Architecting Enterprise Applications for the Cloud: The Unicorn Universe Cloud Framework

Marek Beranek[1], Marek Stastny[1], Vladimir Kovar[1], George Feuerlicht[2]

[1]Unicorn College, V Kapslovně 2767/2,130 00 Prague 3, Czech Republic
[2]Prague University of Economics, W. Churchill Square. 4, 130 67 Prague 3, Czech Republic

marek.beranek@unicorncollege.cz,
marek.stastny@unicornuniverse.eu,
vladimir.kovar@unicorn.eu,
george.feuerlicht@gmail.com

Abstract. Recent IT advances that include extensive use of mobile and IoT devices and wide adoption of cloud computing are creating a situation where existing architectures and software development frameworks no longer fully support the requirements of modern enterprise application. Furthermore, the separation of software development and operations is no longer practicable in this environment characterized by fast delivery and automated release and deployment of applications. This rapidly evolving situation requires new frameworks that support the DevOps approach and facilitate continuous delivery of cloud-based applications using micro-services and container-based technologies allowing rapid incremental deployment of application components. It is also becoming clear that the management of large-scale container-based environments has its own challenges. In this paper, we first discuss the challenges that developers of enterprise applications face today and then describe the Unicorn cloud framework (uuCloud) designed to support the development and deployment of cloud-based applications that incorporate mobile and IoT devices. We use a doctor surgery reservation application "Lekar" case study to illustrate how uuCloud is used to implement a large-scale cloud-based application.

Keywords: cloud computing, software frameworks, micro-services, DevOps.

1 Introduction

The basic enterprise computing objectives that include low cost, reliability, security, scalability, easy deployment and usability have not changed dramatically for decades. However, the hardware and software environment in which enterprise applications are being developed and deployed today is significantly different from that only a few years ago. Enterprise architectures have been evolving to take advantage of the opportunities offered by advances in hardware and software technologies, in particular the increase in processing power and storage capacity and the corresponding cost reductions. The centralized architectures of the 1970s were superseded by client/server architectures of the 1980s, and component-based architectures of the 1990s, and most recently by the

Service-Oriented Architecture (SOA) at the beginning of this century. We are now experiencing another major transformation driven by wide adoption of cloud computing and extensive use of mobile and IoT (Internet of Things) devices. Public cloud platforms (e.g. AWS [1], Microsoft Azure [2], etc.) offer highly elastic and practically unlimited computing power and storage capacity, allowing more flexible acquisition of IT resources in the form of cloud services, overcoming the limitations of on-premise IT solutions. Importantly, this new technology environment is creating opportunities for innovative solutions at a fraction of the cost of traditional enterprise applications. However, to take full advantage of these developments, organizations involved in the development of enterprise applications must adopt a suitable enterprise architecture and application development frameworks. The architecture needs to support various types of mobile devices (smart phones, tablets, etc.) and incorporate interfaces that interact with IoT devices. Unlike traditional enterprise applications that store data on local servers within the organization, most mobile applications store data and deploy application components in the cloud, making it possible for applications to be shared by very large user populations. Given the requirements of modern business environments, the architecture needs to facilitate rapid incremental development of application components, secure access to information and easy and fast cloud deployment. There is now increasing empirical evidence that to effectively address such requirements, the architecture needs to support micro-services and container-based virtualization [3]. However, it is also becoming clear that the management of large-scale container-based environments has its own challenges and requires automation of application deployment, auto-scaling and control of resource usage. The need for continuous delivery and monitoring of application components impacts on the structure and skills profile of IT teams, favoring small cross-functional teams leading to the convergence of development and operations (DevOps). The separation of code development and declarative methods of environment configuration play an important role in increasing the productivity of the software development process. Furthermore, developers of enterprise applications are increasingly turning towards open source solutions that allow full control over the entire software stack, avoiding costly proprietary solutions. Also, while the use of public cloud platforms is economically compelling, an important function of the architecture is to ensure independence from individual cloud providers, avoiding a provider *lock-in*. Finally, the architecture should reduce the complexity of the application development and maintenance process and facilitate effective reuse of application components and infrastructure services.

These requirements demand a revision of existing architectural principles and application development methods. In this paper, we describe the Unicorn Universe Cloud Framework (uuCloud) designed to facilitate the management of modern container-based cloud environments addressing the issues identified above. The uuCloud framework is an integral part of the Unicorn Application Framework (UAF). We have described the features of the UAF in an earlier publication [4], giving a high-level overview of the architecture. In this paper, we focus on the uuCloud framework and describe its key components and operation. In the next section (section 2) we review related literature focusing on DevOps, micro-services and container management frameworks. In the following sections, we briefly overview the UAF architecture (section 3) and describe the uuCloud framework (section 4). Section 5 illustrates the application of the uuCloud framework using an example of the Doctor's Surgery Reservation System (Lekar Reservation System - LRS). Section 6 includes our conclusions and directions for future work. Please, note that all diagrams in this paper are drawn using the Unicorn Business Modeling Language: https://unicornuniverse.eu/en/uubml.html.

2 Related Work

Cloud-based application development frameworks and architectures have been the subject of intense recent interest by industry practitioners and academic researchers, in particular in the context of micro-services and DevOps [5]. As noted in section 1, several recent trends including cloud computing, extensive use of mobile and IoT devices have impacted on the architecture of enterprise applications with corresponding impact on application development frameworks [6]. Namiot et al. [7] discuss the advantages and drawbacks of micro-services. The benefits of micro-services include the ability to use different programming languages for individual services, improved scalability and more rapid, incremental development within smaller teams. However, they also note that the use of a larger number of smaller services increases deployment complexity. One of the most important challenges involves decisions about how to partition the application system into micro-services, i.e. making decisions about service granularity. The paper discusses various service partitioning methods including the Scale Cube [8], partitioning by use-cases and partitioning by resource type. Armin Balalaie et al. [3], [9] consider achieving reusability, decentralized data governance, automated deployment and built-in scalability to be the main motivations for migration to a micro-services architecture. The authors report on their experiences during incremental migration and architectural refactoring of a commercial "Mobile Backend as a Service" to micro-services. They argue that standard virtualization methods introduce a heavy computational overhead and therefore are not cost-effective, and recommend the use of container-based virtualization to reduce overheads and to enable portability. The authors discuss synergies between micro-services and DevOps approach that involves small vertically structured cross-functional teams responsible for individual application components and services. They conclude that the main lessons learned from the migration to micro-services include: 1) the critical importance of service contracts as the number of services increases, 2) the need for skilled developers who understand distributed systems development, and 3) use of development templates. According to Rimal et al. [10] the most important current challenge is the lack of a standard architectural approach for cloud computing. The authors explore and classify architectural characteristics of cloud computing and identify several architectural features that play a major role in the adoption of cloud computing. The paper provides guidelines for software architects for developing cloud architectures. According to Raj et al. [11] "The urgent thing is to embark on modernizing and refining the currently used application development processes and practices in order to make cloud-based software engineering simpler, successful, and sustainable." The authors argue that software development has become an inherently complicated task and that a systematic, disciplined, and quantifiable approach is essential to make software development more manageable and to produce quality software products. A new requirements engineering process and techniques for capturing requirements for cloud-based services was proposed and illustrated using a large-scale case study based on Amazon Cloud EC2 [12]. Adaptation of the software development life cycle for cloud computing has been the subject of recent research interest. The differences between cloud service provider and consumer SDLC life-cycles resulting from the use of externally provided cloud services have been identified [13]. The authors describe a Service Consumer Framework (SCF) that incorporates architectural extensions designed to support operation in cloud computing environments [14].

While container technologies and micro-services have revolutionized application development and deployment, it is also evident that the use of these technologies has its limitations. More specifically, the management of large-scale container-based environments requires automation to ensure fast and predictable application deployment, auto-scaling and control of resource usage. At the same time, there is a requirement for portability across different public and private clouds. To address such issues a number of open source projects have been recently initiated; prominent examples include Cloud Foundry [15], OpenShift [16] and Kubernetes [17]. These projects share many common concepts and in some cases technologies. A key idea of these open source platforms is to abstract the complexity of the underlying cloud infrastructure and present a well-designed API (Application Programming Interface) that simplifies the management of container-based cloud environments. The Kubernetes project initiated by Google in 2014 as an open source cluster manager for Docker has its origins in an earlier Google container management system called Borg [18]. The Kubernetes project is hosted by the Cloud Native Computing Foundation (CNCF) [19] that has a mission "to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self-healing multi-tenant nodes". The objective is to facilitate *cloud native* systems that run applications and processes in isolated units of application deployment (i.e. software containers). Containers implement micro-services which are dynamically managed to maximize resource utilization and minimize the costs associated with maintenance and operations. CNCF promotes well-defined APIs as the main mechanism for ensuring extensibility and portability. A basic Kubernetes building block is a *Pod* - a REST object that encapsulates a set of logically connected application containers with storage resources (Volumes) and a unique IP address. Pods constitute a unit of deployment (and a unit of failure) and are deployed to *Nodes* (physical or logical machines). Lifetime of a Volume is the same as the lifetime of the enclosing Pod allowing restart of individual containers without the loss of data. Pods are externalized as *Services*; Kubernetes service is an abstraction that defines a logical set of Pods and a policy for accessing the Pods (i.e. micro-service). *Replication Controller* is used to create replica Pods to match the demand of the application and provide auto-scaling. Kubernetes uses *Namespaces* to partition resources allocated to different groups of users. The application of Docker and Kubernetes container architecture to multi-tenant SaaS (Software as a Service) applications has been investigated and assessed using SWOT (Strength, Weakness, Opportunities and Threats) analysis and contrasted with developing SaaS applications using middleware services [20]. The authors conclude that more research is needed to understand the true potential and risks associated with container orchestration platforms such as Kubernetes.

While Kubernetes appears to be gaining momentum at present with support for major public cloud platforms including Google Cloud Platform, Microsoft Azure and most recently AWS, there is a number of other projects that aim to address the need for a universal framework for the development and deployment of cloud applications, including the Unicorn Universe Cloud framework described in this paper. While this rapidly evolving area is of active research interest to both academia and industry practitioners, currently there is a lack of agreement about a standard application development framework designed specifically for cloud development and deployment. Moreover, some proposals lack empirical verification using large-scale real-life applications.

3 Unicorn Application Framework (UAF)

The Unicorn Application Framework (UAF) developed by Unicorn (https://unicorn.com/) supports the design, development and operation of enterprise applications. A key UAF architectural objective is to support various types of mobile and IoT devices and to facilitate cloud deployment of enterprise applications utilizing standard framework services that include security and authentication services and support for multi-language environments. This minimizes the programming effort, improves reliability of applications and allows application developers to focus on the functionality that directly supports business processes and adds value for the end users. UAF architecture consists of four frameworks: uuUserInterface (uu5) - framework services for the development of Graphical User Interfaces (GUIs) based on HTML5 [21], uuIoT - framework services for the management of IoT devices, uuCloud - framework services for provisioning of elastic cloud services and uuAppServerKit - framework services for the development of application components (i.e. REST micro-services).

3.1 Unicorn Universe Application (uuApp)

UAF provides environment for the implementation and deployment of uuApp applications. uuApp is a component that implements a cohesive set of application functions designed to solve a set of specific user requirements. uuApp application is composed of sub-applications (uuSubApp) - independent units of functionality that implement specific business functions (e.g. booking a visit to a doctor’s surgery). Each sub-application is implemented as a logical application server (uuAppServer) and is typically associated with a structured (uuAppObjectStore) or a binary (uuAppBinaryStore) data store. We made an architectural decision to associate each sub-application with a single *logical* application server to ensure fast access to persistent data and to maintain security and consistency of the underlying data sources. Using this approach, the access to underlying data sources is controlled by the application server, ensuring that only authorized users can access the data. To improve scalability, individual use-cases (business functions) may be distributed across separate application servers in the form of individually addressable SPPs (Separately Performing Parts) modules.

The UAF follows the View-Model-Controller pattern, implementing the Model component in the form of an application server (uuAppServer) and the Controller and View components of the application on the client (typically a mobile device) using the uuUserInterface framework. The application server implements application logic and externalizes an API that is accessed by application clients. Persistent objects that belong to a sub-application are grouped into application *workspaces* (uuAppWorkspaces) and identified by an Application Workspace Identifier (AWID). Each sub-application is typically assigned a separate application workspace.

4 Unicorn Universe Cloud (uuCloud)

Unicorn Universe Cloud is a framework that supports autonomic provisioning of elastic cloud services using virtual containers and servers. UAF applications are typically deployed into a hybrid cloud environment (i.e. a combination of public and private cloud) in the form of virtualized application servers. To ensure portability and to reduce overheads, the UAF uses container-based virtualization. Our preferred containerization solution is Docker [22]. Docker container virtualizes the application including a complete filesystem that contains all components needed to run the application (i.e. system tools, system libraries, etc.) ensuring that the application runs independently of the platform the container is deployed on. A sub-application is mapped to an application server which is then containerized and deployed to a virtual server, and finally to a physical server. Docker containers can be deployed to a public cloud infrastructure (e.g. AWS or Microsoft Azure) or to a private (on-premise) infrastructure (e.g. the Unicorn platform Plus4U). Using containers for virtualization also improves isolation in multi-tenant environments [3].

4.1 uuCloud Nodes

We use a generic, technology agnostic terminology, *node* instead of container to indicate that applications can be implemented using containers or virtual and physical servers. Node is a unit of deployment with hardware characteristics that include virtual CPU (vCPU) count, RAM size, ephemeral storage, etc. Nodes are classified according to *NodeSize,* e.g. M (Medium size: 1xvCPU, 1 GB of RAM, 1 GB of ephemeral storage) or L (Large size: 2xvCPU, 2 GB of RAM, 1 GB of ephemeral storage). Nodes are further classified as *synchronous* or *asynchronous* depending on the behavior of the sub-application that the node virtualizes. Nodes are grouped into *NodeSets* - sets of nodes with identical functionality (i.e. nodes that virtualize the same sub-applications). Database server virtualization does not use containers and virtualizes structured and binary storage (uuObjectStore and uuBinaryStore) into a logical object called uuAppStore deployed to a virtual server.

4.2 uuGateway

Individual containers (nodes) are typically deployed on a public cloud infrastructure (e.g. Amazon AWS or Microsoft Azure) and access data from cloud-based data stores and databases (e.g. Mongo DB [23], etc.). Fig. 1 illustrates the processing of client requests by the uuGateway. The uuGateway forwards the uuURI formatted client request to a router that passes the request to a load balancer. The load balancer selects a node from a NodeSet of functionally identical nodes, optimizing the use of the hardware infrastructure and providing a failover capability (i.e. if the node is not responsive the request is re-directed to an alternative node). uuURI is a version of a generic REST URI adapted for addressing uuApp applications. uuURI uses a standard string format to route the request to a specified gateway (e.g. Plus4U.net, etc.) and to specify which server should execute the request on behalf of the user. The URI header contains a special signed token that identifies an authorized user. The URI string has the following format:

```
https://gateway/vendor-uuApp-uuSubApp-spp/tid-asid|awid/usecase
```

where:

— gateway is the gateway address, e.g. Plus4U.net
— vendor code (e.g. Plus4U)
— uuApp – application (uuApp) code
— uuSubApp – sub-application (uuSubApp) code
— spp – optional spp (Separately Performing Parts) code within uuSubApp)
— tid – tenant identifier
— asid – identifier of a specific sub-application instance
— awid workspace identifier
— usecase – use case identifier (i.e. API method)

In the case of static server resources, the use case identifier (usecase) is replaced by resource path (resourcePath) that points to a location where the resource is located.

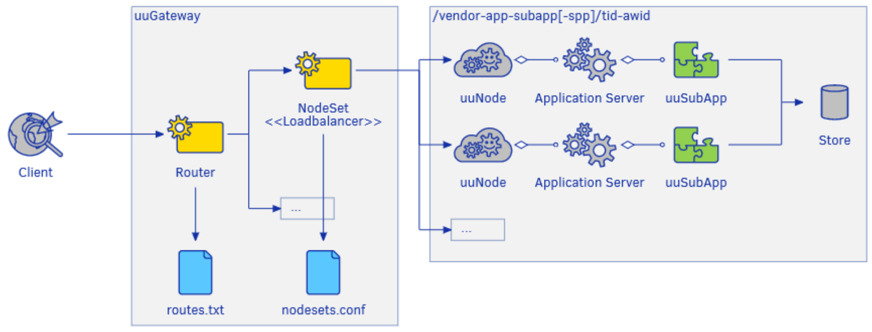

Fig. 1. uuGateway operation

4.3 uuCloud Operation Registry

Operation Registry is a key component of the uuCloud environment that maintains active information about all uuCloud objects (i.e. tenants, resource pools, regions, resource groups, hosts, nodes, etc). uuCloud supports multi-tenant operation; each *tenant* typically represents a separate organization (e.g. a doctor's surgery). Tenants are assigned *resource pools* that define the maximum amount of hardware resources (i.e. number of vCPUs, RAM size and amount of storage) that are available for their operation. The Operation Registry records information about regions (e.g. Azure North (EU-N-AZ)), resource groups and hosts, and holds information about applications deployed to each node.

4.4 uuCloud Control Centre

The *Control Centre* includes tools for deploying and running applications in the uuCloud environment. Applications (nodes) are deployed into a *territory* (i.e. tenant) that is associated with a resource pool and managed using control centre tools. The control centre verifies permissions for the deployment of applications into a specific resource pool. Control centre tools are used to manage nodes, verify free resource pool capacity, locate a suitable host for the application, and to compile and deploy the node image on selected hosts.

5 Lekar Reservation System

To illustrate the application of the uuCloud framework we use the example of a recently implemented Doctor Surgery Reservation System - Lekar Reservation System (LRS). LRS (https://www.plus4u.net/produkty-a-sluzby/lekar/) is an on-line reservation system that manages communication between healthcare professionals (doctors, nurses, medical office staff) and patients in the Czech Republic. The main objective of the system is to enable registered patients to book a visit to any participating medical practitioner at any time (i.e. 24/7) using a mobile device or a computer, without having to phone the surgery during office hours to make an appointment. The LRS application generates automatic SMS and e-mail notifications to alert the patients of an upcoming appointment, processes reservation confirmations/cancellations, and generates reminders for regular check-ups. From the point of view of the health care professional, LRS provides an integrated diary showing appointments from all surgeries and gives a quick and easy access to basic patient information. The benefit for the patient is that all appointments are recorded in an easily accessible diary. Six months after the first release of the LRS application there were 210 active healthcare professionals with thousands of patients from across the Czech Republic using the system. As these numbers are expected to grow significantly in the future, the scalability of the system is a critical design consideration.

5.1 LRS Technical Solution

The LRS application covers three main business functions implemented as cloud services: Reservations, Organization & Management (i.e. registering and de-registering doctors, patients, nurses, and office staff) and Notifications (patient notifications via email and SMS messages). Additional *Common Services* that include user identification and authorization are provided by the UAF components running on uuCloud. The LRS system is implemented using open source technologies (Java, Ruby, MongoDB, Redis, Docker, Linux, etc.). The application is deployed to a hybrid cloud environment using the on-premise Plus4U.net platform (https://www.plus4u.net/en/) in combination with MicroSoft Azure, demonstrating the technical feasibility and commercial benefits of implementing a large-scale system using open source and container-based technologies deployed on a hybrid cloud infrastructure.

5.2 Topology of the LRS solution

Each doctor surgery or medical centre is assigned to its own *business territory* (implemented as a uuCloud tenant) with a separate ObjectStore (MongoDB database instance) ensuring data security by fully isolating individual cloud tenants. Table 1 shows the current allocation of NodeSets to the three main business functions. There are 19 nodes in total (1 Large and 18 Medium size nodes); all nodes are deployed on the Azure North MicroSoft data centre. Asynchronous nodes are used for email and SMS notifications, while the reservation sub-system uses synchronous nodes to perform booking of appointments. Fig. 2 shows the topology of the LRS solution; the Application Deployment tenant is implemented on-premise in the MED-BT business territory of Unicorn Medical (Unicorn organization responsible for the development of healthcare information systems), while the runtime LRS system is deployed on the Microsoft Azure cloud platform using Docker virtualization technologies.

Table 1. Allocation of NodeSets to LRS Business Functions

Business Function	Reservations	Notifications	Organization & Management
NodeSet	1x Synchronous NodeSize L	11x Asynchronous NodeSize M	2x Synchronous 5x Asynchronous NodeSize M

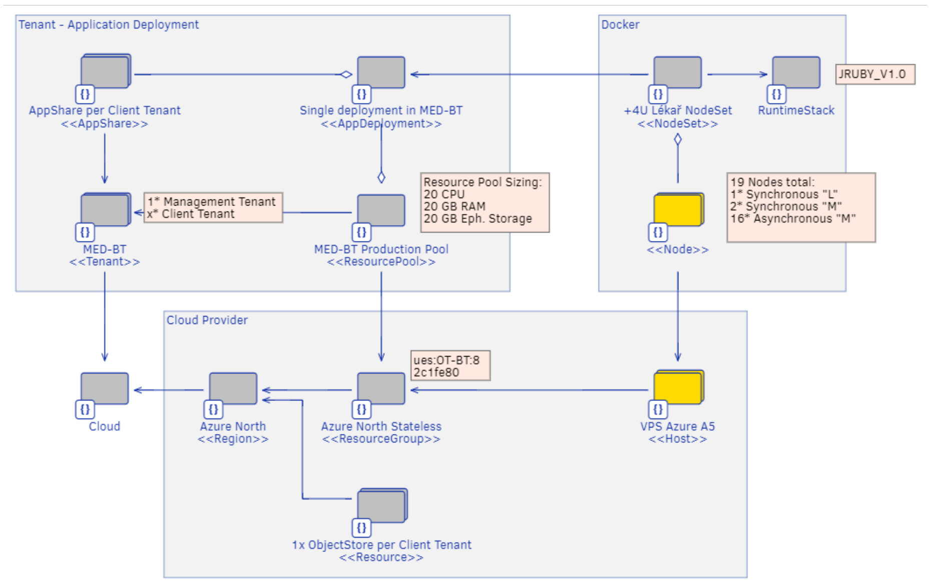
Tenant - Application Deployment
AppShare per Client Tenant <<AppShare>>
Single deployment in MED-BT <<AppDeployment>>
1* Management Tenant x* Client Tenant
MED-BT <<Tenant>>
Resource Pool Sizing: 20 CPU 20 GB RAM 20 GB Eph. Storage
MED-BT Production Pool <<ResourcePool>>
Docker
+4U Lékař NodeSet <<NodeSet>>
RuntimeStack
JRUBY_V1.0
19 Nodes total: 1* Synchronous "L" 2* Synchronous "M" 16* Asynchronous "M"
<<Node>>
Cloud Provider
Cloud
Azure North <<Region>>
Azure North Stateless <<ResourceGroup>>
ues:OT-BT:8 2c1fe80
VPS Azure A5 <<Host>>
1x ObjectStore per Client Tenant <<Resource>>

Fig. 2. Topology of the LRS cloud solution

The present version of the LRS system is the first step towards a more comprehensive outpatient system for individual medical practitioners and large healthcare facilities. The inherent scalability of this cloud solution makes it possible to add computing and storage resources as the need arises with minimal incremental cost while maintaining high availability and response time characteristics.

6 Conclusions and Further Work

We have argued that the accelerating shift towards cloud computing combined with the dominance of mobile computing and the growing use of IoT devices requires a re-assessment of the existing architectural principles and the associated application development frameworks. While the use of micro-services and container-based virtualization brings many benefits, the highly distributed nature of the resulting applications and the short software release cycles present many challenges to organizations involved in the development of cloud-based enterprise applications. A suitable application development framework and associated methods and tools are an essential pre-requisite for achieving successful project results on a repeatable basis. The uuCloud platform described in this paper was developed specifically to address the requirements of cloud-based applications and is used currently for the development of large-scale enterprise applications at Unicorn. We have illustrated the application of uuCloud using a *real-world* Doctor Surgery Reservation System that is used by hundreds of healthcare professional and thousands of patients across the Czech Republic.

There are some similarities between uuCloud and other frameworks such as Cloud Foundry, OpenShift and Kubernetes. Kubernetes, in particular supports similar functionality as uuCloud. We have evaluated these frameworks before starting the UAF project and decided that our needs would be best served with a fully integrated architecture that incorporates uuCloud, uuUserInterface and uuIoT frameworks (see UAF description in section 3). The uuCloud framework currently supports Docker containers and Microsoft Azure cloud infrastructure, but we are extending the framework to incorporate other technology solutions. Our present efforts focus on improving the monitoring tools and ensuring high availability and good response time for applications running on public cloud infrastructure. We are investigating the feasibility and cost of deploying applications across multiple availability zones, with the aim of providing users with similar SLA (Service Level Agreement) guarantees as we are able to provide for on-premise applications. We are continuously monitoring the rapidly evolving landscape of cloud platforms and frameworks. We may decide to align the uuCloud framework with Kubernetes in the future as both frameworks mature and the direction of cloud standardization becomes clearer.

7 References

1. Amazon.com. http://aws.amazon.com/. 2017 [cited 2017 7 July, 2017]; Available from: http://aws.amazon.com/.
2. *Microsoft Azure: Cloud Computing Platform & Services*. 2017 [cited 2017 22 August 2017]; Available from: https://azure.microsoft.com/en-au/.
3. Balalaie, A., A. Heydarnoori, and P. Jamshidi, *Microservices architecture enables DevOps: migration to a cloud-native architecture.* IEEE Software, 2016. **33**(3): p. 42-52.
4. Beránek, M., Feuerlicht, G., Kovář, V. *Developing Enterprise Applications for Cloud: The Unicorn Application Framework*. in *International Conference on Grid, Cloud and Cluster Computing, GCC'17*. 2017. Las Vegas, USA: CSREA Press.
5. Thönes, J., *Microservices.* IEEE Software, 2015. **32**(1): p. 116-116.
6. Mahmood, Z. and S. Saeed, *Software engineering frameworks for the cloud computing paradigm*. 2013: Springer.
7. Namiot, D. and M. Sneps-Sneppe, *On micro-services architecture.* International Journal of Open Information Technologies, 2014. **2**(9): p. 24-27.

8. *Splitting Applications or Services for Scale | AKF Partners*. 2017 22 August 2017]; Available from: http://akfpartners.com/growth-blog/splitting-applications-or-services-for-scale.
9. Balalaie, A., A. Heydarnoori, and P. Jamshidi. *Migrating to cloud-native architectures using microservices: an experience report*. in *European Conference on Service-Oriented and Cloud Computing*. 2015. Springer.
10. Rimal, B.P., et al., *Architectural requirements for cloud computing systems: an enterprise cloud approach.* Journal of Grid Computing, 2011. **9**(1): p. 3-26.
11. Raj, P., V. Venkatesh, and R. Amirtharajan, *Envisioning the cloud-induced transformations in the software engineering discipline*, in *Software Engineering Frameworks for the Cloud Computing Paradigm*. 2013, Springer. p. 25-53.
12. Ramachandran, M., *Business requirements engineering for developing cloud computing services*, in *Software Engineering Frameworks for the Cloud Computing Paradigm*. 2013, Springer. p. 123-143.
13. Feuerlicht, G. and H. Thai Tran. *Adapting service development life-cycle for cloud*. in *Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 3*. 2015. SCITEPRESS-Science and Technology Publications, Lda.
14. Feuerlicht, G. and H.T. Tran. *Service consumer framework: Managing Service Evolution from a Consumer Perspective*. in *ICEIS-2014. 16th International Conference on Enterprise Information Systems*. 2014. Portugal: Springer.
15. CloudFoundry. *Cloud Application Platform - Devops Platform | Cloud Foundry*. 2017 [cited 2017 28 September 2017]; Available from: https://www.cloudfoundry.org/.
16. *OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes*. 2017 [cited 2017 28 September 2017]; Available from: https://www.openshift.com/.
17. *Kubernetes*. 2017 [cited 2017 25 August 2017]; Available from: https://kubernetes.io/.
18. Burns, B., et al., *Borg, Omega, and Kubernetes.* Queue, 2016. **14**(1): p. 70-93.
19. *Home - Cloud Native Computing Foundation*. 2017 [cited 2017 27 September 2017]; Available from: https://www.cncf.io/.
20. Truyen, E., et al., *Towards a container-based architecture for multi-tenant SaaS applications*, in *Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware*. 2016, ACM: Trento, Italy. p. 1-6.
21. WC3. *HTML5*. 2017 21 August 2017]; Available from: https://www.w3.org/TR/html5/.
22. Docker. *What is Docker*. 2015 2015-05-14 21 August 2017]; Available from: https://www.docker.com/what-docker.
23. *MongoDB for GIANT Ideas*. 2017 21 August 2017]; Available from: https://www.mongodb.com/index.