because, when employed, they support and enable the goals of Agile Development. These technologies represent industry solutions to technical challenges. When trends and practices such as these gain popularity, enterprises scramble to leverage them for innovation and positive organizational change.

Organizations who are eager to implement cutting edge technology and patterns may too hastily adopt and attempt to implement microservices running on container platforms, which can lead to significant challenges and failures. Unfortunately, some observers have unfairly criticized microservices, containers, DevOps, and other emergent principles and technologies for these shortfalls, rather than pointing the finger at the organizations whose use of them have proven to be incomplete and fault.

5 MODERN SOFTWARE DEVELOPMENT SOLUTIONS AND CONCERNS

Considering the technological shifts outlined above, it should be evident that modern computing demands and requirements are outgrowing the limitations of legacy infrastructure, architectures, patterns and languages. Enterprises are on a course to failure if they are unable to quickly innovate and deliver to market, pivot with agility, or scale elastically in response to industry trends and projections.

In fact, enterprises across all industries are wrestling with having to quickly make critical cultural and technological decisions that impact every aspect of their business regardless of whether they identify as a "tech company." While they can still be successful, monolithic and tiered systems are no longer ideal. The practice of creating separate functional siloes (e.g., Operations, Development, Security, Infrastructure), and horizontally stratifying application teams is now a hindrance to enterprises realizing their business objectives. Meticulously designing applications so they are dependent on the hardware and environments in which they run limits flexibility and stifles innovation. Most importantly, security and governance concerns are more complex, and can no longer remain responsibilities isolated to a single security team operating independently of the SDLC.

5.1 Modern Software Development Solutions

With speed and agility being paramount to success, naturally solutions have arisen to pursue these advantages. For example, some businesses that previously operated their own on-premise data centers are now opting to partially or fully outsource to cloud IaaS providers, so they can focus their attention on application delivery. Among the platforms being used by businesses to scale and grow are Amazon Web Services [23], Google Cloud Platform [24], and Microsoft Azure [25].

Software development strategy has been transitioned from infrastructure-based to "-as-a-service" offerings [26], where each such offering provides a subset of the computing layers grouped as a platform Platform-as-a-Service (PaaS) providers include RedHat OpenShift [27], Pivotal CloudFoundry [28], and Heroku [29].

Similarly, home-grown software has given way to vendor managed/hosted products. Software-as-a-Service (SaaS) refers to centrally hosted, on-demand software services that may be licensed, subscription-based, or even free to use. Common Software-as-a-Service providers are Gmail [30], Salesforce.com [29], and social media sites such as Facebook [31].

Goals such as speed, agility, flexibility and portability often don't align well with traditional architecture patterns, organizational structures and delivery cadences. To meet the speed of contemporary market demands and therefore remain competitive, companies are adopting delivery and cultural philosophies such as Agile Development and DevOps. Agile Development has taken the traditional waterfall process and reduced the time-to-value by implementing iterative, compressed delivery cycles. This results in shorter feedback loops, reduced risk, fast feature delivery, and greater overall flexibility. DevOps attempts to align the competing goals of developers who want to innovate and move software and products to the market faster, and operations who prefer to keep things stable, so the two can join forces and deliver greater value downstream.

When adhered to, these principles simultaneously address technology, culture, process and organization, and their value comes from using them in combination. These principles help to frame the various decisions that must be made when building systems. Design informed by these principles is crucial, because although microservices themselves may be small, the breadth and impact of their architectures are not. Distributed architecture also raises new security challenges and concerns such as service-to-service authorization and communication [32].

Regardless of architecture, an organization's structure and culture is key to their resilience and success. In fact, Conway's Law states that "Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure" [33]. This is the motivation behind cultural frameworks such as Agile and DevOps. To achieve speed to market, high availability, and rapid

innovation, the organization has to operate in a manner that reflects and supports these goals.

5.2 Modern Software Development Concerns

Requiring authentication for every service call with SOAs was cumbersome due to its intentionally decentralized nature and multidirectional data flows. A Single Sign On (SSO) capability would be an ideal solution, "where a user's credentials are promulgated throughout the Global Information Grid (GIG) to reach all desired services;" but this implementation carries its own challenges [12].

Furthermore, Extensible Markup Language (XML) – which is used to format some SOA application data – is "inherently insecure," and as such special measures must be taken to properly manage its vulnerabilities [12]. Nevertheless, enterprise SOA implementations are prevalent, even today. It wasn't until 2010 that sophisticated, purely JavaScript frameworks such as Ember.JS, Backbone.JS, and Angular.JS [34] [35] revolutionized the modern web by creating rich, responsive, client-side experiences interacting with server-side data via RESTful backend services [35].

Despite significant technological advancements and growing demand, there still existed a great divide between traditional server side application developers and "web" developers. It took some time to bridge that gap due to perceptions that web languages – such as HTML and JavaScript – were less robust or simply used for integration; and that front end developers were less skilled than their back end counterparts. This naturally slowed adoption of rich client enterprise architectures, but also gave birth to the Full-stack Application Developer role. These new developers exhibit a breadth of familiarity and skill throughout the stack layers, and are able to "build complex server-side web applications that use powerful relational databases to persistently store data," and to support multiple front ends [36].

Meanwhile, amid the ebbs and flows of change in the software world, substantial shifts in the hardware landscape began to take shape. For one thing, smart phones and other mobile devices increased digital communication, accessibility to the

Internet, lightweight programming languages, network traffic, demand for client applications, and real-time data.

Microcontrollers, embedded processors, and wearable devices ushered in the IoT era, which has added "smarts," data streaming and connectivity to a myriad of objects communicating over common protocols such as Zigbee [37], Wi-Fi, Bluetooth, and Z-wave [38].

Computing devices have gotten smaller, faster, smarter and increasingly ubiquitous. All these new endpoints, devices and modes of interaction have swelled the data volume, whose boundaries and edges have to be secured and managed in new ways.

6 THE IMPLICATIONS OF SHIFTING SECURITY CONCERNS

Traditional enterprise security is generally the responsibility of a single internal security department comprised of a small group of Information Security (InfoSec) professionals, and a "Red Team" for penetration testing, under the leadership of a Chief Information Security Officer (CISO). Responsibilities and concerns usually center around physical security, endpoint security, disaster recovery, content filtering and phishing prevention, intrusion prevention and protection, incident response and compliance.

Today, enterprises operate dozens to hundreds of application teams, running thousands of applications. Mounting cybersecurity threats mean greater emphasis must be placed on risk assessment, mitigation testing and approval, and external attestation. The increasingly compressed timelines driven by modern delivery frameworks only further compound the criticality and complexity of maintaining adequately hardened systems. Security professionals went from managing one to four software releases per year under traditional Waterfall delivery practices, to one or two releases per month under early Agile cadences. And now with the advent of modern Agile DevOps release cycles, the number of releases can easily soared to 100 or more annually.

Traditional security teams and practices are no longer sufficient for the following reasons:

- They intervene too late in the process to address most vulnerabilities.
- They move too slowly for today's SDLC.
- They are not cost effective for handling simple vulnerabilities.
- It is difficult and costly to find, hire, and train InfoSec professionals.
- Such teams are hard to scale.

To adapt security practices to preventing, mitigating and responding to modern threats and vulnerabilities, developers must play a more integral role; sharing accountability with security teams helps integrate security into every part of the SDLC, also referred to as "shifting security to the left." The entire organization benefits from having a more security-minded workforce. Teams experience less waiting and better support since processes are decentralized and [eventually] automated.

Focusing on improving the security conscience of the existing developer community — rather than replacing them with outside hires who lack unfamiliarity with the teams and products — is essential to the successful adoption of such a holistic security culture. The organization must invest in training their developers in new best practices. This will both improve the quality of their work products and help them build and maintain competitive, industry-relevant skills.

Cultural change is difficult and resistance is likely. Leveraging one or two respected development, security and operations "champions," and/or multi-discipline "champions" from each product team can be an effective strategy for influencing the broader community. The goal is to evolve the shared view of "security" away from the idea of an ominous, isolated enforcement team, and towards a collective conscience of security and accountability.

7 CONCLUSION

Speed and agility underpin success in the digital age. From a software development perspective, a fundamental question is how to achieve secure innovations at speed and scale, versus simply introducing more vulnerabilities into systems even faster. To better understand the role of application developers in the new security ecosystem, this paper explored the evolution of software development architectures and consequent implications on security that have both resulted from and driven SDLC framework innovation.

Increasingly compressed contemporary delivery timelines are requiring significant shifts and changes in the SDLC, which can be at odds with system stability and security. Developers are being called upon to play a more integral security role, by "shifting security to the left" into every part of the SDLC, and by sharing accountability with both the operations and security teams. Veracode and DevOps.com recently conducted a survey to assess the state of cybersecurity and DevOps skills in the workforce. Their findings highlight the fact that developers today lack the formal education and skills they need to produce secure software at the pace required; and moreover, relevant training sources are limited [39].

Along with committing to a number of significant changes in skills, culture, technology and processes, organization must be prepared to invest in ongoing training of their developers in the latest best practices. Higher education too must adapt curriculums to prepare future professionals in the appropriate cybersecurity and secure coding practices to match the skillsets required to meet development shifts in industry.

As a first step in that direction, we are creating a framework to serve as a basis for measuring an organization's "readiness" to meet new SDLC demands based on staff knowledge, skills and experiences in key areas of development, operations, security and culture. Such a framework also will allow organizations to gauge where to focus limited training resources and hiring efforts to remain competitive. Finally,

the framework can help academia establish and teach to relevant competency-facing outcomes so students are better prepared to enter the workforce.

REFERENCES

[1] "CVE Details," [Online]. Available: https://www.cvedetails.com/.

[2] E. Muguet, "The evolution of business applications architecture," 31 May 2015. [Online]. Available: https://www.linkedin.com/pulse/evolution-business-applications-architecture-eric.

[3] J. Chone, "The Five Software Architecture Generations: From Mainframe to Mobile Apps to HTML5," Brite Snow, 11 March 2013. [Online]. Available: http://britesnow.com/blog/software-architecture-evolution-mobile-apps-to-html5.

[4] D. A. Foroughi, "Client-Server and Intranet Computing," [Online]. Available: http://www.usi.edu/business/aforough/cis367notesf2003/c367ch17.htm.

[5] K. Boeckman, "Docker containers vs. virtual machines: What's the difference?" NetApp Blog, 16 March 2016. [Online]. Available: https://blog.netapp.com/blogs/containers-vs-vms.

[6] P. M. Chen and B. D. Noble, "When Virtual Is Better Than Real," in Proceedings of the 8th Workshop on Hot Topics in Operating Systems, 2001.

[7] D. Hyde, "A Survey on the Security of Virtual Machines," [Online]. Available: http://www.cse.wustl.edu/~jain/cse571-09/ftp/vmsec/index.html.

[8] J. S. Reuben, "A Survey on Virtual Machine Security," in Security of the End Hosts on the Internet Seminiar on the Network Security, 2007.

[9] T. Garfinkel and M. Rosenblum, "When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments," in Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HOTOS-X), Sante Fe, NM, 2005.

[10] G. Pek, L. Buttyan and B. Bencasath, "A Survey of Security Issue in Hardware Virtualization," ACM Computing Surveys, vol. 45, no. 3, 2013.

[11] "W3C," [Online]. Available: https://www.w3.org/.

[12] C. Phan, "Service Oriented Architecture (SOA) - Security Challenges and Mitigation Strategies," MILCOM 2007, Orlando, 2007.

[13] W3C, "Web Services Architecture," 11 2 2004. [Online]. Available: https://www.w3.org/TR/ws-arch/.

[14] "SOAP"" [Online]. Available: https://www.w3.org/TR/soap.

[15] SimplyEasyLearning, "What is SOAP?" [Online]. Available: https://www.tutorialspoint.com/soap/what_is_soap.htm.

[16] T. Huston, "What is Microservices Architecture?" SmartBear, [Online]. Available: https://smartbear.com/learn/api-design/what-are-microservices/.

[17] M. F. James Lewis, "Microservices in a Nutshell," ThoughtWorks, 27 June 2014. [Online]. Available: https://www.thoughtworks.com/insights/blog/microservices-nutshell.

[18] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Rielly, 2014.

[19] V. Badola, "Container Virtulization: what makes it work so well?" 27 October 2015. [Online]. Available: https://cloudacademy.com/blog/container-virtualization/.

[20] E. Mueller, "What Is DevOps?" 2 August 2010. [Online]. Available: https://theagileadmin.com/what-is-devops.

[21] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture," January 2004. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa480021.aspx.

[22] Docker, "Home Page," Docker, [Online]. Available: https://www.docker.com.

[23] Amazon, "Start Building on AWS Today," Amazon, [Online]. Available: https://aws.amazon.com.

[24] Google, "Build What's Next," Google, [Online]. Available: https://cloud.google.com.

[25] Microsoft, "Your vision. Your cloud." Microsoft, [Online]. Available: https://azure.microsoft.com/en-us/?v=17.42n.

[26] J. M. Bond, "Who Manages Cloud IaaS, PaaS, and SaaS Services," 19 June 2013. [Online]. Available: https://mycloudblog7.wordpress.com/2013/06/19/who-manages-cloud-iaas-paas-and-saas-services/#5ervices/.

[27] RedHat, "Develop, Deploy, and Manage Your Containers," RedHat, [Online]. Available: https://www.openshift.com.

[28] "Cloud-Native Platform for Deploying and Operating Modern Applications," Cloud Foundtry, [Online]. Available: https://pivotal.io/platform.

[29] Safeforce, "Turn your company into an apps company," Safeforce, [Online]. Available: www.heroku.com.

[30] Google, "Google," Google, [Online]. Available: www.gmail.com.

[31] Facebook, "Facebook," Facebook, [Online]. Available: https://www.facebook.com.

[32] M. Fowler, "Microservice Trade-Offs," 01 July 2015. [Online]. Available: https://martinfowler.com/articles/microservice-trade-offs.html.

[33] F. Brooks, "Conway's Law," [Online]. Available: http://www.melconway.com/Home/Conways_Law.html.

[34] "JavaScript," [Online]. Available: https://www.javascript.com/.

[35] SecureAuth, "AngularJS Best Practices Guide," [Online]. Available: https://www.secureauth.com/sites/default/files/angularjs_best_practices_guide.pdf.

[36] Udacity, "Full Stack Web Developer Nanodegree," Udacity, [Online]. Available: https://www.udacity.com/course/full-stack-web-developer-nanodegree--nd004.

[37] ZigBee Alliance, "Home Page," [Online]. Available: http://www.zigbee.org/.

[38] R. Components, "11 Internet of Things (IoT) Protocols You Need to Know About," [Online]. Available: https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about.

[39] VERACODE, "Securing The Software That Powers Your World," VERACODE, [Online]. Available: https://www.veracode.com/.