Taming the Complexity of Elasticity, Scalability and Transferability in Cloud Computing

Cloud-Native Applications for SMEs

Peter-Christian Quint, Nane Kratzke
Lübeck University of Applied Sciences, Center of Excellence CoSA
Lübeck, Germany
email: {peter-christian.quint, nane.kratzke}@fh-luebeck.de

Abstract—**Cloud computing enables companies getting computational and storage resources on demand. Especially when using features like elasticity and scaling, cloud computing can be a very powerful technology to run, e.g., a web service without worries about failure by overload or about wasting money by paid use of unneeded resources. For using these features, developers can use or implement cloud-native applications (CNA), containerized software running on an elastic platform. Nevertheless, a CNA can be complex at planning, installation and configuration, maintenance and searching for failures. Small and medium-sized enterprises (SMEs) are mostly limited by their personnel and financial restrictions. So, using these offered services can facilitate a very fast realization of the software project. However, by using these (proprietary) services it is often difficult to migrate between cloud vendors. This paper introduces *C*^{4}*S*, an open source system for SMEs to deploy and operate their container application with features like elasticity, auto-scaling and load balancing. The system also supports transferability features for migrating containers between different Infrastructure as a Service (IaaS) platforms. Thus, *C*^{4}*S* is a solution for SMEs to use the benefits of cloud computing with IaaS migration features to reduce vendor lock-in.**

Keywords–Cloud-Native Application; Elastic Platform; Microservice; SME; Vendor Lock-In; Container Cluster; Cloud Computing; Elasticity; Scalability; Transferability

I. INTRODUCTION

Note: This paper presents our research prototype ***C*^{4}*S*** (Container Cluster in Cloud Computing System), which was previously presented in [1]. This extended paper includes further information about our target user group, CNAs as basic technology and more details about ***C*^{4}*S***.

Infrastructure as a service (IaaS) enables companies to get resources like computational power, storage and network connectivity on demand. IaaS can be obtained on public or private clouds. Public clouds are provided by third parties for general public use. Type representatives are Amazon's Elastic Compute Cloud (EC2) and Google Compute Engine (GCE). Private Clouds are intended for the exclusive use by a single organization [3]. They are mostly installed on the respective company's own infrastructure. OpenStack is a cloud platform for providing (not exclusively) private clouds. One big benefit using cloud computing is the elastic scaling. Elasticity means the possibility to match available resources with the current demands as closely as possible [4]. Scalability is the ability of the system to accommodate larger loads by adding resources or accommodate weakening loads by removing resources [4]. With auto-scaling, resources can be added automatically when they are needed and removed when they are not in use [5]. The resources are allocated on demand and the customer only has to pay for requested resources. The system described in this paper will support several, public and private, cloud environments. Features like elastic scaling and transferability will also be available. We define transferability as the possibility to migrate some or all containers between different cloud platforms. This is needed to avoid vendor lock-in by the cloud providers, which is a major obstacle for small and medium-sized enterprises (SMEs) in cloud computing [6]. Only a few research projects deal with the specific needs of SMEs in cloud computing [7].

In the last few years, container technologies like Docker became more and more common. Docker is an open source and lightweight virtualization solution to provide an application deployment without having the overhead of virtual machines [8]. With Docker, applications can be easily deployed on several machine types. This makes launching containers from the same application (image) possible, e.g., on a personal computer or a datacenter server.

Container clusters like Kubernetes (arose from Google Borg) [9] and Mesos [10] can deploy a huge number of containers on private and public clouds. A big benefit of cluster technologies is the horizontal scalability of the containers, the fast development and the contained software defined network, which is often necessary for distributed container applications. Container and container cluster software are mostly open source and free to use.

C*^{4}*S is designed to (automatically) deploy and operate container cluster applications without vendor lock-in. Moreover, the system will be able to monitor the cloud platform, the container cluster and the containers themselves. Beside bare reporting, the system will offer methods to keep the application running in most failure states. Altogether, the ***C*^{4}*S*** can make container cluster cloud computing technologies usable for SMEs without large and highly specialized IT departments. ***C*^{4}*S*** is especially designed as a cloud solution for SMEs.

This paper is **structured** as follows: Section II deals with the target group of the upcoming ***C*^{4}*S*** cloud management system. The technical background in form of cloud-native applications

TABLE I. SME definition of the European Union [2]

Enterprise category	Headcount: annual work unit	Annual turnover	or	Annual balance sheet total
Medium-sized	<250	≤ EUR 50 million		≤ EUR 43 million
Small	<50	≤ EUR 10 million		≤ EUR 10 million
Micro	<10	≤ EUR 2 million		≤ EUR 2 million

and elastic platforms is explained in Section III. The ***C*^{4}*S*** prototype is introduced in Section IV, followed by the major requirements of the system and the reasons for their selection in Section V. The architecture of the research prototype is presented in Section VI, the usage scenarios in Section VII and followed by a report about the current development status in Section VIII. The intended validation of concept is delineated in Section IX. Related work is described in Section X. Finally, the conclusion follows in Section XI.

II. CLOUD COMPUTING AND SMES

SMEs are mostly financially and personnel-wise restricted (see the European definition of SME [11]) and the management of container cluster applications with features like transferability and elasticity is complex. Thus, it can be hard to handle this complexity for a small (maybe only one person size) IT department. In the beginning, using services like IaaS might be very simple but the use of advanced cloud technologies like clusters, containers and cloud benefits like auto-scaling and load balancing can quickly grow into complex technical solutions. The cloud provider supplied services (e.g., auto-scaling) might pose another issue due to often having non-standardized service APIs. This is often resulting in inherent vendor lock-in [12]. However, there are products and services to manage multi-provider-clouds like the T-System Cloud Broker [13]. These management solutions also have disadvantages, i.e. they are mostly commercial, proprietary solutions and also often inherently designed for very big companies. These kind of cloud broker services move the dependencies from the cloud provider to the system/service provider like T-Systems. Amazon EC2 Container Service works only with Amazon EC2-instances, which means there is still a vendor lock-in. Both solutions just shift vendor lock-in to another company or another level of IaaS provisioning. Creating an open source system for easy deployment and managing of cloud applications in a container cluster would support SMEs using these technologies which reduces worries about vendor lock-in.

Our research is about avoiding vendor lock-in in cloud computing. Although our upcoming system is usable for all type of companies, we have placed the emphasis on a solution for SMEs because of their huge potential when using cloud technologies. The majority of companies located in Europe can also be defined as SME and we surmise that our solution for SMEs can rather easily adapt for the use in large companies than otherwise.

There are several definitions about SMEs. The Statistical Office of the European Union (Eurostat) defines SMEs by three thresholds criteria: Staff headcount and either annual turnover or annual balance sheet total (see Table I and [2], [14]). 99.76% of all companies in the information and communication sector (ICT) are SMEs (958,663 in total, data from 2013) [15]. Regardless of this, the turnover per employee is significantly depending on the size of the company (see Figure 1). We suspect that the economy of scale is partly a reason for this: Mass production can reduce the cost of a single part [16]. In the case of ICT SMEs, this might be reasoned by small IT-facilities and limitated hardware performance.

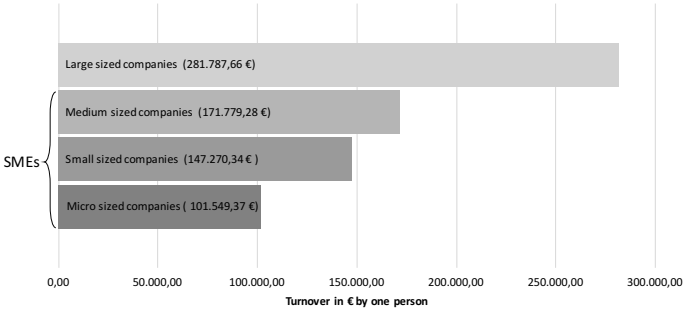

Figure 1. Average turnover by employee in ICT companies, separated by company type (data from 2013) [15]

However, cloud computing can help SMEs to bridge the disadvantages. A large scale of computational resources is not limited to large companies any more. So, instead of buying a server to calculate something in a week, a single person can also use cloud computing to rent more powerful virtual machines which can do this job in a day or even faster. Worries about investment risks can be avoided by using cloud computing. Resources can be requested on demand. This pay on demand model combined with auto-scaling functionality can prevent unnecessary costs [17] when, e.g., a new web service does not have the intended success. On the other hand, if the new service is widely used, there is no danger of service-crash because of inadequate resources. E.g., Animoto, a small video creation service company, scaled from 50 servers up to 3,500 in three days [18] as a result of a viral growth. For most companies, especially SMEs this is probably not possible in this manner by using own infrastructure. However, there is a lack of standardization in cloud computing. Different cloud providers offer different cloud services. Because of non-existing or supported standardized APIs, also nearly similar services provided by different vendors are often not swappable. Oftentimes, this makes a migration to another provider expensive and time-consuming [19] and is economically not use-

ful/possible, especially for SMEs with financial and personal limitations.

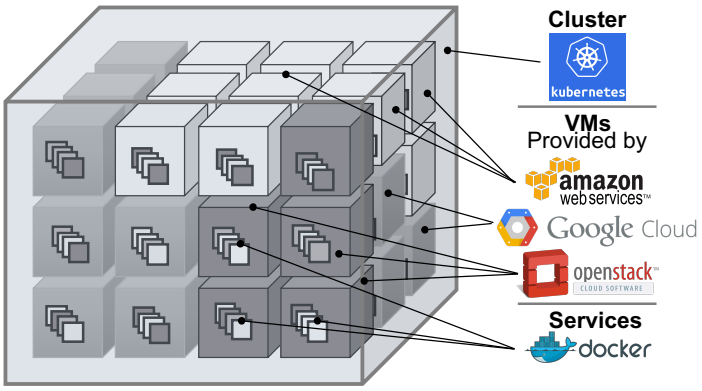

Figure 2. The elastic platform can be a cluster consisting of VMs. These VMs can be offered by different cloud providers.

III. CLOUD-NATIVE APPLICATIONS (CNA)

Migrating to the cloud also provides requirements for the application architecture. Although, it is possible to use a single virtual machine to run an, e.g., monolithic and not distributed application in a cloud environment, the most useful features of cloud computing, like elasticity and scaling, cannot be used or only with big restrictions. In order to apply these features, the application has to be designed to run on distributed systems. Our solution restricts the usable software architecture on CNAs only. This might be a hard limitation but enables running applications in an auto-scaling, fault-tolerant and highly available way with the possibility to migrate to any cloud infrastructure anytime.

A. Definition of Cloud-Native Application

As its name implies, cloud-native applications are designed for working in cloud computing environments. In research, the term ”cloud-native” was firstly used in 2012 by F. Leymann and his research group (the corresponding conference papers: [21], [22]). In general, there is no completely accepted definition of the term ”cloud-native”. This paper follows the understanding proposed by [20]:

A cloud-native application is a distributed, elastic and horizontally scalable system composed of (micro)services which isolates state in a minimum of stateful components. The application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a self-service elastic platform.

B. Components of a CNA

To create CNAs according to the proposed definition, the following technologies may be used:

Microservices. From legacy, applications have been often designed according to a monolithic software architecture. That means, a program is designed as a monolithic block. This block contains all or the most parts of the application. In contrast, in the microservice architectures, the application is spliced into (micro)services. Each service should be *small and focused on doing one thing well* [23]. This has a lot benefits in elastic cloud computing. The development and maintenance is simplified in a way that the services can be developed in small teams which operate independently and are specialized in a particular area. Each service can be updated without touching other application parts. Another big feature is the scalability of the single services. Instead of creating more and more instances of the whole monolith, only the required services have to be increased. Of course, microservices also bring new complexity to the application design. It is important to define what the individual service should contain. In practice, microservices are often not fully self-sufficient but expanding and updating a service should not require the need to adjust all other services.

The microservice architecture is used by well-known companies such as Netflix, Twitter, Dropbox and Amazon.

Elastic Platform. An elastic platform includes one or several container clusters and is used for the automatic deployment of the cloud-native application. The elastic platform uses resources in form of virtual machines. These resources are offered by private and/or public cloud provider.

However, these virtual machines used as nodes in the cluster must not necessarily be running on the same cloud infrastructure (see Figure 2). Popular open-source container clusters are Kubernetes, Apache Mesos and Docker Swarm.

Reference Model. The NIST-Definition of cloud-computing [3] defines three service models and four deployment models. These models are very common and useful for a lot of tasks. However, they are not fine-grained enough to describe the technology layers of a CNA suitable in an engineering point of view. For our research we use the cloud-native reference model (ClouNS) described in [20]. As shown in Figure 3, ClouNS consists of six layers covering four view points. The node centric view point covers the architectural layers 1-3. The layer 1 is the physical host. In cloud computing this part is mostly controlled by the IaaS provider and not visible for the user. The second and third layer corresponds to the IaaS model of the NIST definition. Usually, the IaaS provider offers a set of virtual machine types and operating system images to the user which can select a fitting configuration. However, the virtual machines are mostly connected to an internal network and useable over the internet. The Layer 3 also provides a standardized way of self-contained deployment units (e.g. the Docker engine). The cluster centric view point covers Layer 4. This describes the elastic platform and has often three sublayers. The cluster scheduler connects several (up to over one hundred thousand) nodes (Layer 3) to a logical unit. This enables a cluster of (Layer 3) nodes from different IaaS providers and makes an interoperable cloud and also the migration from one cloud platform to another possible. The use of a scheduler can avoid vendor lock-in. The overlay

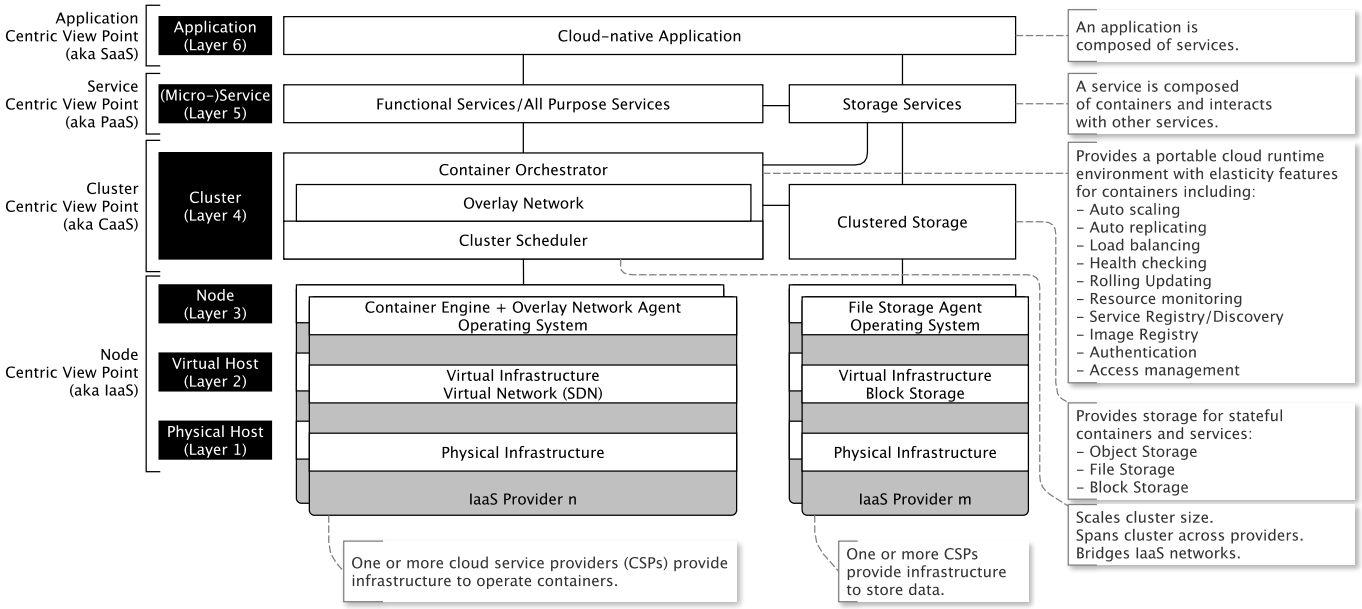

Figure 3. Cloud-native Application Reference Model (ClouNS), taken from [20]

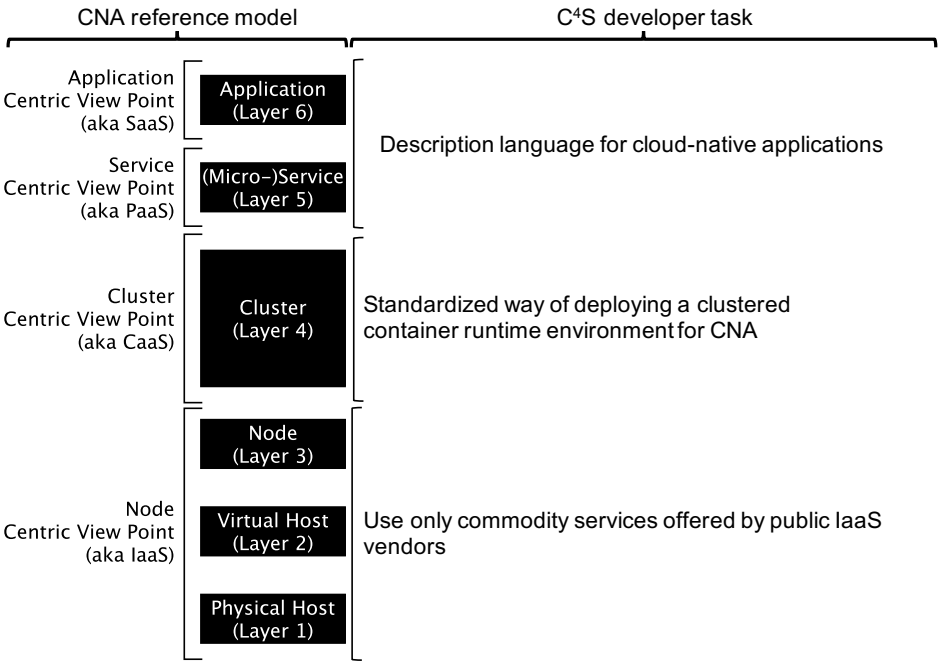

Figure 4. Developer task

network enables the communication between the containers in the cluster. Popular implementations are Flannel and Weave. Some overlay network solutions (like Weave) also enable an encrypted communication which facilitates interoperable working cluster with a high security. The orchestrator arranges the containers on different machines and enables (depending of the used cluster software) load balancing, container scaling and increases the fault tolerance by distributing container instances of the single images on several nodes. The service centric view point covers Layer 5. A (micro)service can also consist of a composition of containers. These containers are stored in the cluster (Level 4) and cannot be assigned to the specific (Layer 3 and under) machines. The application centric view point covers Layer 6, the application. If a Layer 5 service can be used over a human machine interface, it can also be a Layer 6 application. Usually, Layer 6 applications are a combination of Layer 5 services and often corresponding the SaaS model. This reference model is used to classify the single technologies of our cloud-native system and simplifies the description of our system architecture.

IV. RESEARCH PROTOTYPE C^4S

C^4S is our upcoming prototype to deploy and control CNAs.

The system is designed to handle the high complexity of a container cluster with benefits like elasticity, auto-scaling and transferability. Feature requirements and the technical specifications are explained below and illustrated in Table II. As shown in Figure 4, a solution to define secure, transferable and elastic services of typical complexity will be provided by designing and developing a generic cloud service description language. Thus, these services are deployable to any IaaS cloud infrastructure. This work promotes the implementation of easy-to-handle, elastic and transferable CNAs. The basic idea of our approach is to limit the use of cloud provided services to the basic - using computational and storage resources in the form of virtual machines. All other offered services increase the risk of vendor lock-in. Therefore, we use already available container und cluster technologies to provide features like auto-scaling, elasticity, load balancing and fault tolerance.

Next to avoiding proprietary services, we avoid vendor lock-in by the possibility of a fast, easy and always executable cloud platform migration. As illustrated in Figure 2, a cluster consists of several nodes. These nodes do not necessarily run on the same platform. This allows creating an interoperable cloud system. Of course, this technology has a high degree of complexity. Because of the limited (personal and financial) resources of most of the small companies, little (in extreme one man sized) IT departments must also be able to handle this complexity. Therefore, we focus on the usability of the interfaces and a high degree of automation. For a future validation of our approach we are developing the prototype system C^4S. We will show the feasibility of a system which avoids vendor lock-in, supports cloud features like scaling and load balancing and can hide the inherent complexity for the use of SMEs. We assume, that the requirements (described in the next Section V) of the prototype are also included in a future system designed for the productive use.

V. MAJOR REQUIREMENTS OF C^4S

C^4S is a system designed for an easy access to cloud computing benefits without having dependencies like IaaS vendor lock-in. To fulfill this goal the following technical and practical conditions are required.

A. Cloud-Native Applications

The basic feature of C^4S is to deploy a distributed cloud-native (container) application on cloud environments. Therefore, the user can easily configure the needed containers, the interfaces and the cloud environments. According to this configuration, the system can automatically deploy the application. The system will also monitor the application and inform the user if a failure is detected. According to the error, the system will try to fix it automatically to minimize the downtime. The user has an overview over the running application and can also get information about every single container. After the initial deployment, changes of the container composition are still possible. So, application parts (e.g., container types) can be replaced, e.g., to keep the application up to date.

B. Elastic Platform

The elastic platform (Layer 4 in the ClouNS model), technically a container cluster, is used for the automatic deployment of the cloud-native application. The elastic platform uses resources in form of virtual machines (Layer 2). These resources (Layer 2/3) can be obtained by private and/or public IaaS platforms. These virtual machines, used as nodes in the cluster, must not necessarily be running on the same cloud infrastructure. So, this architecture enables the use of a multiple-providers environment. Also, a provider migration is always possible. According to the configuration, the system can create virtual machines automatically and integrate them to the cluster. Another included benefit is the possibility of terminating and correspondingly down scaling of the elastic platform in the scope of functions. In the practical use, the elastic platform can be composed of open-source container cluster systems such as Kubernetes, Apache Mesos and Docker Swam.

C. Elasticity, Load Balancing and Auto-Scaling

As described in Section I, cloud computing enables features like elasticity, scalability and load balancing. C^4S enables the user to handle the inherent complexity of these features in an easy way. C^4S will have auto-scaling features. Thus, it is necessary to differentiate between scaling the services (Layer 5) and the cluster (Layer 4).

Service scaling: C^4S will be able to detect overload of the payload services. In this case, C^4S scales the services with deploying more container instances on the elastic platform.

Platform scaling: The system also detects if the cluster has only a few free resources left to orchestrate the container. In

TABLE II. Relation of the CNA reference model [20] and the functionality of C^4S.

(According to [20]) View Point Classification	Layer Classification		Features of C^4S
Application Centric View Point	Application	Layer 6	Description Language for defining Deploying and controlling Monitoring Scaling
Service Centric View Point	(Micro-) Service	Layer 5	
Cluster Centric View Point	Cluster	Layer 4	Deploying and controlling Load Balancing Monitoring Scaling
Node Centric View Point	Node	Layer 3	Creating and Terminating
	Virtual Node	Layer 2	Monitoring
	Physical Host	Layer 1	(Task of the provider only)

this case, the system automatically requests new virtual machines provided by the IaaS vendors. After the new resources are available they will be used to extend the cluster. Of course, downscaling the container application and the elastic platform is also supported. So, the cost for unneeded machines can be reduced. Downtime (or other system failures) because of overload can be prevented or shortened.

Load balancing is depending on the used cluster software. At first C^4S will support Kubernetes. This software is designed to orchestrate containers, so it provides load balancing functionality.

D. Prevent Dependencies

To avoid vendor lock-in by the cloud provider, the system can install a multi-cloud container cluster with transferability features. All or some containers can migrate from one cloud provider to another on demand. Accordingly, the user can select the using IaaS platform(s) and is able to change this choice anytime. To prevent dependencies by used software and services, the C^4S will be published under MIT license. It is recommended that all third-party parts like the cluster software are also open source. Thus, the consuming companies can adapt the source code to their special needs and are able to avoid dependencies to the C^4S themselves. The system has to be designed in a generic way for several IaaS platforms (see Figure 5, fourth block). Beside the cloud platforms, the users should not be limited by the choice of the container cluster. The modular architecture enables later extensions for missing cluster connectivity (see Figure 5, third block).

E. Optimization for SMEs

The inherent complexity of this architecture has to be useable by SMEs with only a few (in extreme one person size) IT-staff. To hide the complexity, the user should be able to set all necessary parameters easily. Therefore, especially the graphical user interface will be developed under consideration of usability aspects. To handle the complexity, C^4S needs a high degree of automation. This includes features like auto-scaling the services (Layer 5-6) and auto-scaling the elastic platform (Layer 4 and involved Layer 3). Scaling the services depends on the resource consumption of the individual container type. The system creates or terminates container instances in the cluster automatically. Auto-scaling of the cluster is done by creating or terminating virtual machines (Layer 2). So, C^4S requests and obtains new VMs and integrates them as nodes (Layer 3) automatically into the elastic platform (Layer 4). In the reverse direction, unused resources should be set free to save money. Therefore, the system can request a termination of machines automatically. Hence, they are not available as nodes for the elastic container platform any more.

VI. SYSTEM ARCHITECTURE OF C^4S

The architecture, as illustrated in Figure 5, is divided into four blocks. The core of C^4S are the monitoring, the deployment and the storage engine. The user can manage the deployment and get the monitoring events over two interfaces. The other two parts are the elastic platform and the IaaS environments.

A. Interfaces

For deploying, managing and controlling the cloud-native application and the elastic platform, the user can use two different types of interfaces:

The **Command Line Interface** (CLI) can be used to get monitoring information about the cloud-native (container) application, the elastic platform and the host-machines (VMs running on an IaaS platform). The user can manage actions like the initial start of the application and cluster deploying. Therefore, the configurations have to be set manually and the

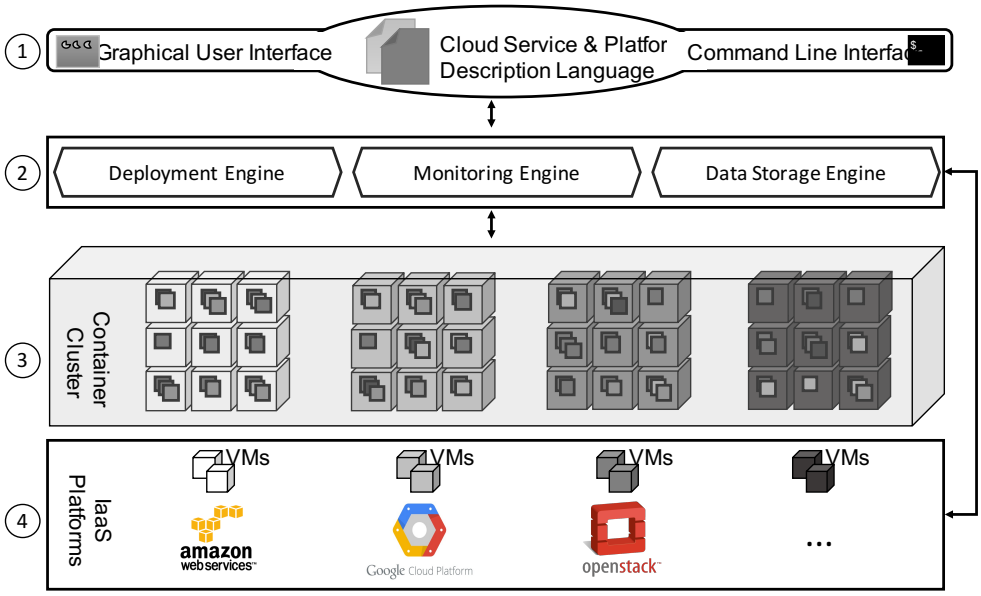

Figure 5. Architecture overview

Cloud Service Description Language has to be used for defining the cloud-application. Configuring the elastic platform has to be done with the use of the **Cloud Platform Description Language**.

The web-based **Graphical User Interface** (GUI) can be used for a more visual and simpler interaction with ***C*⁴*S***. The GUI provides graphical overviews for observing the used IaaS infrastructure, the cloud-native application and the elastic platform. Next to loading and storing the configuration (defined with the Cloud Platform Description Language and the Cloud Service Description Language), the user can use web forms to set the parameters. ***C*⁴*S*** automatically converts the set values into the required language format. In later developing processes several useful workflows will be integrated. So, this will help the user, e.g., to select the optimal host machine type for the individual purposes or processing a cloud provider migration.

After installing ***C*⁴*S***, the cloud-native application and the platform do only have to be configured over the interfaces. Creating and deleting virtual machines, installing the elastic platform and deploying the application on the platform will be automatically done by the system without manual interaction.

B. Engines

C*⁴*S contains several engines. As shown in Figure 6, these engines are interacting to realize features like auto-scaling the platform and the services.

The **Monitoring Engine** is an important part of the system. ***C*⁴*S*** will support several monitoring systems. So, the system has to monitor the **services** and also the **elastic platform** (clustered virtual machines provided by the IaaS platform). The system will proactively collect data and information from all services, the cluster software and the host systems of the platform. In case of the platform monitoring, the system analyzes, e.g., CPU performance, memory usage, free disk size and other performance data. This makes it possible to detect failure states, usage and load problems (like less free resources by high load of the cluster). The monitoring engine is also able to get statistics about the services. So, the health status of every container type can be analyzed. If the monitoring engine detects events such as under- or overload, the deployment engine will be notified.

The **Deployment Engine** is responsible for deploying the elastic platform and the services on it. Therefore, the engine can request the creation or termination of virtual machines running on a cloud platform. The new assigned VMs will be attached to the elastic platform to enlarge it. The engine can send a request to terminate a VM to the IaaS platform. The termination of VMs decreases the cluster. Thus, this can **reduce the cost** because only necessary machines are in use and the customer does not have to pay the IaaS provider for unused resources. In combination with the monitoring engine, the deployment engine is used for **platform auto-scaling** and makes the system more **fault tolerant**, too. So, if a few machines break down because of errors on the infrastructure of the cloud provider, the system automatically creates virtual machines on another provider to prevent or compensate downtime of the application. The workload of the containers can trigger to create or free more of them. The engine is also responsible for deploying the cloud-native application (in form of a service-composition) on the cluster. So, new services or service updates (switching the running service implementation) can be deployed with the deployment engine.

The **Data Storage Engine** is compatible with several block

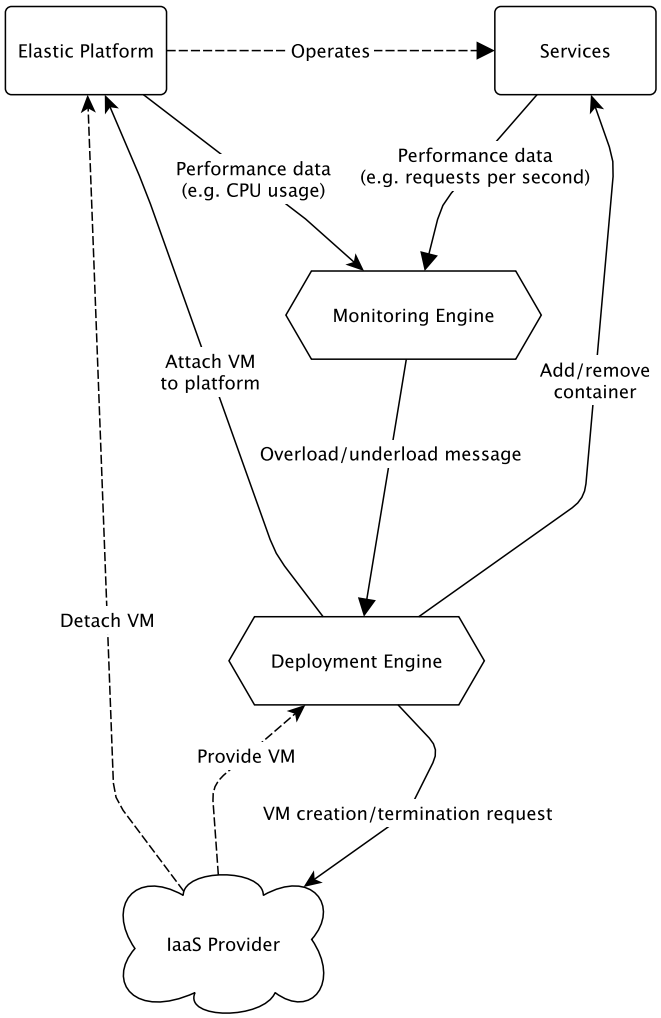

Figure 6. Data flow diagram of the deployment and the monitoring engine

and object storage systems to avoid vendor lock-in. The engine enables scalability and security features for data storage (not illustrated).

C. Elastic Platform

According to the configuration, the elastic platform will be installed automatically. The first versions of ***C*⁴*S*** will support Kubernetes. However, the system is designed generically to enable future extensions. So, several container cluster will be supported in later versions. The deployment engine can enlarge or decrease the cluster by creating or terminating cluster nodes. The deployment of the containers (cloud-native application) will also be controlled by the deployment engine. Next to scaling the container application and the cluster itself, other features like load balancing are included.

D. IaaS Platforms

According to the configuration, the required computing and storage resources (in form of virtual machines) will be requested automatically from the private and/or public IaaS platform(s). Therefore, ***C*⁴*S*** includes the "ecp-deployer", published at [24]. Currently, AWS and OpenStack Mitaka is directly supported. However, the deployer is designed to load plugins for supporting new platforms or new versions of them in a simple manner. This is also well documented to enable an uncomplicated extension-development.

VII. WORKFLOWS

C*⁴*S is designed for **deploying and controlling cloud-native applications** and the **elastic platform**. Like shown in Figure 7, this can be done in the following logical steps:

1. The user can use the command line interface and also the graphical interface to deploy and control the application. First of all, the elastic platform and the application have to be described.

1a. For defining the container-composition (image type, hierarchy, connection, etc.), ***C*⁴*S*** uses a cloud service description language. Using the CLI, this can be done manually. Alternative, if using the GUI, the set parameters will be parsed automatically into the correct format.

1b. Next to the parameters of the IaaS providers, the machine types, the credentials, the platform type (e.g. Kubernetes or Apache Mesos) also scaling limitations, load balancing setting and so on, have to be configured. Therefore, the cloud platform description language has to be used. Similarly to step 1a., this can be done in a comfortable way using the GUI or using the CLI directly.

2a. According to the settings, the system gets resources by the IaaS provider/cloud-platform in form of virtual machines.

2b. The created virtual machines will be automatically included as nodes to the elastic platform.

3. Now, ***C*⁴*S*** starts deploying the applications (e.g. Docker container-composition) on the cluster. After that, the initial deploying of the cloud-native application is finished.

n. (Without illustration:) According to the description, ***C*⁴*S*** is able to create/terminate virtual machines to scale the cluster itself.

To **avoid vendor lock-in**, it is important for ***C*⁴*S*** to support several elastic platforms (e.g. Kubernets or Apache Mesos) and IaaS provider. Nevertheless, ***C*⁴*S*** is designed to **migrate between different cloud provider** in an uninterruptible process. The user only has to set the configuration of the target provider (named "Provider B" in Figure 8) and command the system to start the migration. According to Figure 8, ***C*⁴*S*** will automatically process the following steps:

1. In general, the container type should be deployed on several different nodes for failure protection and for an uninterruptible migration.

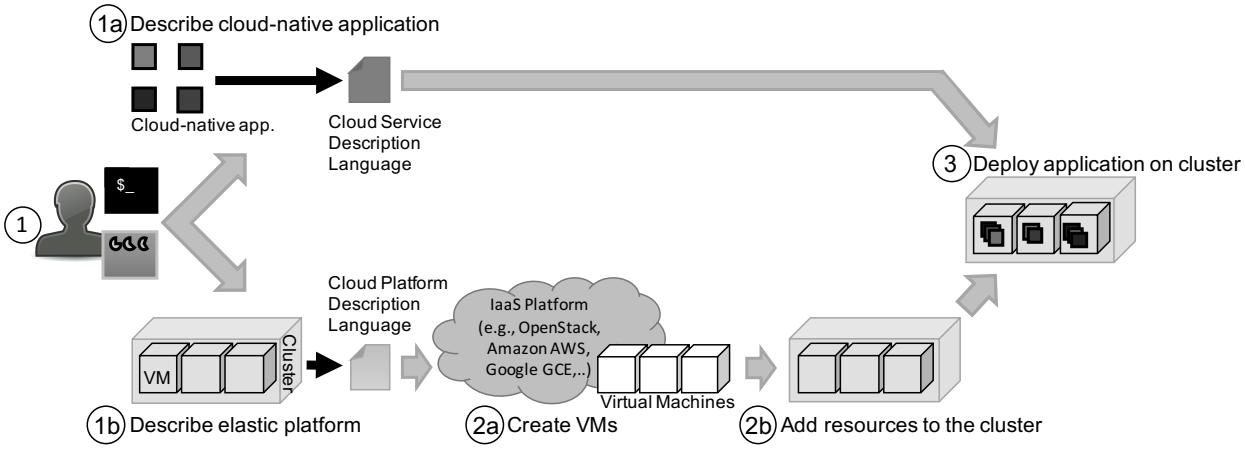

Figure 7. Deploying workflow

2. At first, C^4S allocates virtual machines provided by the target vendor to enlarge the elastic platform.
3. After the system terminates a "Provider A" virtual machine, the cluster decreases and containers are missing. The elastic platform detects that containers are missing and starts deploying them anew. In this way, the system terminates all machines from the original provider successively.
4. In the end, no virtual machine of the initial used provider is in use any more. The migration is completed and in accordance with the normal activities of the elastic platform, load balancing and scaling processes will be done on demand.

VIII. COMPLETED RESEARCH AND CURRENT STATUS

Our system is designed for CNAs. This includes the use of container virtualization (e.g., Docker, Rkt). Container based operating system virtualization can be used for scalable and high-performance tasks [25]. For using more than one container, container cluster can be used to operate in in a highly available and scalable way. The cluster itself also contains a software-designed network (SDN). C^4S is based on this technology. As internal part of C^4S, it is important to know if the network performance impact using these technologies is acceptable or not. Therefore, we developed ppbench [26], a tool for benchmarking network performance according to the use of container technology, the use of a cluster/SDN, the message size, the machine type and the programming language. It could be determined that the performance impact depends on all factors. Therefore, it is not possible to make basic disclosures about what is generally the best programming language or the ideal message size or if the use of a SDN has an acceptable impact on the network performance in general. However, with ppbench, reasonable combinations of these parameters can be determined in the context of a CNA design. The biggest insights from the benchmark tests are: the selection of the programming language has a big effect on the performance and the impact of a SDN depends on the machine type - using high-core machines can make the impact of the SDN negligible. All our findings about performance impacts using container and container clusters are published in [26], [27], [28], [29].

A Container Cluster should run on homogeneous machine types to provide fine-grained resource allocation capabilities [10]. Our solution for a migratable and/or interoperable cloud system requires an elastic platform consisting of nodes (virtual machines) provided by multiple providers. Accordingly, the choice of machine types of different providers is not trivial. We developed EasyCompare, an automatic benchmark suite for selecting most similar virtual machines provided by different cloud service providers. We used our tool to compare VMs provided by the two public cloud service providers Amazon Web Services and Google Compute Engine. In our benchmark experiments we compared 195 machine pairs and could only identify three machines pairs with a high similarity. All other machine pairs (over 98.5% of all possible combinations) are not or only useable with restrictions as cluster nodes. Although the selection of suitable machines is limited, the recommended pairs consist of "n1-standard" (GCE) and "m3" (AWS) machines. These machine types are universally useable and not limited to special tasks. They are also cost effective and not the most expansive machine types. Especially for SME, the limited selection options can have the advantage of a fast and simple decision. Particularly with regard to the performance, we generally recommend the machine-pairs with the highest amount of cores. We have published more details about EasyCompare and the benchmark results in [30], [26].

This first release of the **elastic container platform deployer** is accessible under [24]. The tool enables creating, deleting and changing an elastic platform consisting of many nodes which can be provided by different IaaS platforms/vendors in an easy manner. We have chosen Kubernetes as cluster software for the first version. Nevertheless, the deployer still supports OpenStack, Amazon AWS and Google GCE.

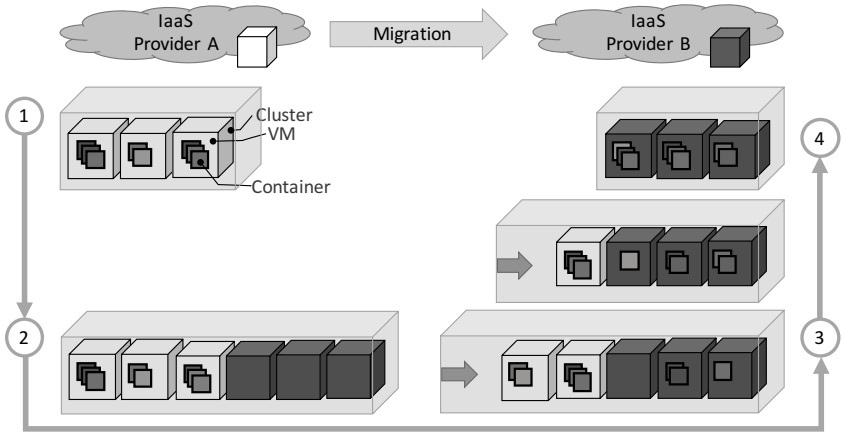

Figure 8. Migration workflow

TABLE III. Functional range of the planned releases

Version	Features
0.1	Kubernetes on OpenStack, Amazon AWS and Google GCE
0.2	Storage extension
0.3	Auto-scaling features
0.4	Deployment language
1.0	Stable version

We will continuously release new versions of C^4S after finishing significant parts. The rough planning, shown in Table III, did not include smaller releases which contain bugfixes, the GUI and enhancements to other cloud platforms or cluster software. We plan to release a stable version of C^4S by the end of 2017.

IX. INTENDED VALIDATION OF CONCEPT

It will be shown that SMEs can manage a container cluster over (multi) cloud platforms. At first it will be demonstrated that building a system which fits all the required features is possible. Therefore, a working, open source C^4S prototype which conforms the requirements set in Section V will be developed. The system has to be implemented in a modular and extendable way. As a cluster platform, C^4S will support Kubernetes first, other cluster environments will follow. Presenting interchangeability and the open source type of C^4S will show that dependencies by the used software can be prevented. To avoid vendor lock-in by the cloud provider, the prototype must be able to install a multi-cloud container cluster. First, the system will be compatible with the IaaS cloud platform type representatives Amazon EC2, Google GCE and OpenStack. To support other platforms, appropriate drivers can be implemented. Transferability features like moving all containers from one cloud platform to another will be implemented. Terminating all containers and virtual machines on one provider and creating them on another at the same time, without changes in features like elasticity and auto-scaling, will proof that C^4S is preventing vendor lock-in. The software will also manage container application deployment. It will deploy a container cluster, create and terminate containers and is usable for deploying applications. Also, workloads will be created to test the auto-scaling features. With enforced failure states, the robustness of the system will be demonstrated. It will be shown that the system is able to keep the applications running even when containers and virtual machines get disconnected. In the second part of the proof of concept, a company will employ the software. Thus, the expense for a small business using the container cluster manager will be evaluated. Finally, a proof of concept will be realized by several business companies. These companies will use the C^4S system on their own for testing a productive application deployment with real workload. Load balancing, elasticity, auto-scaling and transferability features will be applied in production. This way it will be shown that SMEs can handle the complexity of a container cluster application running on multiple cloud platforms without vendor lock-in or dispensing with features like auto-scaling.

X. RELATED WORK

Currently, technologies related to our workspace like container, cluster and cloud computing are still under hyped development. A lot of these technologies are still in productive use, also as a technical baseof very successful IT companies like Netflix, Amazon and others. Because of the hype and also the productive use by big companies, technologies, e.g., Docker are rising their opportunities and scope very fast. However, till now we couldn't find a solution which fits all requirements and features set in Section V, but there are several solutions with overlapping features and/or usage scenarios available.

A. IaaS Management and Transferability

Container migration from one cloud provider to another is an important feature of C^4S.

Vendor lock-in is caused, i.e., by a lack of standards [19]. Currently the proprietary EC2 is the de facto standard specification for managing cloud infrastructure. However, open standards like OCCI and CIMI are important to reduce vendor lock-in situations [31]. C^4S includes a special IaaS driver for each supported cloud provider. Other research approaches in cloud migration can be reviewed under [32]. There are several solutions like Apache Libcloud, KOALA [33], Scalr, Apache jclouds and deltacloud and T-Systems Cloud Broker for managing and deploying virtual machines on IaaS platforms. Except for the T-Systems Cloud Broker, the solutions are open source but have mostly payable services, reduced functionality or limited virtual machine quantities. These systems support features like creating, stopping and scaling virtual machines on IaaS cloud platforms. Some of them like the T-Systems Cloud Broker, Scalr and Apache jclouds are designed for cross-platform IaaS deployment. In contrast to the C^4S requirements, the presented cloud managers are limited to IaaS managing and do not offer container deploying services. Some of them do not prevent vendor lock-in by cloud providers or create new dependencies by itself (e.g., T-System Cloud Broker, KOALA are limited to Amazon AWS API compatible services).

B. Application Deployment

Peinl et al. [34] have defined requirements for a container application deployment system. These strongly coincide with the requirements for the C^4S system, which have been discussed in Section V. The research group also gives an overview about container cluster managing. For easy deploying a container application with monitoring, scaling and controlling benefits, there exist several commercial solutions like the Amazon EC2 Container Service, Microsoft Azure Container Service and Giant Swarm. Limited to the providers own IaaS infrastructure, these solutions are not designed for multi-cloud platform usages, especially between public clouds (a requirement of C^4S). Open source cluster managers like Apache Mesos and Kubernetes are designed to run workloads across tens of thousands of machines. The benefits and issues using cluster technologies are a very high reliability, availability and scalability [10] [9]. However, they are not designed to create and terminate virtual machines (like AWS instances), but to deploy applications on given resources. So, they cannot prevent cloud provider dependencies on their own, but provide essential ingredients to do so. Another cluster management tool for increasing the efficiency of datacenter servers is called Quasar which was developed by the Stanford University and designed for maximizing resource utilization. The system performs coordinated resource allocation. Several techniques analyze performance interferences, scaling (up and out) and resource heterogeneity [35].

XI. CONCLUSION AND OUTLOOK

In Europe, 98% of all companies are small and medium-sized. With cloud computing, SME can have access to computational resources which were limited to large companies in the past. Cloud features like elasticity and scaling can make the use of cloud resources more economical and effective. However, nowadays these features are only offered as non-standardized services by IaaS vendors. Using these services leads to a dependence on the cloud provider. Our approach is to use the base computational and storage resources only. All other offered services increase the risk of vendor lock-in. Therefore, we use already available container und cluster technologies to provide features like auto-scaling, elasticity, load balancing and fault tolerance. This is done by developing a prototype system called C^4S. The system is designed to operate CNAs with the above named features in a vendor lock-in free manner. It supports multi-IaaS-provider environments and is designed to realize cloud-provider migrations. The system is also implemented in a modular and generic way to allow an easy adaptation to different cloud platforms and container cluster software. The used technologies like the container and the storage cluster have an inherent complexity. Especially for SMEs with a small (in extreme one person sized) IT department it is important to hide this complexity. Hence, C^4S provides a high degree of automation.

Although C^4S is just under development, several functionalities have been already implemented. The system can create virtual machines provided by Amazon Web Services, Google Compute Engine and OpenStack and utilize them as nodes for the elastic platform (currently we have chosen Kubernetes as cluster type representative for the first deployer release). Decreasing the cluster by terminating virtual machines is possible, too. The next step is the creation of a deployment language for dedicated containers to run on a Kubernetes container cluster, finding solutions for container cluster scaling problems and handling stateful tasks like file storage. The system will be implemented in a modular and generic way to allow an adaptation for different cloud platforms and container cluster software. With C^4S, SMEs will be able to deploy and operate their container applications on an elastic, auto-scaling and load balancing multi-cloud cluster with transferability features to prevent vendor lock-in.

ACKNOWLEDGMENT

This research is funded by German Federal Ministry of Education and Research (Project Cloud TRANSIT, 03FH021PX4). The authors thank the University of Lübeck (Institute of Telematics) and fat IT solution GmbH (Kiel) for their support of Cloud TRANSIT.

REFERENCES

[1] P.-C. Quint and N. Kratzke, "Overcome Vendor Lock-In by Integrating Already Available Container Technologies - Towards Transferability in Cloud Computing for SMEs," in 7th. Int. Conf. on Cloud Computing, GRIDS and Virtualization (CLOUD COMPUTING 2016), 2016, pp. 38–41.

[2] European Commission, "Commission recommendation (2003/361/ec) of 6 may 2003 concerning the definition of micro, small and medium-sized enterprises," 2003. [Online]. Available: http://www.eur-lex.europa.eu/LexUriServ/site/en/oj/2003/l_124/l_12420030520en00360041.pdf

[3] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, 2010, pp. 50–58.

[5] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011, p. 49.

[6] N. Kratzke, "Lightweight virtualization cluster - howto overcome cloud vendor lock-in," Journal of Computer and Communication (JCC), vol. 2, no. 12, oct 2014.

[7] R. Sahandi, A. Alkhalil, and J. Opara-Martins, "Cloud computing from smes perspective: A survey-based investigation," Journal of Information Technology Management, vol. 24, no. 1, 2013, pp. 1–12.

[8] J. Turnbull, The Docker Book: Containerization is the new virtualization. James Turnbull, 2014.

[9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in Proceedings of the Tenth European Conference on Computer Systems. ACM, 2015, p. 18.

[10] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[11] Definition recommendation of micro, small and medium-sized enterprises by the european communities. Last access 12th Nov. 2016. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2003.124.01.0036.01.ENG

[12] N. Kratzke, "A lightweight virtualization cluster reference architecture derived from open source paas platforms," Open J. Mob. Comput. Cloud Comput, vol. 1, 2014, pp. 17–30.

[13] "Cloud Brokerage - A single portal for all IaaS providers," T-Systems, Tech. Rep., 2014, whitepaper. [Online]. Available: https://www.t-systems.com/whitepaper/77588/dl-wp-cloud-brokerage-tech.pdf

[14] European Union, "User guide to the sme definition," 2015. [Online]. Available: http://ec.europa.eu/DocsRoom/documents/15582

[15] Eurostat, "Annual enterprise statistics by size class for special aggregates of activities (NACE Rev. 2)[Code: sbs_sc_sca_r2]," 2016.

[16] B. J. Pine, Mass customization: the new frontier in business competition. Harvard Business Press, 1993.

[17] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica et al., "Above the clouds: A berkeley view of cloud computing," 2009.

[18] J. Bezos, "Amazon. com ceo jeff bezos on animoto," Startup School, 2008.

[19] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical review of vendor lock-in and its impact on adoption of cloud computing," in Information Society (i-Society), 2014 International Conference on. IEEE, 2014, pp. 92–97.

[20] N. Kratzke and R. Peinl, "ClouNS - A Reference Model for Cloud-Native Applications," in Proceedings of 20th. International Conference on Enterprise Distributed Object Computing Workshops (EDOCW 2016), 2016.

[21] V. Andrikopoulos, C. Fehling, and F. Leymann, "Designing for cap - the effect of design decisions on the cap properties of cloud-native applications," in Proceedings of the 2nd International Conference on Cloud Computing and Services Science, 2012, pp. 365–374.

[22] S. Garcia-Gomez, M. Escriche-Vicente, P. Arozarena-Llopis, F. Lelli, Y. Taher, C. Momm, A. Spriestersbach, J. Vogel, A. Giessmann, F. Junker et al., "4caast: Comprehensive management of cloud services through a paas," in Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications. IEEE Computer Society, 2012, pp. 494–499.

[23] S. Newman, Building Microservices. " O'Reilly Media, Inc.", 2015.

[24] ecp_deploy. Last access 30th Nov. 2016. [Online]. Available: https://rubygems.org/gems/ecp_deploy

[25] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," SIGOPS Oper. Syst. Rev., vol. 41, no. 3, Mar. 2007, pp. 275–287.

[26] N. Kratzke and P.-C. Quint, "How to Operate Container Clusters more Efficiently? Some Insights Concerning Containers, Software-Defined-Networks, and their sometimes Counterintuitive Impact on Network Performance," International Journal On Advances in Networks and Services, vol. 8, no. 3&4, 2015.

[27] N. Kratzke and P.-C. Quint, "ppbench - A Visualizing Network Benchmark for Microservices," in Proceedings of 6th. International Conference on Cloud Computing and Service Sciences (CLOSER 2016), 2016.

[28] N. Kratzke and P.-C. Quint, "Investigation of Impacts on Network Performance in the Advance of a Microservice Design," in Cloud Computing and Services Science Selected Papers, ser. Communications in Computer and Information Science (CCIS), M. Helfert, D. Ferguson, V. M. Munoz, and J. Cardoso, Eds. Springer, 2016.

[29] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," Proceedings of CLOUD COMPUTING, vol. 2015, 2015.

[30] N. Kratzke and P.-C. Quint, "About Automatic Benchmarking of IaaS Cloud Service Providers for a World of Container Clusters," Journal of Cloud Computing Research, vol. 1, no. 1, 2015, pp. 16–34. [Online]. Available: http://jccr.uscip.us/PublishedIssues.aspx

[31] C. Pahl, L. Zhang, and F. Fowley, "Interoperability standards for cloud architecture," in 3rd International Conference on Cloud Computing and Services Science, (CLOSER 2013), 2013, pp. 8–10.

[32] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: a systematic review," Cloud Computing, IEEE Transactions on, vol. 1, no. 2, 2013, pp. 142–157.

[33] C. Baun, M. Kunze, and V. Mauch, "The koala cloud manager: Cloud service management the easy way," in Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011, pp. 744–745.

[34] R. Peinl and F. Holzschuher, "The docker ecosystem needs consolidation," in 5Th Intl. Conf. on Cloud Computing and Services Science (CLOSER 2015), 2015, pp. 535–542.

[35] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," ACM SIGPLAN Notices, vol. 49, no. 4, 2014, pp. 127–144.