Cloud Software Engineering: Traditional or Innovative – The Choice Is Yours

Christoph Bussler
Oracle Corporation, Redwood City, CA 94065, U.S.A.

Keywords: Cloud Software Engineering, IaaS (Infrastructure as a Service), PaaS (Platform as a Service).

Abstract: Cloud environments provide different levels of resource abstractions, most commonly categorized as IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) as well as SaaS (Software as a Service – which is not relevant for this position paper). A detailed discussion of the difference between the IaaS and PaaS abstractions in this paper will lead to the following position: a single cloud software engineering process will not be sufficient for software development in the cloud. Depending on the target abstraction (IaaS or PaaS), the software engineering process will have to be different. It is predicted that the target abstractions of PaaS will dominate those of IaaS in the long run.

1 SOFTWARE ENGINEERING FOR THE CLOUD

1.1 Software Engineering Definition

According to the ISO/IEC/IEEE International Standard the term software engineering is defined as "*application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*" (IEEE, 2017).

This definition is important as it not only covers the software construction (development) part of software engineering, but also includes operation and maintenance in production.

This is extremely relevant in the cloud context as the trend is going towards the total ownership of the code life cycle by engineering teams. This trend is called DevOps (AWS DevOps, 2018) and means that a single team is responsible for its software and owns all aspect as outlined in the software engineering definition above: not only development, but also operation and maintenance.

A DevOps model makes the complete software lifecycle the focus of the engineering teams that in the past only owned the software development part of it, not the operation or maintenance aspect.

Once the operational difficulties are known to the engineering teams they will most likely make different technology and engineering process choices of how to develop a cloud service or a set of cloud services once they are also responsible for the operations and maintenance in production.

This will be a major aspect in the following discussion and plays into the position of the author on software engineering in the cloud: Cloud Software Engineering – one size fits nobody.

1.2 Questions in Context of Cloud Computing

A term not explicitly mentioned in above definition of software engineering is “deployment”. Deployment is the process of installing the outcome of the development process – the software artefacts – to the target system resources, meaning, computing resources like (virtual or bare metal) machines as well as the required middleware like databases or queuing systems.

Deployment is implicitly included, however, in the above definition of software engineering, as without deployment the operation and management is impossible. Deployment becomes a crucial process in context of the cloud: until the existence of the cloud concept the activity of software engineering was for on-premise infrastructure, aka, deployment of software artefacts in data centers that are in control of the organizations themselves.

Organizations determined on their own how their data centers are designed, and what resources are

Bussler, C.
Cloud Software Engineering: Traditional or Innovative – The Choice Is Yours.
DOI: 10.5220/0006912408710878
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 871-878
ISBN: 978-989-758-320-9

being made available to their engineering teams. It is safe to say that every organization's data center was designed differently and therefore shaped the individual software engineering process of that organization to significant extent.

In a cloud the available target system resources are defined by the owner of the cloud (not the customer) and are the same for all customers that are deploying software artefacts into this cloud. Some interesting questions are for a given cloud:

- Given a cloud, and its provided target system resources, what software artefacts does a software engineering process have to develop?
- Given target system resources, how are software artefacts deployed?
- Once deployed, how are the deployed software artefacts operated and maintained in the cloud?

Not all clouds are the same. Different cloud providers made different design choices of how to structure their cloud, and what target system resources to provide. Major cloud provides are Amazon (AWS, 2018), Google (Google, 2018), Microsoft (Microsoft, 2018), and Oracle (Oracle, 2018), to name a few.

1.3 Cloud Software Engineering

The term "cloud software engineering" refers in this position paper to software engineering for a specific cloud with software artefacts developed for that cloud's target system resources first and then subsequently deployed, operated and maintained.

The development phase has many parts to it, including use case analysis, requirements analysis, functional design, non-functional design, implementation or testing. It might follow an agile process or a more rigorous process, depending on the industry or domain within an industry.

In the following, the cloud software engineering process is not discussed to this level of detail as only the outcome in terms of the produced artefacts is relevant for the focus of the discussion.

1.4 Outline of the Position Paper

The remainder of the position paper addresses in Section 2 a very important necessary differentiation when it comes to the target system resources of a given cloud: the customary distinction between IaaS and PaaS. Section 3 provides an example that illustrates the difference of developing for IaaS vs. PaaS target resources. The author's position is stated and elaborated in Section 4. Related work and a conclusion is provided in Section 5 and Section 6, respectively.

2 CLOUD COMPUTING: ALTERNATIVE DEPLOYMENT TARGET SYSTEM RESOURCES

While clouds are different from each other, all cloud providers distinguish IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service – not relevant for the discussion in this position paper).

2.1 Resource Centric (IaaS) versus Functionality Centric (PaaS) Target System Resources

IaaS can be referred to as a resource centric ("traditional") target system. This is the same approach as in a traditional on-premise data center only that the system resources are in the cloud instead of on-premise and have to be taken as provided (no customization possible beyond what has been foreseen by the cloud provider). Typical examples of system resources are (virtual or bare metal) servers, networks as well as block storage; see (Oracle IaaS, 2018), for example, for IaaS resources provided by the Oracle Cloud.

PaaS, in contrast, can be referred to as a functionality centric ("innovative") target system. This is very different from traditional on-premise data centers as it does not expose any infrastructure system resources, but only provides for the specification and use of functionality (often termed serverless architectures). Infrastructure system resources are abstracted away and not accessible by users. Examples are functions, API gateway or database as a service; see (AWS Reference Architecture, 2018) for various example resources. An overview discussion of serverless architectures is in (Roberts, 2018). The following Table 1 shows the difference between IaaS and PaaS in form of examples.

An example use case will be discussed below that shows the difference of developing the same solution on a detailed level: the software artefacts that need to be developed and the process of deployment into the different target system classes.

Table 1: Target System Resources for Resource Centric and Functionality Centric Classification.

Target System Classification	**Target System Resources in Cloud (Examples)**
Resource centric ("IaaS")	• Virtual or bare metal machines • Storage • Network
Functionality centric ("PaaS")	• Functions • API gateway • Database as a service

On a very abstract level, both alternatives are fine and choosing to develop for one or the other is an acceptable choice – nothing is fundamentally wrong with one or the other. However, as always, specific details will make one choice superior over the other.

From a development viewpoint, the following Table 2 shows the difference in the software artefacts as well as the deployment implications; the concrete example use case (below) will discuss the requirements for each in more detail.

Table 2: Development and Deployment for Resource Centric and Functionality Centric Classification.

Target System Classification	**Development and Deployment**
Resource centric ("IaaS")	Software artefacts to be developed • Executables to run on virtual or bare metal machines Deployment activities • Create virtual or bare metal machines, install web servers and databases, deploy executables, setup networks and configure storage
Functionality centric ("PaaS")	Software artefacts to be developed • Functions, API gateway configurations, database access Deployment activities • Upload functions, API gateway configurations, create database account

The examples in Table 2 show very clearly that the artefacts to be developed and their deployment is very different when IaaS resources are targeted versus PaaS resources.

From an operations and maintenance perspective, the following Table 3 shows the difference for a select set of life cycle management operations.

Table 3: Operation and Maintenance for Resource Centric and Functionality Centric Classification.

Target System Classification	**Operation and Maintenance**
Resource centric ("IaaS")	• Scale servers (scale-out and scale-in) • Restart failed processes and servers • Monitor resource utilization and saturation • Monitor executables for failures • Monitor database for proper functioning
Functionality centric ("PaaS")	• Monitor execution of functions for failures • Monitor database for proper functioning

Table 3 shows that the effort to operate and to maintain software deployed to IaaS is a lot higher and more resource intensive than software deployed to PaaS. Since the basic infrastructure resources are abstracted away in the PaaS case they are consequently not part of the operations or maintenance a DevOps team has to perform.

2.2 Fundamental Choices

An engineering team developing software artefacts for deployment in a cloud has to make a choice and a decision very early on in the software engineering process if the service artefacts are designed and developed for IaaS target system resources or PaaS target system resources (or both).

The distinction between IaaS and PaaS (as discussed above) poses a fundamental and consequential choice impacting the engineering and maintenance activities of DevOps teams:

- **IaaS**: Is the cloud viewed as an infrastructure resource provider of (virtual or bare metal) machines, network and storage?
- **PaaS**: Is the cloud viewed as functionality provider of higher level functionality like, for example, functions or persistence services abstracting from infrastructure resources?

CaaS (Container-as-a-Service) (Kubernetes CaaS, 2018) will be discussed in Section 3.3 after the paper outlines the detailed software engineering activities as those are needed as a basis for the discussion how CaaS fits into the discussion.

2.3 Impact to Devops Teams

The choice and decision has not only an extremely high impact for the software engineering process, but also for the DevOps model where operations and maintenance will have to be done on an infrastructure or higher level functionality level, depending on the choice of IaaS or PaaS. The above Tables 1, 2 and 3 show the fundamental difference.

To emphasize, once engineering teams are responsible for the whole software engineering process, from development to maintenance (DevOps teams), the above discussion shows that the functionality centric (PaaS) target system might be advantageous as the level of abstraction is significantly higher, and the deployment, operations and maintenance effort a lot less compared to the resource centric (IaaS) target system.

This is significant as team resources devoted to deployment and maintenance in context of IaaS could be redirected to functionality development in the PaaS context. This is a huge difference as from a client viewpoint, additional functionality is always desired. From a development viewpoint, functionality usually differentiates from the competition.

To be complete in the discussion, a mixed use case might exist as well, where an engineering team decides to develop some of the software using PaaS resources, and some using IaaS resources.

While this use case cannot be excluded, of course, the choice ideally is made very carefully since this mix will permeate all phases of the cloud software engineering process.

3 EXAMPLE USE CASE FOR CLOUD SOFTWARE ENGINEERING

An example use case will illustrate in the following how the choice of an engineering team to develop for IaaS or PaaS resources will influence the cloud software engineering process.

The software will address a specific requirement: compute the lead time for the outstanding parts of an order as part of a supply chain management system (Wikipedia 2018a). A single function suffices that given the input order identifier computes the lead time for all parts to be available as output in expected number of days. This simple application is chosen in order to avoid distraction from the cloud software engineering aspects. This function will be named “LeadTime()” in the rest of the paper as a shorthand notation.

For both cases the programming language Java is chosen, as well as the GIT version control system. The goal is a scalable system that can scale out and scale in as the load requires. Database access is not required (simplification for discussion).

3.1 Leadtime() for IaaS

The following Table 4 structures the activities of the engineering team by the cloud software engineering process phases. The activities are described to sufficient detail in order to demonstrate the distinction between targeting IaaS or PaaS. As an architecture solution Kubernetes (a container management system) is chosen for its ability to provide scaling as well as failure mitigation for failing processes (Kubernetes, 2018) with Helm as the package manager (Helm, 2018).

Table 4: Software Engineering Activities for IaaS.

Cloud Software Engineering Process Phase	**Activities**
Development	Develop Java code implementing LeadTime() Develop Docker image running LeadTime() Java code Develop Helm charts to deploy the LeadTime() image into Kubernetes
Deployment	Create VMs Install Kubernetes software Create Kubernetes cluster Upload Docker image to registry Upload Helm charts Create the supply chain application by running Helm charts
Operations and Maintenance	Monitor Kubernetes cluster for failures Monitor LeadTime() code for failures Scale by changing Helm charts and reapplying those

The skill set of the engineering team must be quite diverse as it has to deploy, operate and maintain IaaS resources, the Kubernetes system (including Docker and Helm) as well as the application itself (which is simplified here in form of the LeadTime() function). This skill set is the same as in the case if the application were to be developed for an on-premise data center.

3.2 Leadtime() for PaaS

The effort to target a PaaS system is a lot less due to the higher level of abstraction removing the various IaaS resources. The following Table 5 shows the various engineering team activities for deploying a function like LeadTime() according to the AWS reference architecture (AWS Reference Architecture, 2018).

Table 5: Software Engineering Activities for PaaS.

Cloud Software Engineering Process Phase	**Activities**
Development	• Develop Java code implementing LeadTime()
Deployment	• Create function and upload Java code in prescribed packaging model • Configure API gateway
Operations and Maintenance	• Monitor function execution • Monitor gateway health

The difference is striking as all activities around creating, deploying, operating and maintaining infrastructure resources as well as the Kubernetes system (with Docker and Helm) are not necessary at all. This is not a surprise as this is the declared goal of providing a higher level of abstraction.

The skill set required by an engineering team is a lot less as the sole focus can be (mostly) on the application functionality, not on infrastructure resources and additional (middleware) software required like Kubernetes, Helm and Docker.

3.3 Caas (Container-as-a-Service)

A service called CaaS (Container-as-a-Service) is being made available by many cloud providers (Kubernetes CaaS, 2018) and very often, if not always, CaaS is being put into the PaaS category. CaaS is briefly characterized in the following and discussed in context of the LeadTime() engineering activities.

CaaS is a service that supports the ability to create one or more Kubernetes clusters without having to create VMs and install the Kubernetes container management software onto the VMs itself. Once a cluster is created, the required Docker images and Helm charts must be uploaded in order for the cluster to configure and start up the various Docker images accordingly.

Having discussed the engineering activities for Kubernetes in Section 3.1 the question arises, what would CaaS simplify across all activities?

Examining Table 4 it becomes clear that two activities are not necessary anymore: the creation of VMs, and the installation of the Kubernetes software. All other activities remain.

While creating VMs and installing the Kubernetes system itself is a significant effort, it does not really affect the development activities or maintenance activities at all; and from the deployment activities it only removes the initial one-time setup.

Therefore, from the viewpoint of the activities of a DevOps model, CaaS really is more like an IaaS service that provides Kubernetes as the infrastructure instead of VMs.

3.4 Microservice Architecture

An architectural style called "Microservice Architecture" has been discussed. According to (Lewis et al. 2018), "*The term 'Microservice Architecture' has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services.*"

A more detailed architecture discussion from an implementation perspective is outline in (Newman 2015). In this book Kubernetes is referred to as a possible execution infrastructure for microservices architectures as Docker containers alone are insufficient to provide the required functionality like scheduling, allocation, or load balancing.

(Balalaie et al. 2016) describe experiences in their work migrating an application to a microservices architecture executing on a Kubernetes infrastructure.

While implementing a microservices architecture on Kubernetes is not the only approach or option,

the references clearly show that the target is the IaaS layer and not the PaaS layer.

4 POSITION: CLOUD SOFTWARE ENGINEERING – ONE SIZE FITS NOBODY

Based on the above discussion on the difference and distinction between IaaS and PaaS resources provided by various clouds there is not a single cloud software engineering process but several forms that have to be distinguished based on the target system resource cloud offerings (aka, IaaS or PaaS).

4.1 Resource Centric "Traditional" Cloud Software Engineering (IaaS)

The resource centric, or "traditional" cloud software engineering process is very similar to software development for on-premise data centers. The main difference is that the resources that are available for deployment like VMs or bare metal machines reside in the cloud instead of in on-premise data centers.

Aside from the added complexity of operating those resources remotely, from a software engineering perspective there is no significant difference as the same principles, same techniques, same architecture trade-offs and same software technologies apply. The similarity of the software engineering applies also for the deployment, operation and maintenance activities, not just for the development aspect. Some slight differences exist in terms of latency and throughput considerations, locality, jurisdictions, but no real or fundamental differences.

The focus of the cloud software engineering approach for IaaS is two pronged:

- **Infrastructure.** The software engineering process has to develop the artefacts required to setup the infrastructure like executables, Docker containers, queue system configuration specifications, and so on.
- **Functionality.** As a second focus, the software engineering process has to develop the functionality of the application that is executed on the infrastructure. A proper design and decomposition has to take place in order to ensure that the application runs on the infrastructure in terms of correctness, latency, throughput, and so on.

4.2 Functionality Centric "Innovative" Cloud Software Engineering (PaaS)

Clouds are providing and continuing to provide higher level functionality that at the same time abstracts away infrastructure resources completely. For example, AWS Lambda functions can be defined and executed without having any visibility (meaning, deployment, operations, and maintenance) of the underlying infrastructure resources.

Software engineering processes that target higher level functionality are very different from traditional software engineering processes as they have to follow and to match the abstraction provided by the cloud. Most or even all aspects of system architecture, infrastructure resource creation and management, scaling, failure recovery, etc., do not have to be addressed by the software engineering process at all anymore. Instead, the focus will solely reside on the application's functionality, and how it will be structured, in terms of granularity, efficiency of algorithms, reuse techniques, etc.

In context of functions, for example, the granularity of functions will have to be decided on, how functions are implemented for reuse, common parameter data types, etc.

The cloud software engineering approach for PaaS is single pronged (compared to the two-pronged approach in case of IaaS). The only focus is functionality, and the artefacts to be developed by the software engineering process have to fit the target resources that the cloud PaaS implementation requires, e.g., functions, events, or subscriptions.

4.3 "Transitionary" Cloud Software Engineering (Mixed Case of IaaS and PaaS)

As long as clouds provide both, IaaS as well as PaaS target system resources, use cases might exist that require to address both. For example, it might be that a special database management system is required that no cloud provider makes available in their cloud by default as a PaaS service. In this example IaaS resources have to be created in order to install the special database management system.

The application functionality, however, can be implemented using PaaS resources like functions, for example. These functions are executed in the PaaS abstraction, while the special database management system is executed in the IaaS abstraction.

From a software engineering perspective the question arises if there is a separate software

engineering approach for the mixed case, or if the mixed cases are treated as two separate projects, one following the traditional process, and one the innovative process. This decision will reside in the eye of the beholder as both approaches are possible.

4.4 Prediction: PaaS–based Cloud Software Engineering Will Prevail

The DevOps movement will steer teams toward the functionality centric model of software engineering due to reduced operations and maintenance work, hence less opportunity for failure, errors or bugs. At the same time tools will be easier to use as infrastructure resources will not have to be made available for operations and maintenance.

At the same time an interesting question arises: is there any long-term value from a cloud provider's viewpoint to continue to provide the IaaS abstraction in addition to the PaaS abstraction? If the PaaS abstraction will provide all necessary functionality for software development in general then why providing IaaS resources?

The case of the special database management system resurfaces here that is not available in a cloud. One possibility is that a cloud provider enables the special database management software to be implemented within the cloud and made available as PaaS service instead. This approach would create an eco-system that allows 3rd parties (aka, companies different from the cloud provider) to provide services directly in the cloud of the cloud provider. From a customer's viewpoint there would be no distinction between functionality provided by the cloud provider itself or a 3rd party.

The prediction is: IaaS will disappear, and PaaS will become extremely expressive in terms of functionality, operations and maintenance.

5 RELATED WORK

The term cloud engineering was coined a while back (see (Wikipedia, 2018b) for details). However, the term based on searches did not take hold in academia or the industry – therefore, in the paper the term cloud software engineering was used.

In the special issue Software Engineering for the cloud (Grundy et al., 2012) the distinction between IaaS and PaaS is recognized, however, not in context of the software engineering process as such, only from a functional and architectural perspective in context of software architecture. No discussion takes place on how the software engineering process has to be specialized for the target system resources.

(Da Silva et al., 2012) goes a step further and discusses briefly that the artefacts are different for the IaaS target abstraction compared to the PaaS target abstraction. In this discussion, however, no implication to the software engineering process as such is discussed, nor the necessity of different software engineering approaches. The DevOps model is not considered in this discussion at all.

(Sommerville, 2012) goes even a step further and asks in the slide set if the cloud software engineering process is any different for the cloud compared to on-premise data centers. However, the question is never answered. And while the slide set makes the continued distinction between IaaS and PaaS from an architectural point of view, software engineering approaches for those are never discussed.

(Balalaie et al. 2016) describe experiences in their work migrating an application to a microservices architecture executing on a Kubernetes infrastructure (IaaS layer). Their section on lessons learned clearly shows the challenges in designing for an IaaS layer due to the visibility of the infrastructure elements.

(Kratzke et al. 2016) propose a cloud reference model (CloudNS) that assumes the visibility and accessibility of a complete stack, including IaaS. It recognizes different abstractions, however, it does not clearly outline the abstractions themselves. The position paper argues show that is is possible to build a service without having to manage resources. In CloudNS this would mean that layers 1 - 4 are not applicable, plus, that CloudNS layer 5 does not have resource management aspects. CloudNS would not be able to support the architecture of this position paper.

(Kratzke et al. 2017) contains a survey of engineering publications, however, based on their own opinion that a cloud application architecture must be microservices based the survey fails to include the category what is termed in this position paper as "Innovative Cloud Engineering".

The conference series titled "IEEE International Conference on Cloud Engineering" (IC2E, 2013 – 2107) does not contain any paper whatsoever that discusses the role of software engineering in the cloud and how it relates to the various layers of abstractions provided (aka, IaaS or PaaS).

6 CONCLUSIONS

The main conclusion is that a single cloud software engineering approach is insufficient in context of clouds that provide different layers of abstractions like IaaS or PaaS (or SaaS). Due to the difference in abstraction, separate customized cloud software engineering approaches are required that are tailored towards the target system abstractions the cloud layers provide, aka, IaaS vs. PaaS target system resources.

Over time it is expected that not all layers that are currently provided by clouds will have the same emphasis. One expectation is that PaaS abstractions will by far dominate the IaaS abstractions in the future and the latter might become relevant as niche only if it at all exist in the long run.

ACKNOWLEDMENT

I want to thank the reviewers whose review comments and suggestions improved the position paper.

REFERENCES

AWS, 2018. Amazon Web Services (AWS) - Cloud Computing Services. https://aws.amazon.com/ (last accessed 5/6/2018)

AWS DevOps, 2018. What is DevOps? https://aws.amazon.com/devops/what-is-devops/ (last accessed 5/6/2018)

AWS Reference Architecture, 2018. https://s3.amazonaws.com/awslambda-reference-architectures/web-app/lambda-refarch-webapp.pdf (last accessed 5/6/2018)

Balalaie, A., Heydarnoori, A. and Jamshidi, P., 2016. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software, 33(3), pp.42-52.*

Da Silva, E., Lucrédio, D., 2012. Software Engineering for the Cloud: a Research Roadmap. *2012 26th Brazilian Symposium on Software Engineering (SBES).*

Google, 2018. Google Cloud. https://cloud.google.com/ (last accessed 5/6/2018)

Grundy, J., Kaefer, G., Keong, J., Liu, A., 2012. Software Engineering for the Cloud. IEEE Software, March/April 2012

Helm, 2018. Helm. https://helm.sh/ (last accessed 5/6/2018)

IC2E, 2013 – 2017. IEEE International Conference on Cloud Engineering. https://dblp.uni-trier.de/db/conf/ic2e/ (last accessed 5/6/2018)

IEEE, 2017. ISO/IEC/IEEE 24765:2017(E) - ISO/IEC/IEEE International Standard - Systems and software engineering -- Vocabulary (https://standards.ieee.org/findstds/standard/24765-2017.html, last accessed 5/6/2018)

Kubernetes, 2018. Kubernetes. https://kubernetes.io/ (last accessed 5/6/2018)

Kubernetes CaaS, 2018. Containers as a Service, the foundation for next generation PaaS. https://kubernetes.io/blog/2017/02/caas-the-foundation-for-next-gen-paas/ (last accessed 6/24/2018)

Kratzke, N., Peinl, R., 2016, September. Clouns-a cloud-native application reference model for enterprise architects. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International (pp. 1-10). IEEE*

Kratzke, N., Quint, P.C., 2017. Understanding cloud-native applications after 10 years of cloud computing-A systematic mapping study. *Journal of Systems and Software, 126, pp.1-16.*

Lewis, J., Fowler, M., 2014. *Microservices*. https://www.martinfowler.com/articles/microservices.html (last accessed 6/24/2018)

Microsoft, 2018. *Microsoft Azure.* https://azure.microsoft.com/en-us/ (last accessed 5/6/2018)

Newman, S., 2014. Building Microservices. O'Reilly, 2015

Oracle, 2018. *Oracle Cloud.* https://cloud.oracle.com/home (last accessed 5/6/2018)

Oracle IaaS, 2018. *Oracle Cloud IaaS.* https://cloud.oracle.com/iaas (last accessed 5/6/2018)

Roberts, M., 2018. Serverless Architectures. https://www.martinfowler.com/articles/serverless.html (last accessed 6/24/2018)

Sommerville, I., 2012. *Challenges for Cloud Software Engineering*. Slide set, 2012. https://www.slideshare.net/sommervi/cloud-software-engineering (last accessed 5/6/2018)

Wikipedia, 2018a. *Supply Chain Management.* https://en.wikipedia.org/wiki/Supply_chain_management (last accessed 6/24/2018)

Wikipedia, 2018b. *Cloud Engineering*. https://en.wikipedia.org/wiki/Cloud_engineering (last accessed 5/6/2018)

DISCLAIMER

The views expressed here are my own and do not necessarily reflect the views of Oracle.