Abstract—With the development of various cloud technologies and their increased usage, the newly arisen challenge is the migration of a cloud-based application and data among various cloud environments and vendors. In the absence of a standardised mean by which full portability can be achieved, various tools exist in the form of APIs, standards, and protocols. In this paper we conduct an overview of the state of the art regarding the topic of migration and portability of cloud-based applications, in order to determine the possible challenges that today’s technologies might have, as well as to find the most appropriate automated way to port an application from one to another cloud, or to migrate an on-premise application to the cloud.

Keywords—Cloud Computing, services, standards.

I. INTRODUCTION

The increase of usage of cloud computing technologies also introduces the existence of various clouds framework softwares that operate separately. Portability implies the ability to use software components across multiple different hardware and software platforms and environments. In this paper, portability between clouds, is defined as the ability to smoothly and fluently transfer a software component, application, service or data into another cloud environment, ensuring undisturbed continuity and elasticity.

As the current trend implies the usage of cloud services, the cloud consumers should be offered the possibility to use services across different clouds, whether it is to simply migrate to another cloud or seek an on-demand scaling of resources by expanding their system onto external resources, outside the native cloud where their application or service resides [1].

Various different approaches have been taken to explore the possibilities of theoretically and practically achieving some level of portability of an application between clouds. These efforts range from scientific papers which offer ideas and overviews, to proof-of-concept implementations or models, to even commercial products. Most groups stress the importance of such a study, given the fact that portability is still one of the greatest and most important obstacles that hinder the increase of adoption of cloud technologies [2].

The aim of this paper is to produce an analysis of the state-of-the-art efforts and their approaches in cloud portability, including their structure and a study of their respective APIs. Furthermore, the paper presents a study of the structure of web and cloud based applications. Lastly, conclusions are drawn regarding the current efforts which identify the weak points in the approaches attempted today.

This paper is separated into several sections which explain the following. Section [II] elaborates the need for cloud portability and clearly outlines the difference between portability and interoperability. In addition, it outlines the architecture of web applications, as the most common type of application in the cloud computing service market, and compares it to the architecture of the cloud applications with the intention to understand the transition process during the migration process. Section [III] introduces an overview of the state-of-the-art efforts that attempt to achieve cloud portability, elaborating upon TOSCA, mOSAIC, AWS CloudFormation as well as Heat. Lastly, Section [IV] draws the conclusions, along with our plans for future work.

II. MOTIVATION AND BACKGROUND

The cloud is a rapidly developing technology that bases its greatest advantages upon the ubiquitous network access to a shared and virtualised set of hardware and software resources. These resources can be numerous and of various nature, most of which including processing, network, storage, message transport and queuing, all of which can be provisioned. The existence of various resources that make up the cloud computing model leads to a different cloud structure depending on the implementation of each of the aforementioned resources. The current software stacks of most cloud services are very heterogeneous and therefore the features each of them provide offer varying levels of incompatibility between each other. This kind of a heterogeneity is a great obstacle in the face of the demands of the cloud consumer as well as the vendor, and can be both seen as an issue with portability and interoperability [3].

A. Difference between cloud portability and interoperability

Cloud portability and cloud interoperability are very frequently seen as synonyms that share the same concept, mainly because numerous solutions solve both, even though originally have been intended for one of them. Still, these two concepts have their differences which will be elaborated below.

Cloud interoperability ensures the simultaneous and mutual work of diverse cloud services, data and application. Primarily it is a concern among cloud service providers, who must offer fluent functionality of the services they offer with the external world, be it public clouds or other cloud providers. This kind of a concept implies a previously specified knowledge of both systems, including agreements upon which the systems will interoperate. While the cloud service providers work to maintain
the interoperability between the systems, the cloud customers are usually entirely unaware of these added functionalities.

On the other hand, cloud portability is a concept that also concerns both the cloud customer and provider. By definition, it deals with the minimization of the human efforts when migrating an application, data or service from one cloud to another, which in turn lessens the need for re-designing and re-writing parts of the application. This way, cloud consumers are able to not only run instances of their application on multiple clouds, but also use external resources [4].

B. Motivation

1) Economic reasons: Portability within the cloud is necessary from an economic aspect for several reasons. On one hand, the customers of the cloud would gain protection over their investments in their application developments. Due to the heterogeneity of the cloud, the migration of a software application, along with its data, from one cloud to another often requires the rewriting of large parts of both the services and the application, all in order to comply with the new environment to which the application is migrating. On the other hand, the cloud service providers would be interested in enhancing the opportunity for application portability in order to promote their own attractiveness within a highly dynamic and demanding market. In addition, the need for portability also allows the development of third party organisations which would be able to mediate between the cloud customer as well as multiple cloud providers. This kind of arrangement would enable deployments depending on the customer’s requirements, and all kinds of necessary adaptations would be provided by the third party.

2) Technical reasons: From a technical point of view, cloud portability is of immense importance in order to exploit cloud elasticity to a maximum extent, as well as to ensure the continuity in a given application along with its functionality of a service. Throughout a lifetime of an application, a point might come at which external resources are required from a public cloud, and so it needs to be redeployed from a private cloud. This kind of a porting process is triggered by the cloud customer. Furthermore, in order to achieve continuity, precaution measures such as regular back-ups must be made, not only of data, but of application instances as well. This implies that an application must be able to use various cloud services which in turn will aid the replication of the application as well as its data. This kind of a porting process is most commonly in accordance to the provider’s agreements, and is usually considered in the design phase.

3) Legal reasons: The laws within a country may change rather abruptly, triggering the need for software to be deployed from one cloud system to another. A cloud service provider may also provide faulty service which in turn would call for immediate redeployment of applications on another cloud platform. In this sense, efficient portability mechanisms are essential for fast recovery. Situations like the aforementioned one are also attempted to be avoided by creating contracts which guarantee a sufficient contingency plans if the cloud provider fails. These can only be provided if portability between various cloud services exists. In addition, contracts between parties may also demand the usage of certain metrics which measure the quality of the cloud’s performance, including its portability, which in turn reveals the demand for a metric system that can assess the level of portability within a cloud [5].

C. Components of cloud applications

While the applications in the cloud computing service market range in nature, the most successful ones are the web applications. Understanding the differences between web applications and Cloud applications is essential for the development of a solid transition process from web applications to cloud applications.

The structure of a traditional web 2.0 application heavily relies upon a three layer architecture which allows numerous benefits, as well as drawbacks in some instances. For example, the separation of the layers into presentation, logic and data outlines the application into parts which interact with each other, but are essentially disjunct Fig. 1. If a certain application is made up of several components, then this kind of a separation in layers greatly aids the connectivity between the components, allowing layers to interact with their respective layer. On the other hand, this architecture runs into a problem when the need for scalability and elasticity escalates. The design patterns that are used in its development are most commonly adapted for a limited number of users. The data in these applications is also often used in databases which have difficulties scaling or replacing in the events of failure, and any change in their schema requires the disruption of the continuity of the application life.

A cloud application is expected to have an unpredictable load and a much larger user base that would exploit the elasticity of a system. The transformation of a web application into a cloud application requires a general architecture of a cloud application. Petcu et al. [7] propose an architecture form presented in Fig. 2. There are numerous commercially available APIs that aid the developers with the building of applications. Some of which attempt to hold to the outline of the aforementioned multi-layer cloud application architecture. TOSCA and mOSAIC specifically offer APIs which facilitate the development of portable and interoperable cloud applications.
III. OVERVIEW OF CURRENT STANDARDS

There have been numerous efforts at researching as well as implementing some of the standards that are already out on the market, most of which free to use and require no royalty compensation if applied. Most of these are independent projects developed by either organisations or companies and have not yet been widely adopted [8] [9] [10].

In this section several of these solutions are elaborated, describing their structure and methodology, as well as the basic outline of their APIs and how they can be used. Some of them are presented in greater detail with the aim to portray the contact points with the developer, explaining in detail how their architecture aids the portability and interoperability of cloud applications.

A. CAMP

One of the very few steps that need to be taken towards an enhanced interoperability is the unification of management interfaces across several cloud infrastructures. OASIS proposes the Cloud Application Management for Platforms (CAMP) standard, which is a specification designed to ease the management of applications across PaaS. CAMP specifies a RESTful self-service application and a platform management API, which is neutral in several aspects, including the language, framework as well as platform. Its expectations primarily lie in the creation of an ecosystem filled with common tools, plugins, libraries and framework, which then offer vendors increased performance when it comes to interoperability between clouds [11].

The aforementioned CAMP API is made up of a resource model and a protocol that is free to manipulate the resources remotely. The resources contained in the resource model are generally separated into a Platform, Assemblies, Platform Components and Application Components. Excluding the Platform, all of these resources can exist in an instantiated and in a deployed form. The platform type of resource describes a PaaS as a whole and references all applications on the platform, allowing the discovery of the other existing Platform Components. These platform components can be database services, web services etc. An Application Component is an artefact, such as source code or metadata which can use other Application Components, depending on its respective Platform Component. Finally, an Assembly is a resource that represents an application by referencing the components it is composed of, allowing management at runtime [12].

B. TOSCA

TOSCA, Topology and Orchestration Specification for Cloud Applications, is created with the aim to enhance the portability specifically for cloud applications and services. TOSCA enables the interoperable description of applications and the infrastructure cloud services, the relationship between various components of the service, as well as the operational behaviour of these services. All of the aforementioned are independent of the supplier that creates the service, including any particular cloud provider or hosting technology. OASIS ensures that increasing the service and application portability within a vendor-neutral ecosystem, TOSCA enables a portable deployment to any compliant cloud, easier and more intuitive migration of existing applications to the cloud as well as offering dynamic, multi-cloud provider applications.

As opposed to CAMP’s limited functionalities that deal with portability and orchestration, TOSCA is superior as it proposes a more comprehensive approach. This industrially-endorsed specification allows the software components of an application along with their respective relationship and management procedures to orchestrate basic operational behavior (deployment, patching, shut down), using descriptions based on the specified XML-based language which ensures a high level of universality and interoperability [13].

TOSCA introduces Service templates which are a concept used to specify the topology and orchestration of IT services. This mainly refers to the structure and invocation of management behavior within the service. The general structure of a Service template can be seen in Fig. 3 encapsulating a Topology Template which describes the structure of an application. In addition, this Topology template is a directed graph consisted of a set of Node Templates and Relationship Templates.
A node in this graph is represented by a Node Template which specifies the occurrence of a Node Type as a component of a service. A Node Type defines the properties of these components and the operations available that have the opportunity to manipulate the component.

A Relationship Template, on the other hand, specifies the occurrence of a relationship between the nodes in the Topology template. Each Relationship Template contains a Relationship Type which defines the nature of the relationship as well as any properties that belong to it. These are most commonly defined separately so they may later be reused [14].

The first available open source tool was first published in 2013 through the OpenTOSCA initiative. A modelling tool called Winery [15] was published, which allows the definition of a cloud application topology using a web GUI. An initial version of the TOSCA container infrastructure was also released. Kostoska et al. [16] pointed the benefits and drawbacks of TOSCA specification. The P-TOSCA was introduced as an extension of TOSCA to overcome several TOSCA’s drawbacks. Using P-TOSCA, porting of a small SOA, [17], [18] and N-tier application [19] was conducted.

Though as a relatively new standard, TOSCA has been offered for implementation on a free basis, and various companies, non-profit organisation, academic institutions, individuals including governments have already attempted to implement their applications using this specification. Numerous companies which are world leaders in the IT industry, such as ASG, Fujitsu, HP, IBM, SAP and others, congratulate this specification as an advancing of higher levels of interoperable and portable cloud solutions [20].

C. mOSAIC

mOSAIC is an open source API and platform that ensures the portability and interoperability between multiple clouds. While the API is the key element that ensure smooth portability, the platform offers numerous features, which aid the portability of an application.

In short, mOSAIC’s intention is to aid the cloud application developer build an application that is easily portable between different clouds, as well as offer guidance as to which are the proper resources required to run an application. Moreover, mOSAIC also offers a software platform including a set of application tools, a semantic engine, a cloud agency as well as a service discoverer. The main issue that mOSAIC attempts to resolve is the one of the developer of the cloud application. Instead of worrying about implementing the application for a specification of a particular cloud, the developer can only focus on the development of the application using mOSAIC’s API, and later on deploy the application on any cloud the developer wishes. Therefore, the application’s portability and elasticity are ensured and the developer has a choice of a programming language of implementation.

mOSAIC offers several layers of abstraction which in turn ease the development of cloud applications. Separating the abstraction into various layers, the developers are able to assemble an application based on several components with defined properties and relationships, previously described by an application descriptor which is used in the lower ends of the deployment process. An Eclipse plug-in, if the developer uses Java, for instance, will define the graphical interface of the application, generating its own code which is then passed onto higher levels of development. In addition, a layer of abstraction is also designed for the owners of existing applications who intend to deploy the application onto other clouds. First parts of the application must be rewritten to comply with mOSAIC’s requirements, which in turn allows for the smooth deployment onto multiple clouds. Lastly, two separate layers of abstraction deal with debugging, testing and manual control of the resources, as well as billing purposes [10].

Fig. 4 presents the layers of the mOSAIC APIs intended to support the cloud components’ development and communication. Applications that use mOSAIC are made up of configurable Cloud Building Blocks which have the ability to communicate between each other. These building blocks can be either a Cloud Resource or a Cloud Component. Cloud components are controlled by the application developer and follow the implemented behaviour which defines its functionalities.

As shown in Fig. 4, Cloudlet APIs are language dependent and provide standard programming components to the developer. The Connector API provides an abstraction for the cloud resources. Depending on the chosen programming language the developer of the application may interact with this API, or not at all. This API ensures the uniformity of the programming paradigms, assuming that the implementation of the connector API in an object oriented programming language generally have similar hierarchies and patterns. The Interoperability API as a member of the Cloudware layer aims to provide interoperability on programming language level, enforcing protocol syntax as well as semantics. Most often it enables Connectors to invoke operations that are enabled in the Driver API, without any knowledge of the technology.
it supports. The Driver API, also known as a wrapper, has the role to wrap the native API and provide the initial level of uniformity. All cloud resources of the same type are processed and exported with the same interface. The Native resource API is most often a library provided by the cloud vendor and is usually language-dependent. This layer provides a very low level of uniformity.

In order to develop a fully mOSAIC compliant application, the developer is required to use the Cloudlet API which would ensure the scalability of the application, offer improved fault tolerance, and provide autonomy, in the sense that the components will be able to run in a cloud environment independent of other components. A Cloudlet Container is the host of a single Cloudlet, however it is possible to host several running instances of the same Cloudlet within the same Container. Each of these containers has a unique identifier which enables its identification during runtime. Therefore, it is not possible to distinguish between the different instances running in the same Container. Still, as a general rule, mOSAIC ensures that the functional behaviour of the Cloudlet must be independent from the number of instances or Containers [21].

D. AWS CloudFormation

AWS CloudFormation is an orchestration approach developed by Amazon, and as such it acts as a proprietary to the Amazon infrastructure. Using JSON templates, it describes a set of AWS resources, forming a stack. This stack, once created, can be managed as an entity. The AWS CloudFormation is able to automatically resolve dependencies. In addition, its primary aim is to orchestrate the creation of AWS resources which have previously been declared by the JSON templates, with the ability to connect these resources afterwards. This way, there is no need to specify management or deployment processes [22].

The AWS CloudFormation implies the work with templates and stacks. Templates are used to describe the user’s AWS resources as well as their properties. This is most commonly done by saving the template with any extension compliant with the JSON standard (.json, .template, .txt), which can then be read by the CloudFormation and considered a guideline for building the specified AWS resource.

A stack, on the other hand, unifies and therefore manages related resources. The user can create, update and delete a set of resources by creating stacks and organising their resources within them. By modifying the stack or applying any changes to it, the resources that belong to the collection assigned to the respective stack are also changed [14].

E. HEAT

As one of the most popular cloud middleware software, OpenStack suggests its program, Heat, aimed to deal with orchestration. Heat is portable between OpenStack clouds, and it enables declarative infrastructure provisioning. It implements a cross-compatible AWS CloudFormation for OpenStack clouds, using multiple languages for describing templates. It provides a template based orchestration that describes cloud applications by executing appropriate OpenStack API calls which in turn generate cloud applications [23].

Like the AWS CloudFormation, Heat also introduces the usage of stacks and templates. These templates allow the creation of most OpenStack resource types, as well as some more advanced functionality such as instance autoscaling or nested stacks.

Heat generally uses two templates: HOT (Heat Orchestration Template) and CFN (short for AWS CloudFormation). The HOT template can only be used with OpenStack and is not backwards compatible with AWS CloudFormation templates. HOT templates are typically required to be expressed in YAML. CFN, on the other hand, are templates normally expressed using JSON [24].

F. IEEE P2301

The IEEE P2301 - Guide for Cloud Portability and Interoperability Profiles (CPIP) standard is a guideline written by the IEEE Computer Society. Its purpose is to assist cloud computing vendors as well as users in the processes of producing cloud systems with increased portability and interoperability based on standardized means and services. The guideline identifies various elements that Cloud Computing Systems possess and provides an enumerated group of portability, management, interoperability interfaces, file formats and conventions used for each of these elements. The guideline groups the suggested enumerations into logical entities which in turn increases the level of portability and commonality [25].

G. Unified Cloud Interface

Recognizing the high level of diversity among the numerous approached towards a standardized mean that increases Cloud portability, the Unified Cloud Interface Project (UCIP) has the goal of producing an API that uses all other API’s that the Cloud systems provide as external interfaces. Since independent cloud systems provide both similarities as well as difference in the way fundamental and additional functionalities are implemented, the UCIP attempts to arrive at a certain level of abstraction that’s beyond the API of the Cloud API. The resolution of the heterogeneities between the cloud APIs and the existing protocols and standards is achieved through the usage of various ontologies as well as the Web Ontology Language (OWL) [26].

H. Open Cloud Computing Interface

With the establishment of the several cloud offerings, spanning the stack, ranging within the Information as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The Open Cloud Computing Interface (OCCI) comprises a large set of open specifications, offering a Protocol and an API for different types of management tasks. Initially the scope of the project were simply IaaS model based Services, however it has developed into a rather flexible API with a focus on interoperability, innovation, integration and portability, while still holding scalability in a high regard.
Currently, all service models are covered by the API. Due to its high level of flexibility, many academic and industry giants currently support the OCCI.

IV. CONCLUSION AND FUTURE WORK

Cloud portability is a well-recognized problem ever since the beginning of the cloud computing technologies. Different approaches for portability across multiple cloud platforms already exist, some of which in the form of standards, while others as platforms or APIs, proof-of-concept attempts etc.

Web applications are the most common and successful type of applications. Their structure heavily relies upon a three layer architecture that consists of a Presentation layer, a Business layer and a Data layer. The cloud application architecture assumes the existence of a Presentation Layer, a Load Balancer, a Business Layer, a Data Layer as well as Message Queues. The Load balancing layer acts as a distributor of requests through instances of the application in order to avoid overload, while the Message queues serve as communication pathways between each of the layers.

None of the existing approaches are widely adopted for several reasons. Firstly, most of them focus on a specific topic and attempt to excel achieving completeness in certain aspects of interoperability and portability. Secondly, the majority of cloud providers are reluctant to share their work as it provides a crucial advantage in their companies on the market. Lastly, despite the numerous attempts and approaches, there is no solution yet that envelops all aspects of portability and migration due to the great versatility of applications, platforms, services and languages.

Several additional approaches can be relevant for the future advancement in the field of cloud portability and interoperability. For instance, the introduction of global standards will enhance the number of implementations compliant with these standards, therefore increasing the likelihood of a unified portability solution. In addition, these standards can be made up to establish metrics, monitoring and security. TOSCA and CAMP are newly emerged standards which still have an increase of users, while ISO/IEC 27017 is a standard for the information security management for cloud systems and is currently in preparation [27]. Furthermore, the introduction of a limitation in the sphere of an architectural design for these portability solutions would help immensely with reducing the heterogeneity between each other. Limiting the architecture to several fixed levels of abstraction will provide greater room for automated and flawless portability, while reducing the room to implement exotic features which would hinder the smoothness of the solution.

REFERENCES