Benefits of Cloud Computing in the Online Printing Industry

Stefan Meissner [1]

[1] TU Dublin, Dublin, Ireland

Keywords: Cloud Computing Print, MassCustomization Print, Cloud-based Print Production, IT-Strategy Print Production

Abstract
The online printing business is moving towards mass customization. This research project is aimed to be an essential research for the online printing industry in order to underpin this progression. The keys to mass customization are 'Economies of Scale' and 'Economies of Scope'. This paper elaborates on the backgrounds and leverages of mass customization in the online printing industry and investigates how the industry can benefit from Cloud Computing. Cloud Computing is a direction in Information Technology (I.T.) that brings applications from local installations into large centralized data centres in order to save costs, raise flexibility, stability, and sustainability. This publication is conducted by the research question of how to go about adopting Cloud Computing technologies and architectures in order to design an economical and sustainable Information Technology (I.T.) infrastructure for the online printing industry? The outcome of this paper is an evaluated design pattern of how to build and manage cloud services for production lines of online printing companies using cloud computing technologies. The evaluation and theory extraction is driven by the development of a preview generation cloud service that generates preview images from PDF files. All sources and artifacts of that service are published on GitHub.com and are referenced and outlined by this document.

Introduction

When someone asks people about the printing industry, their first image is most likely about the heavy printing presses, many piles (or webs) of paper, or barrels of ink. These are the typical characteristics of which printing houses are usually known. However, presses are being developed and installed by press vendors, while suppliers deliver the consumables print-ready. So, the function of printing houses is to bring all together in order to reproduce customers artworks visually onto substrates. Today, customers normally deliver artwork files digitally. Once received and processed, printing presses consume digital data directly or imposed on printing plates. The nature of printing has changed significantly towards Information Technologies (I.T.) over the last decades. The industry has digitalized many parts of the printing process. Examples include order management and routing, ganging, pre-flight checks, colour management, rasterization, imposition, generation of pre-settings for press and finishing, production data acquisition. The number of file formats involved in the print production process has also increased considerably. File formats include Portable Document Format (.pdf), Job Definition Format (.jdf), Print Production Format (.ppf), Tagged Image File Format (.tiff), International Color Consortium Profile (.icc) etc. The number of print specific file formats is an indicator of how software and I.T. influences the printing industry. This is particularly relevant in the online printing industry segment, where integration and automation are major keys,

printing providers are transforming increasingly more towards software development companies - specialized in printing.

This research project is aimed to be an essential research for the online printing industry in order to underpin this progression. Along the technological and structural development in the printing industry, so too the I.T. business has experienced notable evolutions. Many new technologies, concepts, and architectures have emerged over the last decade. One of the most critical topics that appeared in this field is most likely Cloud Computing. When developing highly integrated and automated print production systems, a fundamental challenge is to define the technological and architectural I.T. framework used for that system. Print production systems are usually complex, as many people, legacy systems, devices, and third-party applications are involved. Therefore, the focus of the I.T. framework must be on sustainability, as complex systems are typically slow-acting. Sustainability implies that a technology is supported by a broad community, rather than from a single vendor only. Furthermore, the technology shall be approved by commonly known real-world projects to have shared experiences, best practices, and experts by the hand. The author's objective is to explore the frontiers of the technologies and architecture in both the printing industry and Information Technologies. The result of the publication is the definition and evaluation of an I.T. framework, which enables online printing providers to design sustainable and economic I.T. systems in order to increase their level of integration and automation. This publication addresses the following research question: **How to adapt Cloud Computing technologies and architectures in order to design an economical and sustainable Information Technology (I.T.) infrastructure for the online printing industry?**

Central to this, and supporting this research is the prototyping of a cloud service for the generation of preview images. The prototype is designed as a minimal viable product (MVP) and is used for the definition and evaluation of the I.T. framework. Ries (2011) has defined the concept of a minimal viable product (MVP) as "a version of a new product, which allows a team to collect the maximum amount of validated learning about customers with the least effort." Lenarduzzi and Taibi (2016) entitle the MVP as one of the most critical steps of the Lean Startup methodology, needed to start the learning process by integrating the early adopters' feedback as soon as possible. Acknowledging this, Lenarduzzi and Taibi (2016) outline in their publication "MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product" how broadly the MVP concept has been adapted, interpreted and further developed over time. Maurya (2017), refined the MVP definition and set the primary focus on the minimal viability: "A Minimum Viable Product is the smallest thing you can build that delivers customer value (and as a bonus captures some of that value back)." This publication will follow the stricter definition by Maurya: The prototype developed in the context of this publication focuses on a simple preview generation cloud service and requires to be usable for everybody. Based on that prototype, the direction of research and theory extraction is being conducted. Further, the prototype is used for evaluation purposes and to ensure the validity of this publication.

This research is following an inductive approach. Fisher (2010, p. 109) defines induction when a conclusion is drawn from experience or experimentation. In contrast to deduction,

where the conclusion follows in logic the premises stated in the beginning, he mentions a risk that inductive arguments may base more on wishful thinking than on a carefully judged balance of probabilities. This publication has a particular focus on that risk in order to provide an objective view. Fisher (2010, p. 149) further argues that theories are an attempt to generalize the findings from specific instances. Supporting this, Bryman and Bell (2015, p. 19) stated that the inductive approach is where the theory is an outcome of the research. The research strategy applied in this paper is therefore clearly a qualitative one. Bryman and Bell (2015, p. 38) observe a predominant relationship between the inductive approach and the qualitative research strategy, as the emphasis is placed on the generation of theories. The theory generation is also the focus of this publication. The research design of the publication will be a case study design. Fisher (2010, p. 69) differentiates the case study as a design pattern which focuses on an in-depth understanding of a situation. Further, he stated, case studies are giving a holistic account of the subject of the research by elaborating on the interrelation between factors such as people, groups, policies, technologies. According to Bryman and Bell (2015, p. 67), the case study entails the detailed and intensive analysis of a single case. This design pattern is a widely used one in business research. In contrast to other research designs, a case study focuses on a bounded situation or system. As examples, Bryman and Bell (2015, p. 67) list a single organization, a single location, a person, or a single event. The case treated by this publication covers a single cloud service for the generation of preview images.

The publication's scope covers the I.T. framework for the development of a prepress cloud service as evaluation as well as the deployment of the service in a managed environment. The document introduces the deployment models of cloud environments; however, for the evaluation, the publication assumes that a cloud environment does exist and is ready to orchestrate services. Chifor (2017) refers to orchestration to the automated arrangement, coordination, and management of services. Sun (2015) acknowledges Chifor and states that orchestration allows users to coordinate services in a cloud. Orchestration includes not only the deployment, but also the management, such as availability, scaling, and networking. This publication focuses on the development of services, rather than the setup and operation of a cloud environment. Cloud environments can be provided and managed by both experts within a company or on a pay-per-use basis from public cloud providers. This publication applies to the latter solution. Although the publication does not cover cloud environments in detail, it considers design criteria for services, which let them seamlessly integrate into any cloud environment.

The envisaged audience of this document is the technician and technical manager operating in the online printing sector. The document is structured in four content chapters – excluding the 'Introduction'. The first chapter, 'Printing Technologies', analyses the technical and economic trends and backgrounds, the production volumes, the technical state of the art, as well as I.T. challenges in the printing industry. The second chapter, 'Cloud Technologies and Architectures', gives a technological and architectural overview in Cloud Computing and finishes with a technical concept, best practices, and recommendations of how to build cloud services. 'Development of a Preview Generation Service' is the third chapter. This chapter is the summary of the preview generation service development

and contains the theory evaluations. It lifts out the implementation of the primary design criteria elaborated in the second chapter. Further, the chapter refers to the source code, and the cloud service live demo as well as the interface specification of the service. The final chapter, 'Conclusion', contains the lessons learned during the evaluation as well as recommendations of an I.T. framework of how to use Cloud Computing technologies and architectures in the online printing industry.

Printing Technologies

The online printing business is moving towards mass customization. According to Pine (1993, pp. 48-49), Mass Customization is the derivative of Mass Production. Whereas Mass Production focuses on the mass production of standardized products, Mass Customization refers to mass production of customized or even individualized goods. Both of these methods have the objective of minimizing the per-unit cost. Mass Production achieves the low unit cost primarily by way of increasing the output of the single standardized product (Economies of Scale). At the same time, Mass Customization besides raises the product variance of a single production process (Economies of Scope). Fogliatto et al. (2012) observed that Mass Customization had become the dominant production form in many markets. König (2013) sees a fundamental change in commercial printing since the online procurement of print products. She referred to this emerging process as Mass Customization as the process enables customers to individualize standard products with their content. Nowadays, Mass Customization continues to be an essential topic in the online print business. Cimpress (2019 a), for example, a sizeable online print provider, announced on their strategic website that they are "building and deploying an increasing number of modular, multi-tenant micro-services and technologies as a mass customization platform." Mass Customization is also a strategy employed at Flyeralarm, another sizeable online print provider. Flyeralarm's founder Fischer (2019), stated in an interview that Flyeralarm "stands for standardization and 'mass customization'." Today's online printing business continues to focus on the customization of standard products as König has already described in 2013. As advances in process integrations continue to be made, maybe once real individual products being available as opposed to customized standard products.

We must clarify and understand what is the meaning of 'mass' as it pertains to Mass Customization regarding the printing industry? An analysis of major online printing platforms such as Vistaprint, Flyeralarm, WIRmachenDRUCK, Onlineprinters, or Saxoprint reveals a broad range of pre-defined standard print products, including (folded) leaflets, magazines, posters, letterheads, business cards, flags, textiles, packaging, cups. With regard to the magnitude of print jobs, Flyeralarm (2019) states that its production is up to 15,000 orders per day and has generated a revenue of 350 million Euros in 2018. Dividing the €350MN by 250 working days and 15,000 jobs per day, the result is an average job value of €93.33. Assuming an average job value of €100 and 250 working days per year provides a rough approximation of the number of jobs produced by online print providers based on their annual turnover:

Table 1: Calculated number of jobs per print provider in 2018

	Turnover [€]	Jobs per year	Jobs per working day
Vistaprint	1,250 m *(Cimpress, 2019, p.31)*	12.5 m jobs*	50,000 jobs*
Flyeralarm	350 m *(flyeralarm, 2019)*	3.5 m jobs*	14,000 jobs*
Online-printers	200 m *(Onlineprinters, 2019)*	2.0 m jobs*	8,000 jobs*
WIR-machen-DRUCK	150 m *(WIRmachen-DRUCK, 2019)*	1.5 m jobs*	6,000 jobs*
Saxoprint	100 m *(CEWE, 2018)*	1.0 m jobs*	4,000 jobs*

** Calculated values based on 100 Euros average job value and 250 working days*

This table aims not to represent the exact figures of the listed companies, but rather to provide a rough approximation of the volume of printing produced by these companies. Nevertheless, Saxoprint (2019) has stated on their website that they are processing more than 4,000 jobs daily – a figure that corresponds to the estimate of Saxoprint's printing activity in the table above. The analysis reveals that the term 'mass' in the context of the online printing business ranges from several thousand jobs per day, up to several tens of thousands of jobs.
As previously mentioned, Mass Customization is a progression of Mass Production. Therefore, principles of Mass Production such as 'Economies of Scale' are also relevant in Mass Customization. Silberston (1972) describes 'Economies of Scale' as the effect on production costs concerning different rates of output. He argues that high volumes allow enterprises to invest in technology and expertise. These investments will further lower the production costs - but will never be economical for small production volumes. As a result, the higher the output of a production line, the lower the cost per unit. Pine (1993, pp. 16-17) writes about a self-enforcing cycle in mass production whereby low prices lead to an increase in demand, resulting in a further increase in the production volume, which once again lowers unit costs. Pine elaborates that this cycle could even be reinforced by standardizing products. Standardized products avoid complexities and customized work, which in turn increases the output and lowers the unit costs. Product standardization is a feature that can also be observed in the on-line print business. Providers such as Flyeralarm, Vistaprint provide only a limited selection of paper types, product sizes, and colors. According to Keane (CEO, Vistaprint), this product standardization is patented (United States Patent No. US 6,650,433 B1, 2003). The self-enforcing cycle of mass production can also be found in online print businesses. The dominating companies are typically emerging firms and new industry entrants which exhibit fast growth.

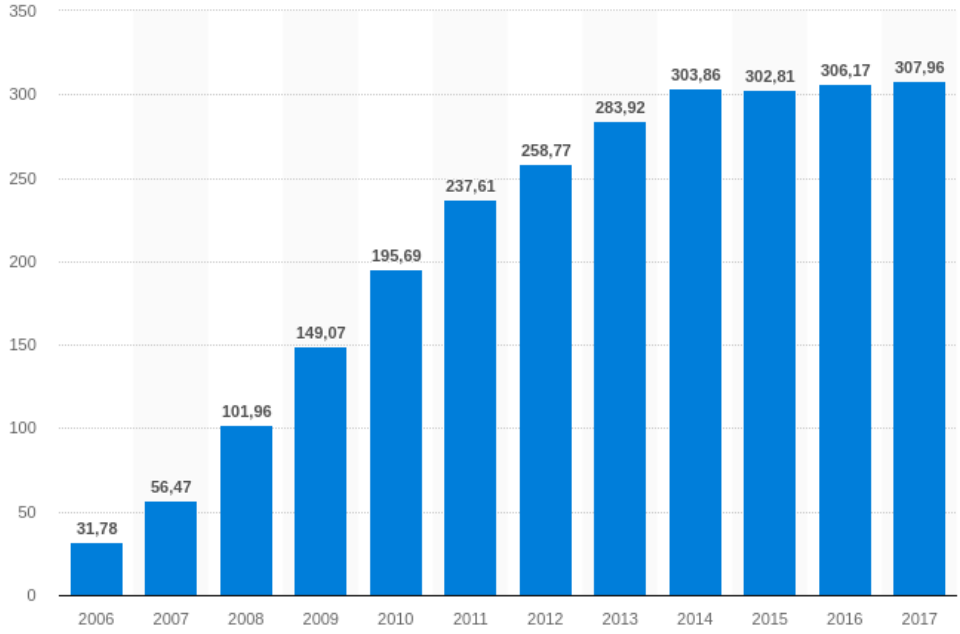

Figure 1: Revenue Flyeralarm in Euro (Statista, 2019)

Flyeralarm, for instance, was founded in 2002. After just ten years in business, Flyeralarm had a revenue of €260mn. Such printers produce on large and highly specialized devices, which would never be affordable nor economical for small printers (Flyeralarm, 2017). Besides, when analyzing their career portals, they hire experts who are exclusively responsible for the process optimization and automation, so that costs can be decreased.

'Economies of Scope' may be the key to mass customization. Panzar and Willig (1981) define 'Economies of Scope' as cost savings within an enterprise achieved by the combination of multiple production lines bundled into a single production line, due to shareable inputs. They argue that, as soon as shareable inputs are implemented in multiple product lines, the costs function exhibits 'sub-additive' behaviour. Panzar and Willing refer to all kinds of resources needed for production (material, device, applications, even buildings) as 'input'. Pine (1993, p. 196) sees 'Economies of Scope' "as the best method to achieve mass customization," as it focuses on the creation of modular components that can be configured into a wide variety of end products. One method of applying 'Economies of Scope' in printing is known as ganging. Keane et al. (United States Patent No. US 6,650,433 B1, 2003) have examined ganging as the aggregation of multiple print jobs such as business cards, folders, brochures, in order to produce them all in a single print run, resulting in production costs savings. The CIP4 Consortium (CIP4, 2018, p. 70) refers to the 'ganging process' as 'Sheet Optimization'. Other forms of shared inputs in print production lines include prepress processes. The XJDF Specification (CIP4, 2018, p. 57) defines all processes

performed before printing as Prepress Processes. Most of the Prepress Processes consist of services that check, correct, and optimize the customer's artwork data, as well as prepare the digital printing form. Examples include Pre-flight, Colour-Correction, Image-Enhancement, Imposition and Trapping (CIP4, 2018, pp. 77-93). If prepress services exist as modular components, they can be used as shared inputs between multiple print production lines.
When sharing prepress services between production lines, each service shall work independently, abstracted by an Application Programming Interface (API) for integration. Orenstein (2000) describes APIs as the proper way of how one application consumes a service provided by another. Reddy (2011) espouses a more centralistic view of APIs: He describes an API as being "an abstraction for a problem and specifies how clients should interact with software components that implement a solution to that problem." Reddy's perspective considers APIs more as a self-contained component rather than just an interface of a service. Zalando (2019 a) supports Reddy's view and goes even further: They want developers to embrace APIs as independent products. This approach "facilitates a service ecosystem that can be evolved more easily and used to experiment quickly with new business ideas by recombining core capabilities." APIs have assumed a central role in contemporary systems. So, too, at Amazon: Jeff Bezos, Amazon's CEO, already issued an internal mandate in 2002 in which he strictly instructed all teams to communicate exclusively by way of APIs in order to make the company scalable (Kim, 2017). Amazon has, therefore, become one success story for shared inputs and 'Economies of Scope'. From a more general perspective, Mass Customization and APIs are closely related to each other, as APIs ensure the reusability of services. Concerning Mass Customization in the printing industry, this means that precise API specifications of prepress services are a prerequisite for making them shareable between production lines. Interoperability Conformance Specifications (or simply 'ICS Documents') are the API specifications of the Graphic Arts Industry. ICS Documents are self-contained specifications which are sub-classed from a master specification for a particular use case. The International Colour Consortium (ICC, 2019), for instance, is the publisher of the iccMAX specification, which specifies an extended colour management system. One use case of an ICS Document in that field is the ICS for 'Spot Colour Overprint Simulation', which specifies the visualization and simulation of ink overprints. Whereas the ICC standardizes the colour management, CIP4's mission is the standardization of the process automation in the printing industry (CIP4, 2019 a). In order to achieve this, CIP4 maintains various specifications, including the Exchange Job Definition format (XJDF) and the Job Definition Format (JDF). CIP4 also follows the approach of Interoperability Conformance Specifications, as previously described (CIP4, 2018, p. 10). An example of a JDF based ICS Document is the 'MIS to Conventional Printing ICS', which is a subset of the JDF Specification and defines the interoperability requirements between a Management Information System (MIS) and a sheet-fed offset press (CIP4, 2015). The focus of a master specification is to standardize the communication concept as well as providing an integrated data model describing all aspects of an entire field at the right level of granularity. Interoperability Conformance Specifications elaborate one use case and specify which aspects of the master specification are relevant in which situations and which one not (Meissner, 2019). In the context of shareable prepress services, this means that each kind of service

requires an individual ICS Document specifying how to communicate and which data must be exchanged.

When dealing with modularised and shareable services that are abstracted by an API, next, these services need to be deployed in a productive environment. As previously explained, Mass Customization means to process up to several tens of thousands of jobs per day. Besides, these jobs typically do not occur consistently in the run of a day. In such an environment, flexibility, scalability, and reliability of shared services are essential. According to Dikaiakos et al. (2009), Cloud Computing is a direction in Information Technology that brings applications from local installations into large centralized data centres. This approach would not only raise the application's scalability and reliability, but it would also reduce the costs. Islam et al. (2013) parallels the emerging interest in Cloud Computing on the part of businesses and individuals. They argue that Cloud Computing comes with a new service-centric technology that fosters business agility and the quality of services. Moreover, they also see the cost optimization of Cloud Computing. The remainder of this publication subsequently addresses the following research question: **How to adapt Cloud Computing technologies and architectures in order to design an economical and sustainable Information Technology (I.T.) infrastructure for the online printing industry?**

Cloud Technologies and Architectures

The term 'Cloud Computing' (or 'The Cloud') refers to a highly scalable and reliable on-demand Information Technology (I.T.) Infrastructure. Cloud Computing includes the physical hardware as well as the software running on it. The U.S. National Institute of Standards and Technology (Mell & Grance, 2011, p. 2) has determined Cloud Computing to be "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." Armbrust et al. (2010, p. 50) have defined Cloud Computing similarly, but from a more service-oriented perspective. They see Cloud Computing as applications delivered as services over the Internet, as well as the hardware and systems software of the datacentre providing these services. Whereas Armbrust et al.'s definition explicitly sees a relationship between Cloud Computing and services provided over the Internet, Mell & Grance (2011) do not. Instead, Mell & Grance (2011, p. 3) describe an on-premise private Deployment Model of Cloud Computing. The service-oriented perspective is, in fact, key in Cloud Computing. The service concept should not be limited to the (final) service applications only. However, hardware- and system management underneath might also be considered as services – albeit on a lower abstraction level.
Services in the context of Cloud Computing provide functionality not only to the end-user but also to developers who are responsible for developing and operating such services. A (final) service providing functionality to the

end-user is typically based on a set of lower-level services that abstract the cloud infrastructure underneath. This method enables developers to provide higher-quality software in less time. Those abstraction levels of services are commonly referred to as Service Models. Gibson et al. (2012) maintained that infrastructure-, platform-, and software-as-a-service are the predominant Service Models. They have classified Infrastructure-as-a-Service (IaaS) as the management of servers, storage and virtualization, and Platform-as-a-Service (PaaS) as a middleware allowing developers to write applications without the requirement of a more in-depth knowledge of the underlying infrastructure. Software-as-a-Service (SaaS) they have classified as applications that reside in the cloud rather than on the user's device. A similar classification has also been examined by Mell & Grance (2011, pp. 2-3). They have defined IaaS as the capability to provide processing, storage, networking, and other fundamental computing resources, along with PaaS, as a means of deploying applications onto to cloud without knowledge of the cloud infrastructure underneath. SaaS is described by Mell & Trance (2011, p. 2) as having the capability to consume applications deployed to the cloud infrastructure. Foulds (2018, pp. 10-11) also supports the three Service Model classifications (IaaS, PaaS, SaaS) as previously outlined:

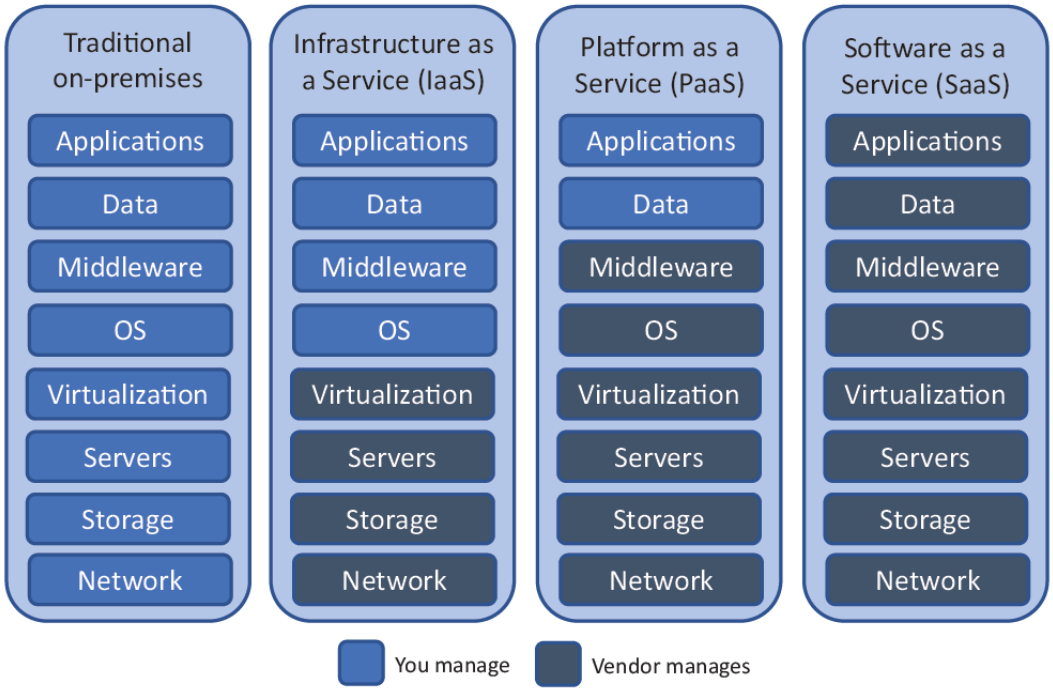

Figure 2: Cloud computing services models (Foulds, 2018, p.11)

This illustration by Foulds (2018, p. 11) depicts four columns - one for each Service Model. Each column represents the identical technology stack needed to provide a service application (SaaS). Each box in a column represents a component of the technology stack. The components are ordered bottom-up, by abstraction level. The background colour of each box indicates who is responsible for managing this component in the appropriate Service Model. Blue background colour indicates management by the consumer (referred to as "you"), while dark-blue refers to management via the provider (referred to as "vendor"). In addition to the three previously discussed Service Models, Foulds (2018, p. 11) has also visualized the Service Model "Traditional on-premises," which in this context means that no cloud infrastructure is used at all – everything is managed by the consumer ("you"). It seems that there is a common understanding of IaaS, PaaS, and SaaS among developers and cloud providers. However, Foulds' (2018) Service Model "Traditional on-premises" can be supported in the context of his book. However, from a general perspective, this Service Model contradicts the concept of Deployment Models of clouds.

Generally, one can differentiate between two theoretical manifestations of Deployment Models in Cloud Computing: public and private. A public Deployment Model is a cloud infrastructure that is hosted by a cloud provider and is used by (many) tenants. In contrast, the private Deployment Model is a cloud infrastructure that is hosted on a company's internal I.T. infrastructure and is primarily targeted for its exclusive use. Keung & Kwok (2012, p. 21) support this differentiation between public and private cloud infrastructure in the same way. They present in their paper a score-based Cloud Deployment Selection Model (CDSM), which helps small and medium-sized enterprises (SME) to choose the right Deployment Model, should they opt to implement Cloud Computing. Unfortunately, Keung & Kwok's (2012) perspective recognizes only public and private Deployment Models. They do not consider the concept of hybrid cloud infrastructure, a mix of both, and probably the most suitable Deployment Model for most companies. Mell & Grance's (2011, p. 3) standard document also conforms to the definition of public and private Deployment Models as previously introduced in this paragraph. Also, they have defined the hybrid cloud Deployment Model as a combination of both public and private. Weinman (2016, pp. 7-8) has published case studies that give insight into the cloud strategies of some commonly known companies. He states that there are as many cloud strategies as there are companies. However, the case studies of Weinman reveal that most companies work with a hybrid cloud - either because of their I.T. strategy, or because of an active migration process from private to public cloud and vice versa.

Containerization appears to be the superior technique when developing cloud applications. Conceptually, containerization is another Service Model which is in the abstraction level between Infrastructure-as-a-Service and Platform-as-a-Service:

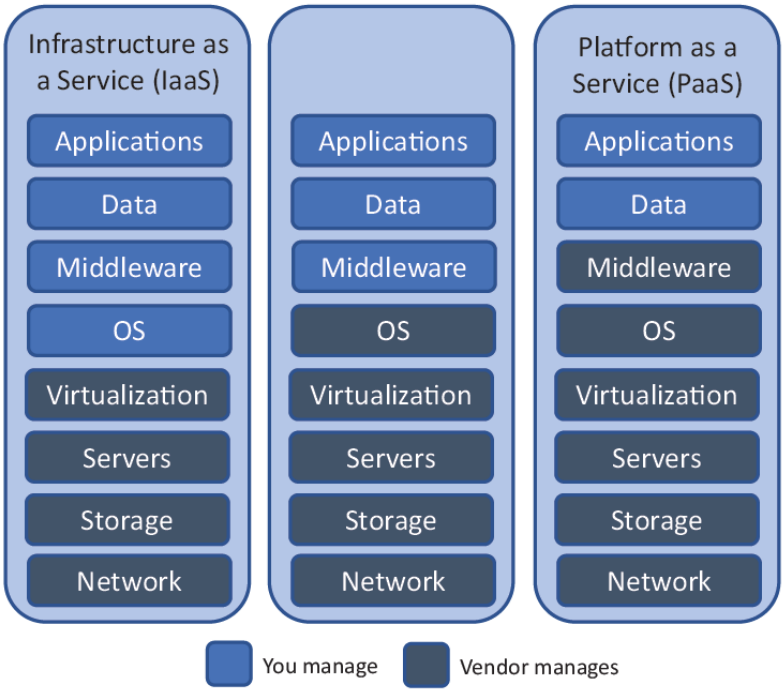

Figure 3: Container-as-a-service (Foulds, 2018, p.11)

Figure 3 is a modification of Foulds' (2018, p. 11) illustration of Cloud Computing services models (Figure 2). In Figure 3, a new column appears between IaaS and PaaS in which the vendor manages everything below and including the operating system (O.S.). Both virtualized machines (IaaS) as well as containers presenting an O.S. interface to the developer, according to Bernstein (2014, p. 83). However - whereas IaaS requires a complete implementation of the O.S. - containers only gives a "view", or a "slice" of the host O.S.. Bernstein (2014, p. 82) further states that, as containers share a common host O.S., the deployments are significantly smaller in size than are IaaS deployments. This approach enables developers to store hundreds of containers on a single physical host. Another advantage of the shared O.S. is that (re-) starting a container does not (re-) start an entire O.S. This makes containers flexible and easy to handle in contrast to virtual machines. Docker (2019), a leading company in containerization, conforms to Bernstein's explanations. Beyond that, Docker (2019) describes a container as a standard unit of software that packages up code and all its dependencies so that an application can run quickly and reliably on a broad range of computing environments. Many major cloud providers provide I.T. infrastructure to host and execute containers. Examples include Amazon's Elastic Container Service for Kubernetes (Amazon EKS), Google's Kubernetes Engine, or Microsoft's Azure Kubernetes Service (AKS). However, containers can also be executed on a local machine or within a company's private network. It seems that containerization

has become the de-facto standard in Cloud Computing, as it is broadly supported and implemented.

Containerization has become a notable direction in I.T. development, and it seems that Docker Container Runtime is the de-facto standard. Sysdig (2018), has published a study on containerization based on a sample of itemized information of 90,000 productive containers. The information was from "a broad cross-section of vertical industries and companies ranging in size from mid-market to large enterprises across North America, Latin America, EMEA, and Asia Pacific." In this study, they observed a "tremendous momentum across the Docker container ecosystem year-over-year." Additionally, Sysdig (2018) also notes the dominance of Docker containers in 2018:

Figure 4: Container Runtimes (Susdig, 2018)

The figure above illustrates the market shares of the most common container runtimes: The Docker container runtime leads with 83 %, followed by CoreOS RKT (12 %), Mesos Containerizer (4 %), and Linux Containers LXC (1 %). However, according to Sysdig (2018), the market share for the Docker container runtime decreased from 99 % of the market in 2017. Sysdig's (2018) assertion regarding a "tremendous momentum across the Docker container ecosystem year-over-year" can be underpinned by Google Trends (2019) (Figure 5).

The Google Trends (2019) diagram indicates a continuous uptrend for the search term 'docker' since 2013. Regarding the market shares of container runtimes, there is no other source for affirmation. Nonetheless, the dominance of docker containers becomes conspicuous as soon as one carries out further research into containerization. The container runtime supported by apparently all public cloud providers remain Docker.

In contrast to traditional computers and virtual machines, containers are stateless and immutable and only contains a single application. They are stateless means that any state of persistent data must be stored outside the container in a database, or some other type of external storage. Statelessness ensures that containers can be replicated or destroyed at any time without the fear of data loss. Immutability means that, once created, a container cannot be

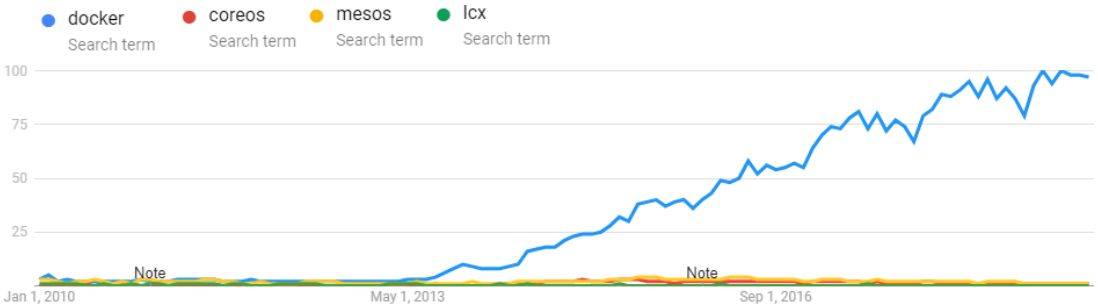

Figure 5: Trend of Search Terms 'docker', 'coreos', 'mesos', 'lcx' (Google Trends, 2019)

updated, patched, or reconfigured. Each modification of a container requires a new version. This approach ensures that a container version is almost identical - independent of the cloud environment it is executed. Designing containers as stateless, immutable single applications maximizes their flexibility during execution. This method parallels Google's explanations of the concept of containers in their articles "Best Practices for Operating Containers" (Google Cloud, 2019 b) and "Best practices for building containers" (Google Cloud, 2019 a). Docker Inc. (2019 a) simply elaborates on design criteria, explaining that containers should be as ephemeral as possible while referencing the chapter "VI. Processes" of Wiggins (2017) publication "The Twelve Factors" regarding a methodology for building software-as-a-service apps. In that chapter Wiggins (2017) states that "Twelve-factor processes are stateless and share-nothing." In explaining this opinion, Wiggins states that all persistent data must be stored in a stateful backing service such as a database, and any cached data must never be assumed on future executions. Wiggins' statement is substantially equivalent to that of Googles regarding statelessness and immutability. The meaning is the same, but from slightly different perspectives. Another design criterion pertains to logging. When running applications in containers, all log information must be provided via standard output (stdout) and standard error (stderr). Logs make applications replicable during runtime by recording all notable events that occur in combination with metadata, such as the timestamp or the process id. The traditional means of logging is that applications record logs in local text files. The use of stdout and stderr methods allows container runtimes to continually collect all container logs during execution using a unified approach and provide them, if required. As a result processes are conducted in a simplified and comprehensive structure. Subsequently, processes may include a centralized log file retention or an automated log file analysis. The article "Best Practices for Operating Containers" (Google Cloud, 2019 b) acknowledges the method of writing log information to stdout and stderr by explicitly advising implementers to use that container-native logging mechanisms. Furthermore, Google considers 'logging' to be an integral part of application management as "logs contain precious information about the events that happen in the application" (Google Cloud, 2019 b). Hromis (2019) also supports the importance of logging and logging centralization. Hromis deals with approaches on how to centralize logging using Docker Log Drivers: Depending on the type of central log management system and the preferred way of transmission, a different kind of log driver is required. However, though log drivers may exhibit apparent differences, the collective input of all log drivers is the collected log information written by the managed containers in their appropriate stdout and stderr.

Containerization also affects application architecture with regards to microservices. Balalaie et al. (2016, p. 42) describe the Microservice Architecture as a cloud-native architecture that aims to design systems as individual and independent services. The communication between these services is based on lightweight standard API technologies such as RESTful (Representational State Transfer) or RPC (Remote Procedure Call). By the use of containerization, each microservice will be conceptually represented by a single container. According to its load, each container (microservice) can selectively be executed once or multiple times simultaneously. As the docker container format is broadly supported, a high degree of portability can be ensured (Balalaie, Heydarnoori,

& Jamshidi, 2016, p. 46). Singh and Peddoju (2017) observed the growing popularity of the microservice design in cloud applications. They justify that trend as microservices allow selective scaling of single tasks (services) according to their demand in contrast to monolithic applications, which can only be scaled in it's entirety. Additionally, they explain that microservices increase the systems flexibility and lower costs. As scalability is one key feature in Cloud Computing, it seems suitable to split up medium and large sized applications into microservices. However, small applications potentially should be provided as containers, but following the monolithic approach.

Singleton (2016, p. 16) sees microservices as perhaps the only solution to build and manage complex software systems as he sees a conceptual limit to the number of functions one team can specify, test, and maintain. In contrast to traditional monolithic architectures, where all system services consist of a joint code base and are tightly coupled, microservices are decoupled services. Each one has its independent code base and might use an individual technology stack (Singh & Peddoju, 2017, p. 852). Further, Singh and Peddoju (2017, p. 852) elaborate on microservices that the granular design pattern enables developers to work simultaneously on different services. This method results in better scaling and increased flexibility of enterprise applications. Singleton (2016, p. 18) further addressed the release frequency and agility of microservices in contrast to that of monolithic applications. Due to their complexity, monolithic applications usually require extensive test- and release times (up to several months and more). In contrast, microservices would be able to be tested and released often and frequently, even multiple times daily. Nevertheless, Singleton (2016, p. 17) argues that the microservice architecture comes with substantial costs as additional machinery for communication, routing, deployment, and monitoring of service is required. He therefore recommends microservices only for large systems. However, cloud infrastructure has continued to evolve since the publication of Singleton's 2016 paper. Today the substantial costs Singleton mentioned are certainly on a more economical level and should, therefore, be reconsidered for each specific use case. Chelladhurai et al. (2016) have examined security issues regarding containerization. The main advantage of containerization - the sharing of the host's kernel – is, at the same time, the weak point of the technology. In contrast to virtualization, containerization has no extra isolation layer between the applications and the host, which prevents the host from an escalation out of a compromised container. Once an attacker has access to the host, one might perform attacks such as denial of service (DoS) or privilege escalation. Chelladhurai et al. (2016) have listed security enhancements implemented by the container runtime providers to secure the system. These enhancements show that security benefits from keeping the container runtime up to date with the most recent version. System providers, for instance of Docker Swarm or Kubernetes, have guidelines that dictate the announcement and publishing of essential security updates. In addition to containers runtime security, Chelladhurai, Chelliah, and Kumar (2016) also suggests design criteria for containers to make them more secure. The first is to minimize the number of binaries and services running in a container as a means of decreasing the containers attack surface. Another criterion is to design containers being executed with a read-only file system. The read-only file system shields a container from being manipulated. These design criteria are also acknowledged

by Google Cloud's (2019 a) best practice guide in which Google advises removing needless tools to reduce the attack surface. Google also recommends using read-only file systems in order to prevent attackers from installing their tools. According to Google, this protection can be enforced by avoiding running applications inside containers with root privileges. Chelladhurai et al. (2016) also write about resource restrictions such as limiting kernel calls, memory, CPUs, and network access. These resource optimizations are not listed in Google Cloud's best practice guide. When creating an ordinary container, the security-specific design criteria listed above appear to be sufficient. However, if security is crucial, one should delve deeper into the resource-specific optimizations.

Development of a Preview Generation Service

The generation of preview images is a typical process implemented in most print production lines. Preview images represent a visualization of content on a surface and are used as input resources for various applications. Applications include, for instance, ink coverage calculation or providing a visual representation of what being produced (CIP4, 2018, p. 209). As preview generation is a broadly used but also an easy definable process, it is a good one to be picked up for evaluation purposes of the concepts, architectures, and techniques described in the previous chapters. This chapter will consider the development of a preview generation cloud service as an evaluation and proof-of-concept of cloud computing in the printing industry. The preview service is designed as a stateless Docker Container providing both an XJDF based Application Programming Interface (API) for the automated generation of preview images as well as a graphical user interface for the manual interaction with the service. The service consumes PDF Documents and produces PNG preview images of the first page of the PDF Document in the desired resolution (default: 72 dpi). The service is implemented against an Interoperability Conformance Specification (ICS), which abstracts the business logic of the service. All project sources, as well as the ICS Document, are available on github.com: https://github.com/ricebean-net/PreviewService. The latest version Docker container image is built and published continuously on Docker Hub: https://hub.docker.com/r/ricebean/preview-service. One live test instance of the preview service is running at https://preview-service.ricebean.net.

Application Programming Interfaces (APIs) are a significant key when building distributed systems based on scalable (Economics of Scale) and configurable (Economics of Scope) services. Recall, APIs abstract and decouple services in a way so that they can be reused in multiple production lines and even in various applications. Besides, precise API specifications allow to implement a service in a production line, even when the service self has not (entirely) implemented yet. This kind of abstraction enables development teams to work in parallel, even if they depend on each other. Lin (2018) and Trieloff (2017) promote, therefore, an 'API First' development approach, which requires to specify the API before starting any writing of code. Many development teams broadly support the 'API First' strategy – so too at Zalando (2019 b). Within the context of the development of the preview generation service, also the 'API First' development approach has been chosen. Remembrance, Interoperability Conformance Specification (ICS Documents) are the API specifications in the printing industry. Fol-

lowing the 'API First' approach means that the ICS Document defining the preview service's API must completely be specified before the implementation of the service can start.

This publication follows the CIP4's guidelines of Interoperability Conformance Specifications (ICS Document). The consortium provides a set of standardized ICS Documents for the printing industry based on JDF (CIP4, 2019 b) and XJDF (CIP4, 2019 c). CIP4 does not provide an official guide on how to write ICS Documents. However, when analyzing existing ICS Documents, a consistent structure emerges: An ICS Document differentiates between the two roles 'Manager' and 'Worker'. 'Worker' relates to the service provider (here: preview service), whereas 'Manager' is the service's consumer. Recall, ICS Documents subset the master specification. The core of an ICS Document is a precise list of elements and attributes, initially defined by the master specification, which are relevant in the use case described by the document. ICS Documents further define which of these elements and attributes are mandatory, conditional required, optional, or even prohibit to read or write by the appropriate roles. In contrast to CIP4's ICS Documents, which are published as PDF, the Previews Services ICS has been published as a 'GitHub Markdown' document (GitHub, 2019): https://git.io/JeSGG. Even Markdown should not be the preferred way of distribution of ICS Documents; within this publication, it is reasonable, as it is easy to read and modify.

Once the Interoperability Conformance Specification of the preview service is finished, the implementation of the preview service can begin. The technology stack used in order to implement the service is usually up to the developer or the development team, as long as the requirements defined in the ICS Document are satisfied. This principle is precisely one meaning of "abstraction for a problem," as previously cited Reddy (2011). The technology stack used within this publication is based on Java, as the author is most experienced in this technology. Java is a standard programming language maintained by Oracle, which is broadly used in enterprise- and web-applications as well as in cloud computing (Oracle, 2019). The core of the author's implementation is a third-party command-line tool called ImageMagick. This ImageMagick (2019) tool is a free, open-source utility that includes functionality for image format conversion as well as for resizing of images. The Java application provides the endpoints as required by the ICS Document and is responsible for the internal file and configuration handling. All image related work is internally forwarded to ImageMagick. ImageMagick may be considered to be the PDF Engine of the preview service. Although ImageMagick is a powerful tool, it shall never be used to create or modify the print files because it is not optimized for printing.

Once the preview service development has been completed, the application needs to be packaged as a container. One significant dependency of the preview service is ImageMagick. ImageMagick is not a Java library but an independent command-line tool that needs to be installed separately on to the target system. Before containerization, it was essential to install the Java Runtime Environment (JRE), ImageMagick as well as the Preview Service itself on the target system. This installation was often a manual effort and prevented the system from being flexible and scalable. Containers have changed this significantly: Recall, Docker (2019) describes a container as a standard unit of software that packages up code and all its dependencies so that an application can

run quickly and reliably on a broad range of computing environments. In order to create a new version of such a standard unit of software – also known as Container Image – for the preview service, the file 'Dockerfile' in the project's root needs to be executed (see https://git.io/JeS48). Here, the execution generates a new container image containing the latest version of the preview service, the JRE as well as ImageMagick. When finished, the container image is being tagged with name and version (e. g. 'ricebean/preview-service:1.0') and finally is uploaded to 'Docker Hub' (see https://hub.docker.com/r/ricebean/preview-service). 'Docker Hub' are called repositories to archive and share container images with teammates, customers, or the public (Docker Inc., 2019 b). 'Docker Hub' is not the only repository provider, but probably the most common one.

Container Images stored in a repository can easily be deployed on a broad range of diverse environments. As prior mentioned, target environments include Amazon's Elastic Container Service for Kubernetes (Amazon EKS), Google's Kubernetes Engine, or Microsoft's Azure Kubernetes Service (AKS). In case just a simple container without a data storage needs to be deployed, fully managed (called 'serverless') container runtime environments are additionally provided. Cloud Run, for instance, is such a serverless container runtime environment provided by Google Cloud (2019 c). The author has also chosen this one in order to deploy the preview service for demonstration purposes (see https://preview-service.ricebean.net) It should be noted that the first call of the application may take some time as the Java application within the container has to be launched. Along public cloud providers, Container Images stored in a repository can even be deployed on the readers local desktop computer. A running Docker Engine is the only prerequisite needed (Docker Inc., 2019 c). When the engine is running, a container can simply be activated executing the 'docker run' command in the command-line, extended with optional parameters as well as the appropriates containers tag:

```
$ docker run -p 8080:8080 ricebean/preview-service:1.0
```

Code example 1 docker run command

The code sample above illustrates the command needed in order to run the preview service on the local environment. The first part of the command ('docker run') advises the Docker Engine to start a container. The second part ('-p 8080:8080') is a parameter that instructs the engine to forward the localhost's port 8080 to the container's port 8080. This parameter is required, as the container will be started as an autonomous machine on the user's local computer. The last part of the command ('ricebean/preview-service:1.0') is the globally unique tag of the preview service container image - version 1.0. It may be considered that, once a Container Image is available on a public repository, it can be deployed easily on a wide range of diverse environments.

As earlier mentioned, containers are designed to be immutable and stateless. Accordingly, all running instances of the same container image behaves almost identical – independent of the environment they are running in. This design criteria of containers are beneficial for testing, as these criteria ensure that the identical functionality and configurations can be deployed on both the test systems as well as the productive systems. In the context of the preview generation service, this means, each new version can be deeply tested before being deployed productively. One test scenario of the preview

service is, for instance, the pdf rendering test. This test consists of a set of pdf documents next to the expected preview results. When executing that test, the test sends all pdf documents to the test instance of the preview service, waits for the generated preview images, and compare them with the appropriate expected preview results. If there is no difference, the test succeeds, otherwise it fails. This test method protects the system from unwanted changes. In preview generation, slight changes from one version to the next seems not to be that crucial as it is just a preview image. However, the situation will change as soon as a service modifies the customer's artwork files, which is mission-critical. Using containers allow system operators to test adequately mission-critical services in a separate test environment as containers ensure an identical behavior independently of their environment.

Logs are an integral part of application management as they make applications replicable during runtime. They record all notable events that occurred in combination with metadata, such as the timestamp or the process id. In the following is a screenshot of the logs produced by the preview service form applications start through the generation of the first preview image. The following image depicts the consoles standard output (Figure 6).

When the preview service starts, first, the 'Spring' banner appears. Spring (2019) is the programming framework that has been used for the development of the preview service. Following the banner, the first log message appears. Log messages written by the preview service are following Spring's default log scheme: {TIMESTAMP} {SEVERITY} {METADATA} {MESSAGE}. The timestamp of the first message is '2019-12-22 10:00:38.544', the severity is 'INFO', and the actual log message is 'Starting application on (…)'. The timestamp reflects the date and time when the message has been generated. The severity defines whether this message is of informal ('INFO'), warning ('WARN') or critical ('ERROR') nature. The message itself is the human-readable description of the log event. The

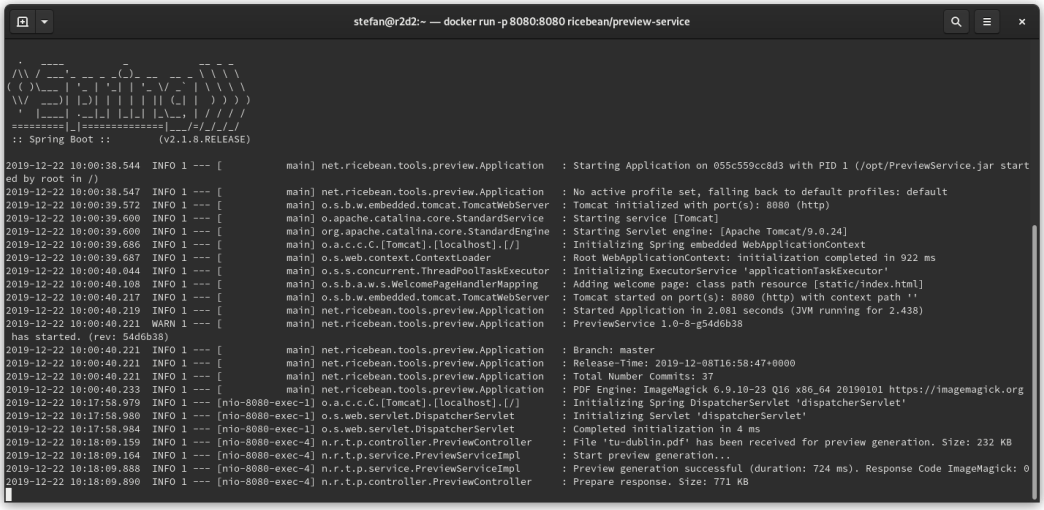

Figure 6: Logs Preview Service StdOut

log events depicted in the figure above reveals that the start of the preview service took from 10:00:38.547 through 10:00:40.233. During that time, multiple log messages containing the application's status and meta-information has been produced. The log events produced by the generation of the preview image of the file 'tu-dublin.pdf' can be seen up from 10:17:58.221. Each running container instance produces such kind of log messages. Keeping the overview of all messages is important, as it is an indicator of the health of the entire system. However, the more containers are involved in the system, the more challenging the task. In such a situation, a central log management and analysis tools become unavoidable. The tool selected for the preview service deployed at the Google Cloud (see https://preview-service.ricebean.net) is Stackdriver Logging (Google Cloud, 2019 d) as this is the pre-configured one in the Google Cloud.

The containerization security aspects addressed prior in this document pertain to both the container runtime engine and the container image. As the preview service is running in Google Cloud's serverless environment, primarily security issues regarding the container image have been considered. One important criterion in that field is the minimalization of the container's size in order to decrease the containers attack surface. When creating a new container image, first, a Base Image needs to be selected, in which the custom application is going to be installed. Base Images are container images containing an operating system layer, as well as some pre-installed software, and therefore have a significant impact on the size of the final container. The preview service image is built upon the 'openjdk:8-jre-slim' Base Image (https://git.io/JeS48 - line 44). This image provides a Linux kernel and comes with pre-installed essential Debian Linux tools as well as a Java 8 Runtime Environment (JRE). Another method to minimize container sizes are multi-stage builds (Docker Inc., 2019 d). Multi-stage builds are a cascade of temporary containers during image generation in order to keep compilation tools out of the final image. The compilation of the application's source code usually requires additional compilation tools. Java, for instance, requires the Java Development Kit (JDK). By the use of multi-stage builds, temporary containers are cascaded in order to generate the final application, which finally is being copied into the final image (https://git.io/JeS48 - line 21). This method allows developers to keep the compilation tools out of the final image – which again minimizes the container size. Minimal container sizes are not only beneficial for security reasons, but also storage and orchestration.

Conclusion

Exposing prepress cloud services as containers is the preferred way to go. The concept of containerization allows developers to provide services as standard units of software that can be easily deployed and managed on many target cloud environments (public, intern, local). The standard units contain not only the self-made application, but also all external dependencies. The Preview Generation Cloud Service, for instance, requires an ImageMagick (I.M.) pre-installed on the Operating System, as this command-line application is doing the actual preview generation. Providing the preview cloud service using the conventional way, someone would have to install ImageMagick on the host system. Further, it shall be ensured that the same version of I.M. is used as previously was used during testing. Also, it shall be ensured that the I.M. version keeps

the same and is not being updated accidentally, e.g., by automatic system updates. When having mission-critical service, the conventional way requires a high management effort in order to get equal service stability and quality as containerization provides out of the box. Containers allow to put all dependencies in a defined version as well as the application itself into one standard unit of software. So, it can be ensured that an application is always running in a well-defined environment. Further, having a standard unit containing all dependencies makes services easily interchangeable between cloud systems.

Containers are flexible regarding (third-party) applications, libraries, and programming languages used inside. Recall, containers are a kind of virtualization of the Operating Systems (O.S.). So, the only prerequisite of libraries and applications installed in a container is that they would technically run on the host O.S. This means, when the host O.S. is a Linux amd64, all software packaged inside the container must be runnable on such as system architecture. The preview cloud service, for instance, has been implemented in Java – but only because the author is best experienced in that programming language. The service could also have been implemented in PHP, Python, Node.js, C++, C#, or any other one. This kind of flexibility works, because the programming language's runtime environment can be installed in the container in the same way as ImageMagick has been installed. As the preview service is a Java application, a Java Runtime Environment (JRE) is required to be beside in the container. This flexibility of containers does no longer require that an entire production system must be built from one single technology only. Each development team can decide for themself, which (third-party) applications, libraries, and programming language they want to use. The technology stack used may differ from the team's experience and even from the service's business case. The concept of individual and isolated services, along with the high flexibility, is how containers promote the microservice architectures. Microservices are the way to enable printing houses to build flexible, adaptable, and sustainable software systems - as previously outlined in this document.

The concept of containers to encapsulate an individual service application, including all depending third-party applications and libraries in a standardized unit, makes the technology beneficial for mission-critical tasks. Besides, the fact that containers are immutable and stateless even reinforce the beneficial effect. Both ensure that a container image of the same version behaves equally – independently of the cloud environment. Note, the worst-case scenario in a printing house is NOT that a service breaks, and the entire system is interrupted from operation for a while. The worst-case scenario will be if a service is manipulating customer artworks in a way that only the customers recognize. In such a scenario, a printing company produces actively waste – maybe even for multiple days. Whereas overcapacities can compensate for the prior scenario, the costs of the latter scenario could be immense, as damaged goods are being produced and delivered. Containers help to minimize this risk, as they allow a sustainable testability. The architectural concept of containers guarantees that a container behaves identically in a test environment as later, when running in production. This capability enables companies to fundamentally test each version of mission-critical service and ensure that precisely the tested functionality is being deployed to production. Containerization is a convenient and sustainable method for mission-critical

services.

The fact containers can be started and stopped quickly and, at any time, improves the reliability of the overall system. This behaviour allows, for instance, to scale a system elastically: As soon as a performance peak is notified, immediately additional instances of the container can be started and vice versa. The additional container instances are not limited to be started in the same cloud environment. So, companies become capable of outsourcing their performance peaks to external computing resources. Further, the capability to quickly start and stop container instances enable services to be updated with zero downtime. This functionality is typically part of the cloud orchestration and ensures that the new service version is started and up working before the prior one is going to be shut down. As a result, a service can be updated without downtime and interruption of consumers. In case a service update fails or causes an operational error, the update can easily be rolled back by re-deploying the formerly running version of the service. Designing services as containers will raise the reliability of the overall system as many operational exceptional cases can be addressed.

Interoperability Conformance Specifications (ICS) are a proper way to specify Application Programming Interfaces (API) in the printing industry. CIP4 typically defines the two roles which are involved in the communication (for instance, 'Worker' and 'Manager') and define in an accurate way, which role has to read or write which information. Defining roles raises the precision of interface specifications resulting in better interface implementations. However, ICS Documents are typically published as PDF Documents. Meaning, they are primarily designed for human readability. CIP4 has not changed this method since publishing the first document. However, in order to further improve the quality of API implementation, it could be helpful to have an API specification which is both human and machine-readable. Such a specification would tightly couple the specification with an implementation, as tools for validation and source code generation can be used. There are some API specification standards out, such as OpenAPI (SmartBear Software, 2019), which allow developers to combine the human- as well as the machine-readable description of an API in one document. However, this kind of specification typically has a strict structure, which reduces the flexibility in writing. Maybe the CIP4 Organization can consider this as an idea of how to improve their ICS Documents.
Another question coming up during evaluation pertains to XJDF / XJMF and would need further conceptional consideration by the CIP4 Organization. In order to submit a job to the preview generation service, a ZIP Archive, containing an XJMF Message 'SubmitQueueEntry', which references an XJDF Document, which further references the Artwork PDF has to be generated. This ZIP Archive has to be sent to the preview service's static URL endpoint. The same situation also pertains to the XJMF Message 'ReturnQueueEntry', which initiates the response of the service, containing the generated preview image. The question is why this indirection is required to have an XJMF Message, which is referencing the actual XJDF Document. Of course, the XJMF brings the XJDF in a context such as 'submit', 'return', 'resubmit', 'cancel' as well as provide further information such as priority. However, this method can be simplified and streamlined by using Representational State Transfer (REST) (Fielding, 2000) and Hypertext Transfer Protocol (HTTP) (Fiedling & Reschke, 2014). HTTP Methods and

encoding information in the URL can be used to replace XJMF for any job-related communication. As a consequence, XJDF Documents can be sent directly to a service's endpoint.

Acknowledgements

At this point, I would like to extend my warmest thanks to Dr. Kevin Byrne and Dr. Martin Delp, who have provided strong support and kept me on track towards publishing this first paper. Both Kevin and Martin provided tremendous help and support in my personal development, enabling me to elevate my skills and thought processes to a level expected in academia. I would also like to wholeheartedly thank my wife, Anna, who kept the world around me up and running while I sat in front of my computer conducting research and writing this paper. Without her at my side, such a project would never have been possible. Another prominent supporter of this project has been my friend Windsor Tanner, who assisted me with a substantial amount of proofreading. Last but not least, I would like to thank Dr. Martin Habekost and the two anonymous reviewers who helped to get this paper published.

References:

1. Armbrust, Fox, Griffith, Joseph, Kath, Konwinski, . . . Zaharia. (2010, April). A view of cloud computing. Communications of the ACM, 53(4).
2. Balalaie, Heydarnoori, & Jamshidi. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In I. C. Society, IEEE Software (pp. 42-52).
3. Bernstein. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. In IEEE Cloud Computing, vol. 1, no. 3 (pp. 81-84).
4. Bryman, & Bell. (2015). Business Research Methods. Oxford: Oxford University Press.
5. CEWE. (2018). CEWE Geschäftsbericht 2018.
6. Chelladhurai, J., Chelliah, P. R., & Kumar, S. A. (2016). Securing Docker Containers from Denial of Service. In IEEE International Conference on Services Computing (pp. 856-859). San Francisco, CA, USA: IEEE Computer Society.
7. Chifor, A. (2017, May 30). Container Orchestration with Kubernetes: An Overview. Retrieved Decembre 29, 2019, from https://medium.com/onfido-tech/container-orchestration-with-kubernetes-an-overview-da1d-39ff2f91edium.com/onfido-tech/container-orchestration-with-kubernetes-an-overview-da1d39ff2f91
8. Cimpress. (2019 a). OUR STRATEGIC CAPABILITIES. Retrieved from https://cimpress.com/our-strategic-capabilities/
9. Cimpress. (2019 b). Cimpress 2018 Anunual Report. Cimpress.
10. CIP4. (2015, December 2). MIS to Conventional Printing ICS. Retrieved from https://confluence.cip4.org/display/PUB/MIS+to+Conventional+Printing+ICS
11. CIP4. (2018). XJDF Specification 2.0 Final. Zurich: CIP4.
12. CIP4. (2019 a). CIP4 Organization. Retrieved September 15, 2019
13. CIP4. (2019 b). ICS Documents. Retrieved December 3, 2019, from https://confluence.cip4.org/display/PUB/ICS+Documents
14. CIP4. (2019 c). Spec Incubator. Retrieved December 08, 2019, from https://confluence.cip4.org/display/TSC/Spec+Incubator
15. Dikaiakos, M., Pallis, G., Katsaros, D., Mehra, P., & Vakali, A. (2009). Cloud Computing - Distributed Internet Computing for IT and Scientific Research. IEEE Internet Computing, 10-13.
16. Docker. (2019, June 10). What is a Container? Retrieved from https://www.docker.com/resources/what-container
17. Docker Inc. (2019 a, June 24). Best practices for writing

Dockerfiles. Retrieved from https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
18. Docker Inc. (2019 b). Repositories. Retrieved December 08, 2019, from https://docs.docker.com/docker-hub/repos/
19. Docker Inc. (2019 c). About Docker Engine - Community. Retrieved December 09, 2019, from https://docs.docker.com/install/
20. Docker Inc. (2019 d). Use multi-stage builds. Retrieved December 22, 2019, from http://docs.docker.com/develop/develop-images/multistage-build/
21. Fiedling, R., & Reschke, J. (2014, June). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Retrieved from https://tools.ietf.org/html/rfc7231
22. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Irvine: UNIVERSITY OF CALIFORNIA.
23. Fischer, T. (2019, June 04). Flyeralarm: Start-up Spirit – Interview with company founder, Thorsten Fischer. (B. Zipper, Interviewer) Retrieved from https://www.beyond-print.net/flyeralarm-start-up-spirit-interview-with-company-founder-thorsten-fischer/
24. Fisher, C. (2010). Researching and Writing a Dissertation. Harlow: Pearson Education Limited.
25. flyeralarm. (2017, March 13). Hochleistungszuwachs im Maschinenpark. Retrieved from https://media-channel.flyeralarm.com/hochleistungszuwachs-im-maschinenpark/
26. flyeralarm. (2019). About us. Retrieved August 10, 2019, from https://www.flyeralarm.com/uk/content/index/open/id/1499/about-us.html
27. Fogliatto, F. S., Silveira, G. J., & Borenstein, D. (2012). The mass customization decade: An updated review of the literature. In Elsevier, Int. J. Production Economics (pp. 14-25).
28. Foulds, I. (2018). Learn Azure in a Month of Lunches. Shelter Island: Manning Publications Co.
29. Gibson, J., Eveleigh, D., Rondeau, R., & Tan, Q. (2012). Benefits and Challenges of Three Cloud Computing Service Models . In Fourth International Conference on Computational Aspects of Social Networks (pp. 198-205). IEEE.
30. GitHub. (2019). Mastering Markdown. Retrieved December 08, 2019, from https://guides.github.com/features/mastering-markdown/
31. Google Cloud. (2019 a, June 17). Best practices for building containers. Retrieved from https://cloud.google.com/solutions/best-practices-for-building-containers
32. Google Cloud. (2019 b, June 17). Best Practices for Operating Containers. Retrieved from https://cloud.google.com/solutions/best-practices-for-operating-containers
33. Google Cloud. (2019 c). Cloud Run. Retrieved December 09, 2019, from https://cloud.google.com/run/
34. Google Cloud. (2019 d). Stackdriver Logging. Retrieved December 22, 2019, from https://cloud.google.com/logging
35. Google Trends. (2019, June 12). Google Trends. Retrieved from https://trends.google.com/trends/
36. Hromis, A. (2019). Docker Reference Architecture: Docker Logging Design and Best Practices. Retrieved June 30, 2019, from https://success.docker.com/article/logging-best-practices
37. ICC. (2019). Interoperability Conformance Specifications. Retrieved September 15, 2019, from http://www.color.org/iccmax/ics.xalter
38. ImageMagick Studio LLC. (2019). ImageMagick. Retrieved December 08, 2019, from https://imagemagick.org/
39. Islam, A., Irfan, M., Mohiuddin, K., & Al-Kabashi, H. (2013). Cloud: The Global Transformation. International Conference on Cloud & Ubiquitous Computing & Emerging Technologies (pp. 58-62). IEEE.
40. Keane, R., Robertson, E., & Coursol, S. (2003). United States Patent No. US 6,650,433 B1.
41. Keung, J., & Kwok, F. (2012). Cloud Deployment Model Selection Assessment for SMEs: Renting or Buying a Cloud. In 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing (pp. 21-28). IEEE Computer Society.
42. Kim, J. (2017, November 22). The API Manifesto Success Story. Retrieved from https://www.profocustechnology.com/enterprise/api-manifesto-success-story/
43. König, A. (2013). Mass Customization of Print Products. Berlin, Germany: Beuth University of Applied Sciences.
44. Lenarduzzi, V., & Taibi, D. (2016). MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product. In 2. 4. (SEAA), Davide (pp. 112-119). Limassol, Cyprus: IEEE.
45. Lin, J. (2018, August 1). API-first software development for modern organizations. Medium - Better Practices. Retrieved from https://medium.com/better-practices/api-first-software-development-for-modern-organizations-fdbfba9a66d3
46. Maurya, A. (2017, June 12). What is a Minimum Viable Product (MVP). Retrieved December 23, 2019, from https://blog.leanstack.com/minimum-viable-product-mvp-7e280b0b9418
47. Meissner, S. (2019, April 3). XJDF - Standardisierter Datenaustausch ZWISCHEN Unternehmen. Retrieved from https://www.slideshare.net/stefanmeissner/xjdf-datenaustausch-zwischen-unternehmen
48. Mell, & Grance. (2011). The NIST Definition of Cloud.

Gaithersburg: National Institute of Standards and Technology.
49. Onlineprinters. (2019, May 15). 15 Jahre Onlineprinters: Firmengründer Walter Meyer im Jubiläumsgespräch. Retrieved August 10, 2019, from https://www.diedruckerei.de/magazin/15-jahre-onlineprinters-firmengruender-walter-meyer/
50. Oracle. (2019). Java Powers Our Digital World. Retrieved December 08, 2019, from https://go.java
51. Orenstein, D. (2000, January 10). Application Programming Interface. COMPUTERWORLD, p. 66.
52. Panzar, J. C., & Willig, R. D. (1981). Economies of Scope. In A. E. Association, Ninety-Third Annual Meeting of the American Economic Associatio.
53. Pine, B. J. (1993). Mass Customization. Boston, Massachusetts: Harvard Business School Press.
54. Reddy, M. (2011). API design for C++. Elsevier Inc.
55. Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. New York: Crown Business.
56. Saxoprint. (2019). Zahlen & Fakten. Retrieved August 10, 2019, from https://www.saxoprint.de/ueberuns/unternehmen/fakten
57. Silberston, A. (1972). Economies of Scale in Theory and Practice. In W. o. Society, The Economic Journal, Vol. 82, No. 325, Special Issue: In Honour of E.A.G. Robinson (pp. 369-391).
58. Singh, V., & Peddoju, S. K. (2017). Container-based Microservice Architecture for Cloud Applications. In IEEE, International Conference on Computing, Communication and Automation (ICCCA2017) (pp. 847-852). IEEE.
59. Singleton, A. (2016). The Economics of Microservices. In I. C. Society, IEEE Cloud Computing (pp. 16-20). IEEE.
60. SmartBear Software. (2019). API Development for Everyone. Retrieved January 07, 2020, from https://swagger.io
61. Spring. (2019). Spring Boot. Retrieved December 22, 2019, from https://spring.io/projects/spring-boot
62. Statista. (2019, September 7). Umsatz der flyeralarm GmbH in den Jahren 2006 bis 2017 (in Millionen Euro). Retrieved from https://de.statista.com/statistik/daten/studie/388615/umfrage/umsatz-von-flyeralarm/
63. Sun, L. (2015, November 4). Container Orchestration = Harmony for Born in the Cloud Applications. Retrieved December 29, 2019, from https://www.ibm.com/blogs/cloud-archive/2015/11/container-orchestration-harmony-for-born-in-the-cloud-applications/
64. Sysdig. (2018, May 29). Retrieved June 12, 2019, from 2018 Docker usage report.: https://sysdig.com/blog/2018-docker-usage-report/
65. Trieloff, L. (2017, June 2). Three Principles of API First Design. Medium - Adobe Tech Blog. Retrieved from https://medium.com/adobetech/three-principles-of-api-first-design-fa6666d9f694
66. Weinman, J. (2016). Migrating to or away from the Public Cloud. IEEE Cloud Computing, pp. 6-10.
67. Wiggins, A. (2017). The Twelve Factors. Retrieved June 24, 2019, from https://12factor.net/
68. WIRmachenDRUCK. (2019). WIRmachenDRUCK – druckreifer Erfolg. Retrieved August 10, 2019, from https://www.wir-machen-druck.de/unternehmen.html
69. Zalando. (2019 a). Zalando RESTful API and Event Scheme Guidelines. Retrieved September 14, 2019, from https://opensource.zalando.com/restful-api-guidelines
70. Zalando. (2019 b). Zalando's Engineering and Architecture Principles. Retrieved December 03, 2019, from https://github.com/zalando/engineering-principles

Stefan Meissner

Stefan Meissner, Dipl.-Ing. (FH)
PhD Candidate TU Dublin

Dublin, Ireland

d06114283@mytudublin.ie